



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING  
SHARDA SCHOOL OF ENGINEERING AND TECHNOLOGY SHARDA  
UNIVERSITY, GREATER NOIDA**

**Detection and classification of potholes in Indian Roads using YOLO**

*A project submitted  
in partial fulfillment of the requirements for the degree of Bachelor of  
Technology in Computer Science and Engineering*

**by**

**Akshat Shukla (2019006615)**

**Amartya Raj (2019563653)**

**Abhimanyu Rathore (2019522304)**

**Supervised by:**

**Dr. Sonia Setia  
Associate Professor**

**MAY, 2023**

## **CERTIFICATE**

This is to certify that the report entitled “ **Detection and classification of potholes in Indian Roads using YOLO**” submitted by **Akshat Shukla (2019006615), Amartya Raj (2019563653) and Abhimanyu Rathore (2019522304)** to Sharda University, towards the fulfillment of requirements of the degree of “**Bachelor of Technology**” is record of bonafide final year Project work carried out by them in the “Department of Computer Science & Engineering, Sharda School of Engineering and Technology, Sharda University”.

The results/findings contained in this Project have not been submitted in part or full to any other University/Institute forward of any other Degree/Diploma.

**Signature of the Guide**

**Name:** Dr. Sonia Setia  
**Designation:** Assistant professor

**Signature of Head of Department**

**Name:** Prof. Dr. Nitin Rakesh  
**Place:** Sharda University  
**Date:**

**Signature of External Examiner**

**Date:**

## **ACKNOWLEDGEMENT**

A major project is a golden opportunity for learning and self-development. We consider ourselves very lucky and honored to have so many wonderful people lead us through in completion of this project.

First and foremost, we would like to thank Dr. Nitin Rakesh, HOD, CSE who gave us an opportunity to undertake this project.

Our grateful thanks to Dr. Sonia Setia for her guidance in our project work. Dr. Sonia Setia, who in spite of being extraordinarily busy with academics, took timeout to hear, guide and keep us on the correct path. We do not know where we would have been without her help.

CSE department monitored our progress and arranged all facilities to make life easier. We choose this moment to acknowledge their contribution gratefully.

Name and signature of Students:

Akshat Shukla (2019006615)

Amartya Raj (2019563653)

Abhimanyu Rathore (2019522

# CONTENTS

<b>TITLE.....</b>	<b>i</b>
<b>CERTIFICATE.....</b>	<b>ii</b>
<b>ACKNOWLEDGEMENT.....</b>	<b>iii</b>
<b>LISTOF FIGURES.....</b>	<b>v</b>
<b>Chapter-1: Project Introduction .....</b>	<b>7</b>
1.1: Motivation.....	7
1.2: Overview.....	7
1.3: Expected outcome.....	8
1.4: Gantt Chart.....	9
1.5: Possible risks.....	10
1.6: SRS.....	10
<b>Chapter-2: Methodology.....</b>	<b>12</b>
2.1: Product / system view.....	12
2.2: System components & Functionalities.....	13
2.3: Data & relational views.....	13
<b>Chapter-3: Design Criteria.....</b>	<b>15</b>
3.1: System Design.....	15
3.2: Design Diagrams.....	16
3.3: Information & Communication design.....	17
<b>Chapter-4: Development &amp; Implementation.....</b>	<b>19</b>
4.1: Developmental feasibility.....	19
4.2: Implementation Specifications.....	20
4.3: System modules and flow of implementations.....	22
4.4: Critical modules of product/system.....	23
<b>Chapter-5: Results &amp; Testing.....</b>	<b>25</b>
5.1: Result .....	25
5.1.1: Success cases.....	35
5.1.2: Failure cases.....	35
5.2: Testing.....	36
5.2.1: Type of testing adapted.....	36
5.2.2: Test results of various stages.....	36
5.2.3: Conclusion of Testing.....	37
<b>Chapter-6: Conclusion &amp; Future Improvements.....</b>	<b>38</b>
6.1: Performance Estimation.....	38
6.2: Usability of Product / system.....	39
6.3: Limitations.....	40
6.4: Scope of Improvement.....	40
<b>References</b>	
<b>Annuxer 1</b>	
<b>Annuxer 2</b>	
<b>Annuxer 3</b>	

## **LIST OF FIGURES**

Fig.1.	Gantt Chart	9
Fig.2.	Diagram of System View	12
Fig.3.	Design Diagram	16
Fig.4.	Code for frame extraction process	25
Fig.5.	Frames stored in folder which can be further used	26
Fig.6.	Pre-trained CNN model applied to input data	27
Fig.7.	Code for storing processed images	28
Fig.8.	Difference between processed image vs original image	29
Fig.9.	Code for capturing Gps coordinates	30
Fig.10.	Code for YOLO algorithm	31
Fig.11.	Final output of algorithm	32
Fig.12.	Code for uploading data on firebase	33
Fig.13.	Uploaded data on firebase	34
Fig.14.	Code for video capturing.	44
Fig.15.	Code for video capturing.	45
Fig.16.	Code for frame conversion.	46
Fig.17.	Code for frame conversion (continuation).	47
Fig.18.	Code for uploading GPS location to Firebase.	48
Fig.19.	Code for capturing GPS locations.	49
Fig.20.	Code for GUI interface for image processing.	50
Fig.21.	Code for applying pre trained image processing model.	51
Fig.22.	Code for applying sharpening to the image.	52

Fig.23.	Code YOLO object detection algorithm.	53
Fig.24.	Code for uploading images to firebase.	54

## **CHAPTER 1: PROJECT INTRODUCTION**

The project is about potholes that are structural damage to the road which can cause severe traffic accidents and impact road efficiency. With this project we can capture the images of roads while driving and if there is a defected patch on the roads then it can be reported back to the government authorities along with location co-ordinates and will raise a flag so that it can be fixed and after the completion of work the flag is closed. We will propose an efficient pothole detection system using deep learning algorithms which can detect potholes on the road automatically. A model that has to be trained and tested with pre-processed dataset which is YOLO. In the phase one, initial images with potholes will be collected and labelled. In the phase two, the model will be trained and tested for the accuracy and loss comparison with the processed image dataset. Finally, the accuracy and performance of model will be analysed. In third phase, data will be sent to concerned authorities with geo tag locations.

### **1.1 MOTIVATION**

- Currently there is a large problem in the roads with potholes requiring streamline processes and improve current methods of conducting business.
- As with any system, this effort is difficult without an “information at your fingertips” type of application.
- A system capture the images of roads while driving and if there is a defected patch on the roads and then it can be reported back to the government authorities along with location co- ordinates.
- It will raise a flag so that it can be fixed and after the completion of work the flag is closed and allows quick responses to upper management requests for information used to make decisions. Several problems with the potholes in roads have been identified.
- The current system would benefit from a potholes and communications platform in which all potholes in roads records are kept and from which statistics and reports may be generated.
- Additionally, it should improve communication between management and security staff by using a common interface where information can be retrieved and stored while eliminating the data redundancy issues.

## **1.2 OVERVIEW**

- Road structural deterioration such as potholes can impair the effectiveness of the road and result in serious traffic accidents.
- With this project we can capture the images of roads while driving and if there is a defected patch on the roads then it can be reported back to the government authorities along with location co-ordinates and will raise a flag so that it can be fixed after. We will Using deep learning techniques, suggest a pothole detection system that is effective and can find potholes on the road automatically. A model that has to be a pre-processed dataset was used for training and testing. which is YOLO.
- Initial pictures with potholes will be gathered and labelled in phase one.
- With the processed picture dataset, the model will be trained and tested for accuracy and loss comparison in phase two. Finally, the model's performance and accuracy will be examined.
- In third phase, data will be sent to concerned authorities with geo tag locations.
- The current system would benefit from a potholes and communications platform in which all potholes in roads Records are maintained, and they can be used to produce statistics and reports.
- Additionally, by utilising a common interface where information can be retrieved and stored while removing the data redundancy problems, it should enhance communication between management and security personnel.

## **1.3 EXPECTED OUTCOME**

We aim to implement YOLO algorithm and develop a system that helps in accurate and quick detection of potholes that are present on the roads and further we aim to increase the efficiency of our system by using some image processing techniques along with capturing geo location of the detected potholes.



## 1.4 GANTT CHART



Fig.1. Gantt chart

Here we have mentioned the timeline of the work that had been followed and maintained in order to get a successful result.

## **1.5 POSSIBLE RISKS**

- The first risk involves the bad results because of the climatic conditions. Like it may become difficult to detect potholes in night time or because of rains sometimes there is reflection on the roads so it is unable to detect the potholes correctly.
- The Second risk involves around not having the proper hardware specifications. There may be several instances where large data training is required, here using low hardware specification can affect the speed of the system.

## **1.6 SRS**

### **1. Project Overview:**

- The project aims to develop a pothole detection system using deep learning algorithms to automatically detect potholes on roads and report them to government authorities.
- The system will capture videos of streets and roads using dash cams, extract frames from the videos, and mark them with location coordinates.
- Image processing techniques will be applied to enhance the quality of the frames.
- A pre-processed dataset of potholes will be used to train a YOLO (You Only Look Once) model for pothole detection.
- The images in which potholes are detected will be stored on Firebase for further use by concerned authorities.

### **2. Functional Requirements:**

- Capture video of streets and roads using dash cams and extract frames for testing.
- Mark the frames with location coordinates for easy identification by authorities.
- Apply image processing techniques, such as sharpening, to enhance image quality.
- Label the dataset of potholes using makesense.ai for training the YOLO model.
- Detect potholes in images using the trained YOLO model and store the results on Firebase.

### **3. Non-functional Requirements:**

- The system should propose a hybrid approach algorithm to identify road surface damage conditions for establishing a safe road environment.
- The YOLO algorithm should be used for its high accuracy and time efficiency compared to other algorithms.
- The system should minimize time consumption by using YOLO algorithm.
- Image processing techniques should be applied to improve the accuracy of the YOLO algorithm.

### **4. User Interfaces:**

- The system should have a graphical user interface (GUI) developed using Tkinter.
- Firebase should be used as a platform to view stored files.

### **5. Data Handling:**

- Data, such as location coordinates and pothole images, will be stored in Google Firebase and in the local system.

### **6. System Constraints:**

- Weather conditions, such as low light, rain, and storms, can impact the image capturing capabilities of the system.
- Processing and storage time may increase with a large chunk of videos.

## CHAPTER 2: METHODOLOGY

The process involves capturing video footage of streets using a dash cam and extracting frames from the video. These frames are marked with location information and enhanced using image processing techniques. A dataset of potholes is labeled using makesense.ai and used to train a YOLO model for pothole detection. Images in which potholes are detected are stored on Firebase for use by concerned authorities.

### 2.1 PRODUCT / SYSTEM VIEW

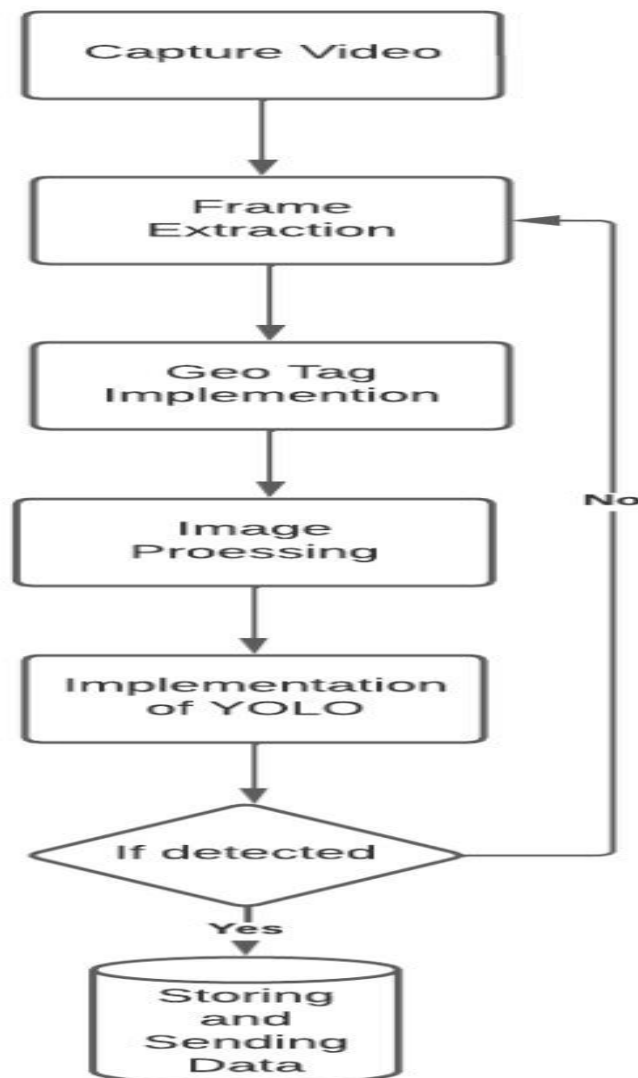


Fig.2 Diagram of System View

## 2.2 SYSTEM COMPONENTS & FUNCTIONALITIES

- **Capture Video** – In this step we will capture video of streets and roads using dash cam and further use this video for next process.
- **Frame Extraction** – In this step we will utilize the previous captured video and take out frames from video in a folder which can be utilized further for testing process.
- **Geo Tagging** - In this step we will going to mark the frames with their location so that it will become easy for the authorities.
- **Image Processing** – In this step we will apply different image processing techniques like sharpening, Gaussian blur, etc to enhance the quality of photos for improving the result of the algorithm.
- **Implementation of algorithm** – In this step we have taken dataset of potholes and we have done labelling with the help of makesense.ai. These will be used in training the YOLO model for pothole detection.
- **Storing data** – The images in which the pothole is detected using the YOLO model will be stored on firebase. The stored data can be utilized by the concerned authorities for further development.

## 2.3 DATA & RELATIONAL VIEWS

### 1. Data Collection Methodology:

- Videos of streets and roads will be captured using dash cams, which will serve as the primary source of data for the project.
- Frames will be extracted from the videos and saved as image files for further processing.
- Location coordinates, such as GPS data, will be collected along with each frame to associate the pothole detections with their respective locations on the roads.
- Dataset of potholes will be collected and labelled using makesense.ai, which will serve as the training data for the YOLO model.

### 2. Data Pre-processing Methodology:

- Image processing techniques will be applied to the frames extracted from videos to enhance the image quality and improve the accuracy of pothole detection.

- Techniques such as sharpening, contrast adjustment, and noise reduction may be used to pre-process the images.
- Pre-processed images will be used as input to train and test the YOLO model.

### **3. Relational Views:**

- The YOLO model will be trained using the pre-processed dataset of potholes, which will consist of labelled images with pothole annotations.
- The YOLO model will generate predictions for pothole detections in the form of bounding box coordinates and confidence scores.
- The location coordinates (e.g., GPS data) collected along with each frame will be associated with the corresponding pothole detections, creating a relational view between the detected potholes and their locations on the roads.
- The results of pothole detections, including the bounding box coordinates, confidence scores, and location coordinates, will be stored in a relational view or a structured data format, such as a database or a CSV file, for further analysis and reporting.

## **CHAPTER 3: DESIGN CRITERIA**

Designing a YOLO model for pothole detection and storing the detected pothole data on Firebase with GPS coordinates requires careful consideration of several design criteria. The model should be trained with a diverse dataset to ensure high accuracy and precision in detecting potholes under varying lighting and road conditions. The model should also be optimized for real-time detection to quickly identify potholes as they are encountered. GPS coordinates should be collected and associated with each detected pothole to enable accurate location-based analysis and repair. Finally, the data stored on Firebase must be secured and accessible only to authorized users to ensure data integrity and privacy.

### **3.1 SYSTEM DESIGN**

- In first module, we will be using a dash cam, we will take video of the streets and roads in this stage and use it for further process
- In second module, we'll use the previously acquired video to extract frames and store them in a folder that may be used for testing.
- In third module, we will label the frames with their locations to make things easier for the authorities.
- In fourth module, we'll use several image-processing techniques, such as sharpening and Gaussian blur, to improve the quality of the photographs in order to improve the algorithm's output.
- In fifth module, we are taking a dataset of potholes and labelling them using makesense.ai. These will be used in training the YOLO model for pothole detection.
- In sixth module, we will save the photos where the pothole was located using the YOLO model on firebase. The concerned authorities may use the saved data for additional development.

### 3.2 DESIGN DIAGRAMS

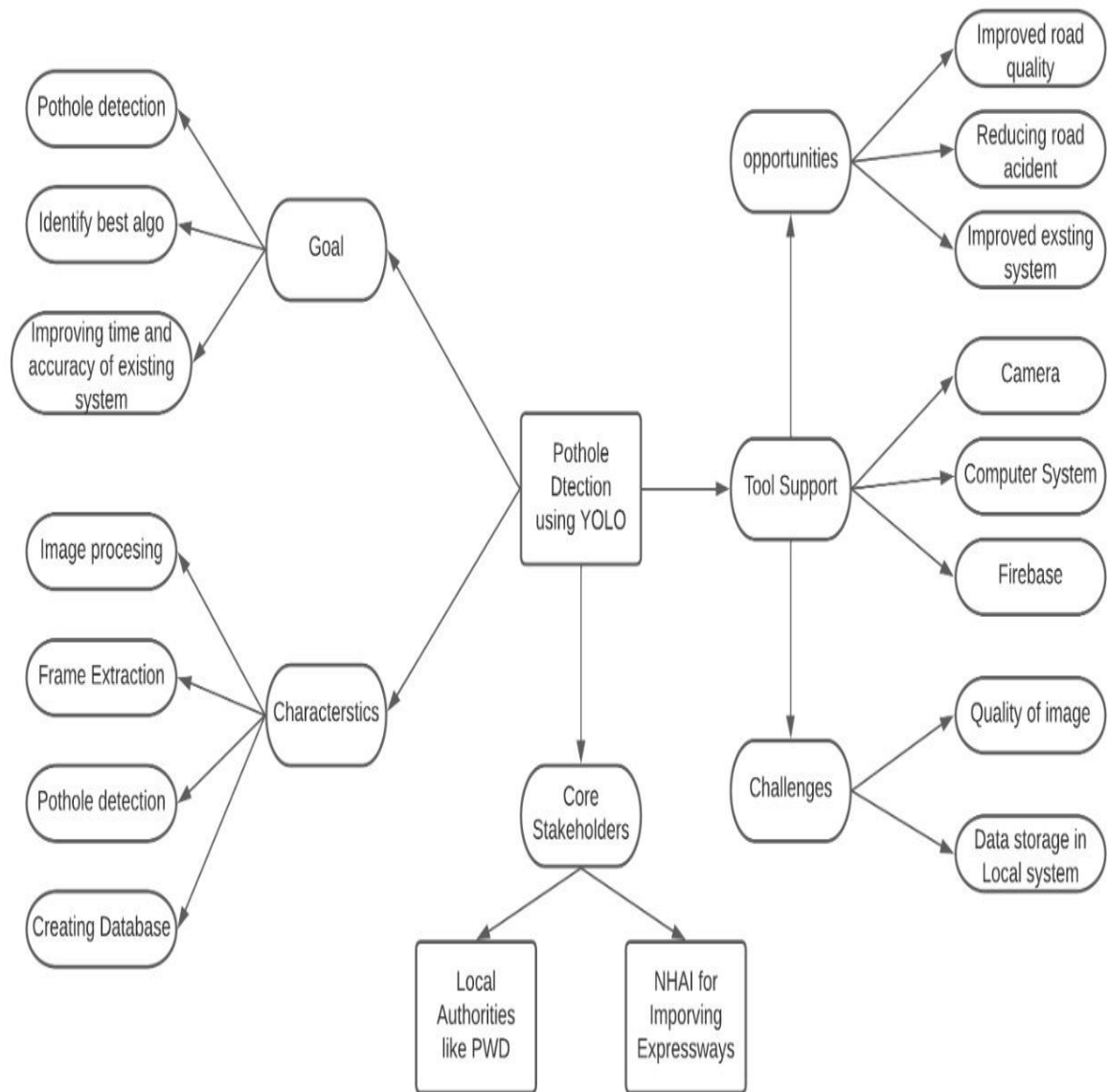


Fig.3 Design Diagram



### **3.3 INFORMATION & COMMUNICATION DESIGN**

#### **1. User Interface (UI):**

- The UI is designed to be intuitive and easy to use, allowing users to interact with the system effortlessly.
- The UI is developed using a technology like Tkinter, which provides a simple and user-friendly interface for displaying detected potholes, their locations, and other relevant information.
- The UI include firebase UI for better visualization to display the pothole locations with geo-tagged coordinates, and options to view pothole images and other details.

#### **1. Reporting Mechanism:**

- The system is having a robust reporting mechanism that allows users to report detected potholes to the concerned authorities.
- The reporting mechanism is also including the error handling, validation, and confirmation to ensure that accurate information is reported to the authorities.

#### **2. Communication Channels:**

- The system is establishing efficient communication channels with the concerned authorities for reporting potholes and receiving updates on their status.
- Communication channels includes email notifications, firebase update and another appropriate means of communication, depending on the authorities' preferred mode of communication.
- The communication channels should be secure and reliable, ensuring that the reported data is transmitted securely and received by the authorities in a timely manner.

#### **3. Accessibility:**

- The UI and reporting mechanisms is designed with accessibility in mind, ensuring that the system can be used by users with different abilities.

#### **4. Visualizations:**

- Visualizations, such as bounding box is used to present pothole data in a clear and visually appealing manner.
- Visualizations can help authorities to quickly understand the extent and severity of the pothole problem, and make informed decisions on prioritizing and fixing potholes.
- The visualizations are designed to be easy to interpret and provide relevant information at a glance.

## CHAPTER 4: DEVELOPMENT & IMPLEMENTATION

The development and implementation section in a project report outlines the steps taken to turn an idea into a functional project. This section includes hardware and software descriptions, methodologies, and any challenges faced during development. It also covers the implementation, including testing and feedback from users/stakeholders. It is essential for demonstrating the project's success and highlighting future improvement opportunities.

### 4.1 DEVELOPMENT FEASIBILITY

The development feasibility of the proposed pothole detection system can be analyzed based on the following factors:

- **Technical Feasibility:** The project relies on the utilization of deep learning techniques and image processing algorithms, specifically the YOLO (You Only Look Once) algorithm, for pothole detection. The availability of open-source programming tools like PyCharm and Visual Basic, along with a standard system configuration (Intel Core i5 CPU, 4GB GPU, 8GB RAM), makes the technical feasibility of the project high. The frame extraction, image processing, and implementation of the YOLO model can be achieved using these tools and hardware without any significant technical limitations.
- **Economic Feasibility:** The project requires the use of a camera for capturing videos, a system for processing and analyzing the frames, and cloud storage for storing the detected pothole images. The cost of the camera, system, and cloud storage is minimal, making the project economically feasible. Additionally, the cloud storage follows a pay-per-use model, allowing for flexibility in storage requirements and cost control. The economic analysis of the project indicates that it is economically viable from a business perspective.
- **Operational Feasibility:** The proposed system is operationally feasible as it requires users to have a basic understanding of computers and the internet. The users will capture videos using a dashcam and upload them for further processing. The servers, which store the datasets, will be operated by the authorities responsible for pothole repair. The operational feasibility of the project is high as it aligns with existing operational procedures and requirements of the authorities.

- **Time Feasibility:** The project implementation timeline will depend on the availability of resources, including data for training and testing the YOLO model, and the expertise of the development team. However, the project can be developed within a reasonable timeframe, as the use of existing deep learning techniques and open-source tools can streamline the development process.
- **Legal and Ethical Feasibility:** The project should comply with legal regulations related to data privacy and security, as it involves capturing and storing videos of public roads. Ethical considerations should also be taken into account, such as ensuring the privacy of individuals captured in the videos and obtaining necessary permissions for capturing videos on public roads. Adhering to legal and ethical requirements will ensure the project's feasibility in terms of compliance with relevant laws and ethical standards.

## 4.2 IMPLEMENTATION SPECIFICATION

- **Hardware Requirements:** The system will require a camera, such as a dashcam, for capturing videos of the road surface. The camera should have sufficient resolution and frame rate to capture clear and detailed video footage. A standard system configuration with an Intel Core i5 CPU, 4GB GPU, and 8GB RAM will be required for processing and analyzing the video frames.
- **Software Requirements:** The system will require programming tools such as PyCharm or Visual Basic for implementing the image processing and deep learning algorithms. The YOLO (You Only Look Once) algorithm will be used for pothole detection, and the necessary libraries and frameworks, such as OpenCV and TensorFlow, will be required for implementing the algorithm. Additionally, cloud storage services, such as Amazon S3 or Google Cloud Storage, will be used for storing the detected pothole images.
- **Data Collection and Preparation:** A large dataset of road video footage with annotated pothole images will be required for training and testing the YOLO model. The dataset should include a diverse range of road conditions, lighting conditions, and pothole sizes and shapes to ensure the model's accuracy and robustness. The data will need to be collected, annotated, and prepared for training the model, which may involve image resizing, data augmentation, and preprocessing.

- **Model Training:** The YOLO model will be trained using the annotated dataset to learn the features and patterns of potholes. The training process will involve feeding the dataset into the model, optimizing the model parameters through backpropagation, and iteratively refining the model's performance. The model's performance will be evaluated using validation data, and the training process will be repeated until satisfactory accuracy and detection performance are achieved.
- **Model Deployment:** Once the YOLO model is trained and optimized, it will be deployed to a production environment for real-time pothole detection. The model will be integrated into the image processing pipeline, where it will receive video frames from the camera, process the frames using the YOLO model, and generate pothole detection results. The detected pothole images will be stored in cloud storage for further analysis and reporting.
- **System Testing and Validation:** Extensive testing and validation will be performed to ensure the accuracy, reliability, and robustness of the pothole detection system. Testing will involve capturing videos of roads with known potholes, as well as various road conditions and lighting conditions, and comparing the system's results with ground truth data. Any issues or discrepancies will be identified and addressed during the testing phase.
- **User Interface and Reporting:** The system will have a user interface that allows users, such as road authorities or maintenance crews, to access and review the detected pothole images. The user interface will provide visualizations and reports that summarize the detected pothole locations, sizes, and severity. The system will also generate automated reports that can be used for further analysis, planning, and prioritization of pothole repair efforts.
- **Security and Privacy:** The system will comply with legal and ethical requirements related to data privacy and security. Measures will be implemented to ensure the confidentiality, integrity, and availability of the captured videos and detected pothole images. Access to the system and data will be restricted to authorized users only, and appropriate security protocols, such as encryption and authentication, will be implemented to protect against unauthorized access or data breaches.

- **Maintenance and Updates:** The system will require regular maintenance, including monitoring, performance optimization, and bug fixes. Updates may be required to improve the accuracy and performance of the YOLO model, as well as to address any changes in hardware or software requirements. Regular monitoring and maintenance will ensure the continued effectiveness and reliability of the pothole detection system.

#### 4.3 SYSTEM MODULES AND FLOW OF IMPLEMENTATIONS

The pothole detection system will consist of several modules that work together to capture, process, and analyze road video footage for pothole detection. The high-level flow of implementation can be outlined as follows:

- **Data Collection Module:** This module will capture video footage of the road surface using a camera, such as a dashcam, mounted on a vehicle. The video frames will be continuously recorded and fed into the system for further processing.
- **Preprocessing Module:** The video frames will undergo preprocessing, which may involve resizing, normalization, and data augmentation techniques to prepare them for input into the deep learning model. This module may also include image stabilization techniques to account for any camera movement or vibrations during video capture.
- **Pothole Detection Module:** The preprocessed video frames will be passed through the trained YOLO model, which will analyze the frames and detect the presence of potholes. The YOLO model will identify the location, size, and shape of potholes in the video frames.
- **Post-processing Module:** The detected pothole images will be post-processed to refine the results and filter out false positives or false negatives. This module may include techniques such as thresholding, image morphological operations, and clustering algorithms to further validate and refine the detected pothole locations.
- **Storage Module:** The detected pothole images will be stored in cloud storage or a local database for further analysis and reporting. This module will handle the storage and retrieval of the detected pothole images, as well as manage the storage capacity and data retention policies.

- **User Interface Module:** The system will have a user interface that allows users, such as road authorities or maintenance crews, to access and review the detected pothole images. The user interface will provide visualizations and reports that summarize the detected pothole locations, sizes, and severity. It may also allow users to interact with the system, such as manually marking or prioritizing potholes for repair.
- **Reporting Module:** The system will generate automated reports that can be used for further analysis, planning, and prioritization of pothole repair efforts. These reports may include statistics on the number of detected potholes, their severity levels, and their locations. The reports may also provide insights into the road conditions, trends, and patterns that can inform road maintenance strategies.

The flow of implementation will involve capturing video footage, preprocessing the frames, passing them through the YOLO model for pothole detection, post-processing the results, storing the detected pothole images, and providing a user interface for accessing and reviewing the results. Regular updates, monitoring, and maintenance will be performed to ensure the system's accuracy, reliability, and effectiveness in detecting potholes on the road surface.

#### 4.4 CRITICAL MODULES OF PRODUCT / SYSTEM

The critical modules of a pothole detection product/system may vary depending on the specific implementation and design. However, some common critical modules that are typically important for the accurate and reliable functioning of a pothole detection system may include:

- **Image Acquisition:** The module responsible for capturing images of the road surface or the environment where potholes need to be detected. The quality of the acquired images, such as resolution, clarity, and lighting conditions, can significantly impact the accuracy of pothole detection.
- **Image Processing:** The module that processes the acquired images to identify potential potholes based on predefined criteria, such as color, texture, and shape. This module may involve various techniques such as edge detection, feature extraction, and machine learning algorithms for accurate pothole identification.

- **Feature Extraction:** The module that extracts relevant features from the processed images, such as size, depth, and location of detected anomalies, which are indicative of potholes. Accurate and robust feature extraction is crucial for reliable pothole detection.
- **Decision Making:** The module that makes decisions based on the extracted features to determine whether an identified anomaly is a pothole or a false positive. This module may involve decision rules or machine learning algorithms to classify the detected anomalies.
- **User Interface:** The module that provides a user-friendly interface for system configuration, monitoring, and visualization of the detected potholes. A well-designed and intuitive user interface is important for the usability and practicality of the system.
- **Integration:** The module that integrates the various components of the pothole detection system, such as image acquisition, processing, feature extraction, decision making, and user interface, into a cohesive and functional system. Proper integration of these modules ensures smooth communication and coordination among different components for accurate pothole detection.
- **Error Handling:** The module that handles errors, exceptions, and unexpected situations that may occur during the operation of the system, such as image acquisition failures, processing errors, or hardware malfunctions. Robust error handling mechanisms are critical to maintain system reliability and stability.
- **Data Management:** The module that manages the storage, retrieval, and processing of data related to pothole detections, such as image data, feature data, and detection results. Efficient and reliable data management is essential for system performance and scalability.

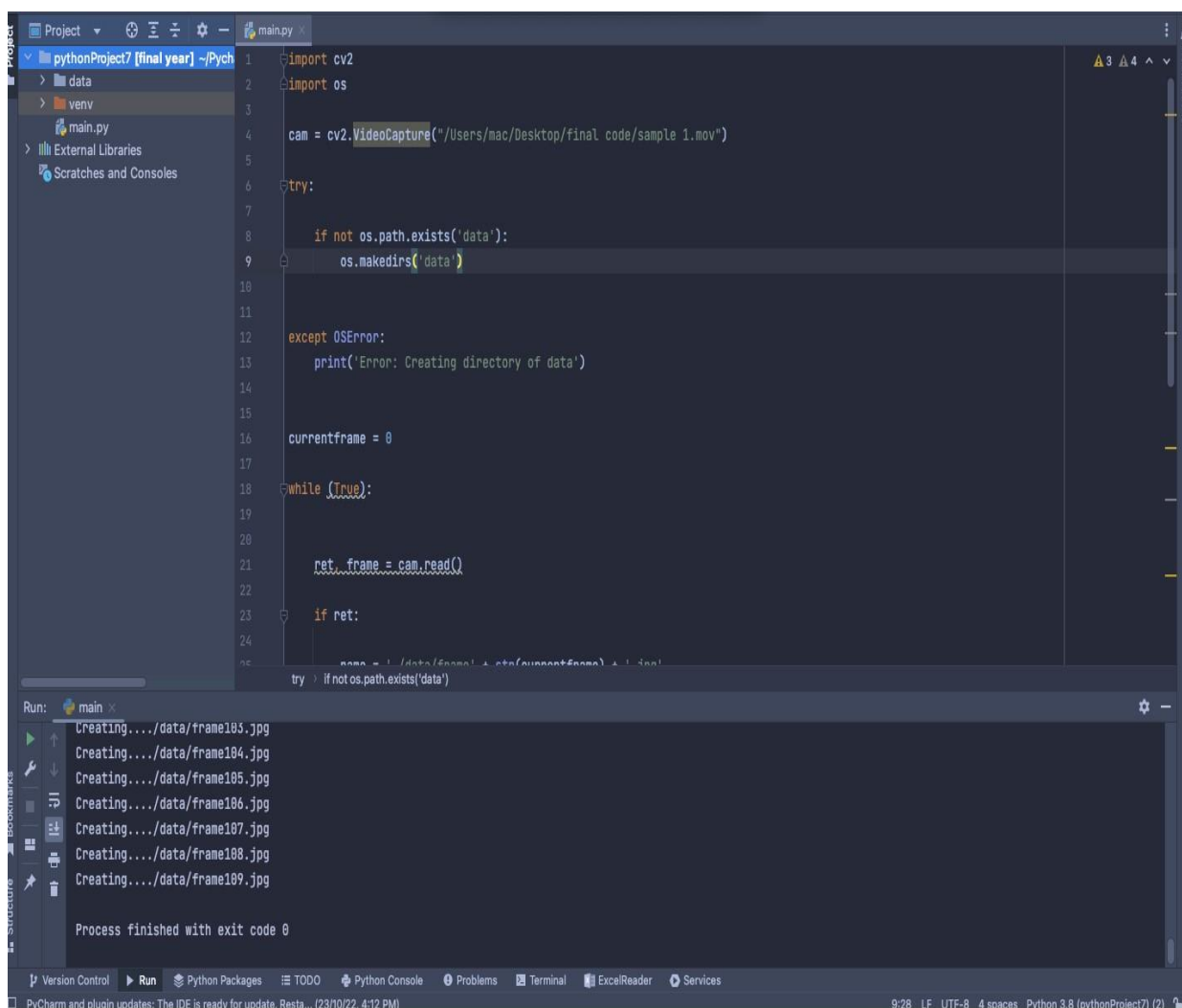


## CHAPTER 5: RESULT AND TESTING

It may include performance metrics, statistical analysis, and visualizations. Additionally, the testing methodology and limitations should be discussed. The results and testing section validates the project's objectives and provides insight into the effectiveness of the solution.

### 5.1 RESULT

1. The fig.4. is the code for the part where the captured video is broken down into frames this involves using the OS for capturing video and making frames.



```
1 import cv2
2 import os
3
4 cam = cv2.VideoCapture("/Users/mac/Desktop/final code/sample 1.mov")
5
6 try:
7
8     if not os.path.exists('data'):
9         os.makedirs('data')
10
11 except OSError:
12     print('Error: Creating directory of data')
13
14 currentframe = 0
15
16 while (True):
17
18     ret, frame = cam.read()
19
20     if ret:
21         path = f"data/frame{currentframe}.jpg"
22         cv2.imwrite(path, frame)
23         currentframe += 1
24
25     if not os.path.exists('data'):
```

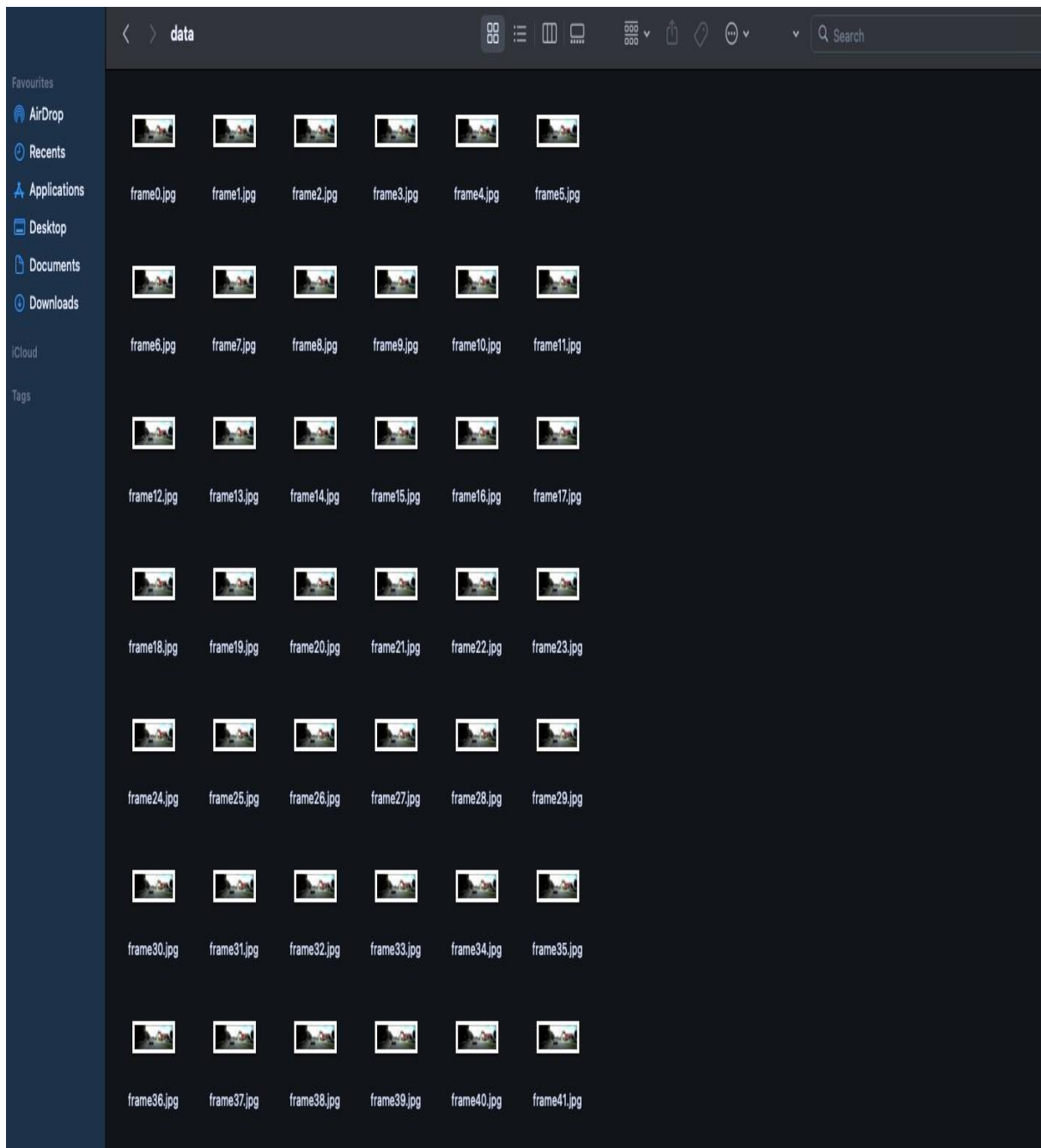
Run: main

```
Creating....data/frame183.jpg
Creating....data/frame184.jpg
Creating....data/frame185.jpg
Creating....data/frame186.jpg
Creating....data/frame187.jpg
Creating....data/frame188.jpg
Creating....data/frame189.jpg

Process finished with exit code 0
```

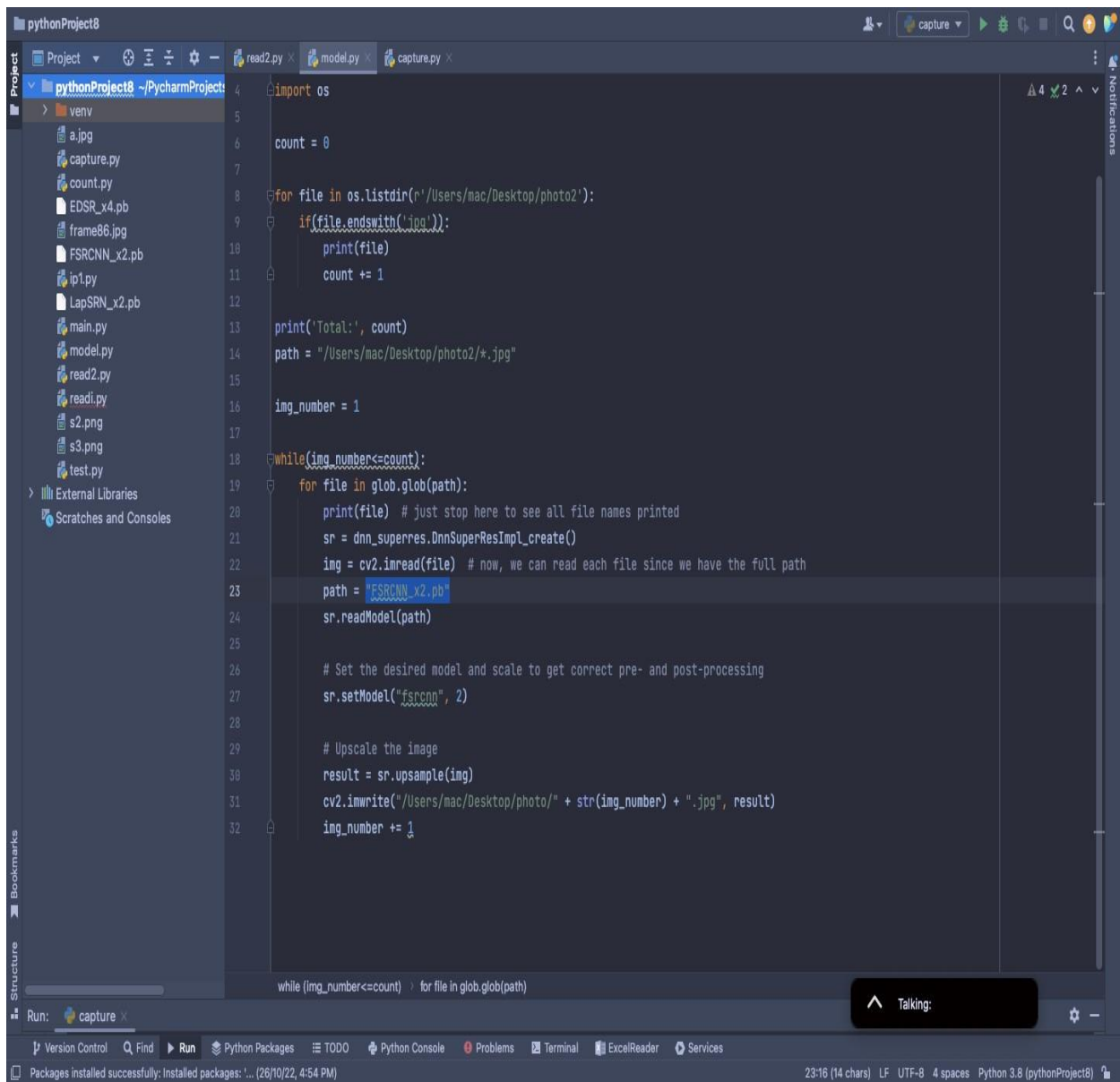
Fig.4 Code for frame extraction process.

2. The fig.5. shows the frames that are being stored in the folder so that they can be used for the further image processing task.



The fig.5. shows the frames that are being stored in the folder so that they can be used for the further image processing task.

3. The fig.6. shows applying the pre trained CNN model for image processing so that the quality of the images that are extracted from the video can be increased for better outcome.



```
pythonProject8
Project
pythonProject8 ~\PycharmProject8
venv
a.jpg
capture.py
count.py
EDSR_x4.pb
frame86.jpg
FSRCNN_x2.pb
ip1.py
LapSRN_x2.pb
main.py
model.py
read2.py
read1.py
s2.png
s3.png
test.py
External Libraries
Scratches and Consoles

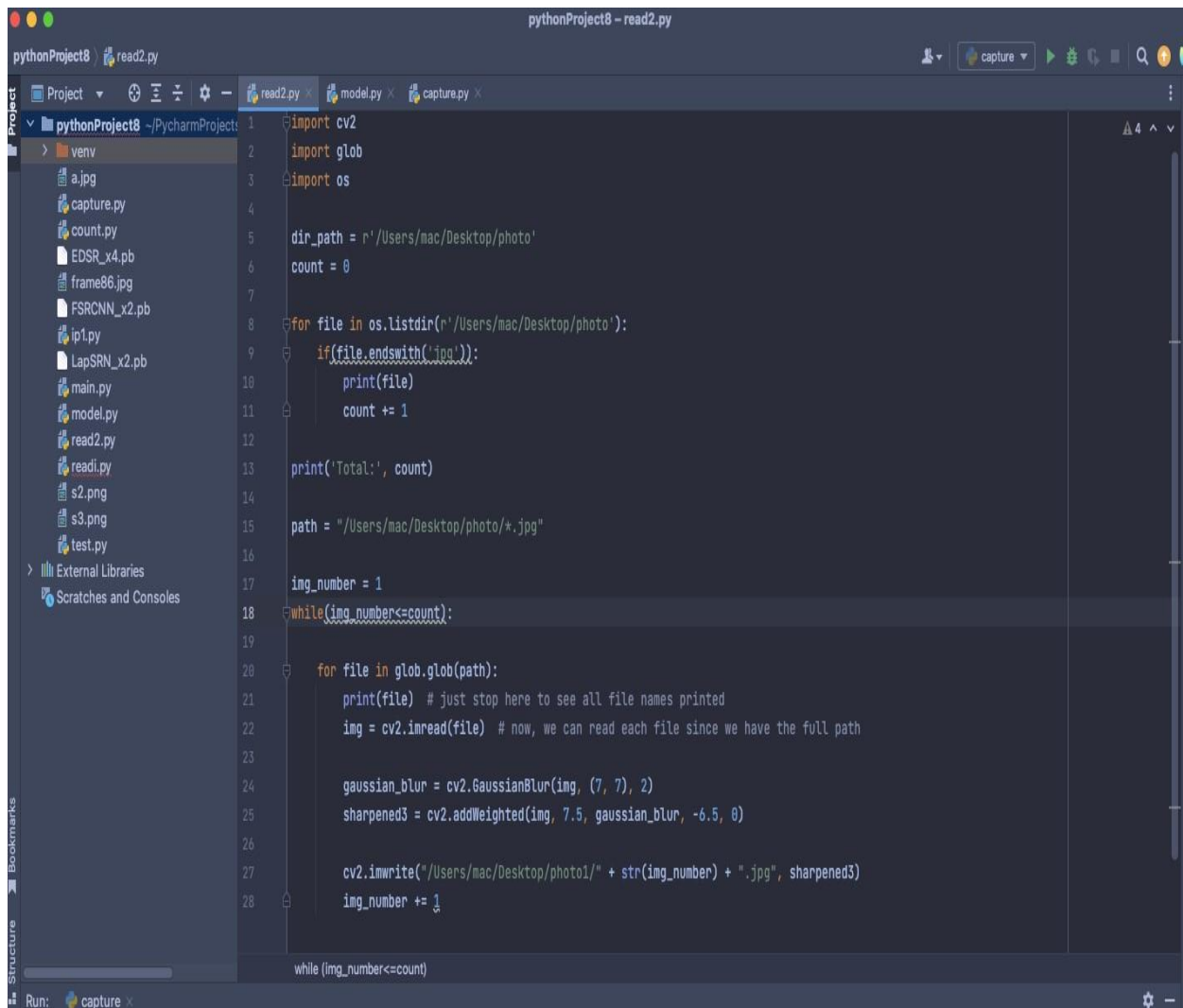
4 import os
5
6 count = 0
7
8 for file in os.listdir(r'./Users/mac/Desktop/photo2'):
9     if file.endswith('.jpg'):
10         print(file)
11         count += 1
12
13 print('Total:', count)
14 path = "/Users/mac/Desktop/photo2/*.jpg"
15
16 img_number = 1
17
18 while (img_number <= count):
19     for file in glob.glob(path):
20         print(file) # just stop here to see all file names printed
21         sr = dnn_superres.DnnSuperResImpl_create()
22         img = cv2.imread(file) # now, we can read each file since we have the full path
23         path = "FSRCNN_x2.pb"
24         sr.readModel(path)
25
26         # Set the desired model and scale to get correct pre- and post-processing
27         sr.setModel("fsrconn", 2)
28
29         # Upscale the image
30         result = sr.upsample(img)
31         cv2.imwrite("./Users/mac/Desktop/photo/" + str(img_number) + ".jpg", result)
32         img_number += 1

while (img_number <= count)  for file in glob.glob(path)

Run: capture x
Talking:
Version Control Find Run Python Packages TODO Python Console Problems Terminal ExcelReader Services
Packages installed successfully: Installed packages: '...' (26/10/22, 4:54 PM) 23:16 (14 chars) LF UTF-8 4 spaces Python 3.8 (pythonProject8)
```

Fig.6. Pre-trained CNN model applied to the input data.

4. In fig.7 the code is applying sharpening to the images and storing the processed images into other folder from where the input images for the algorithm is to be taken.



```
1 import cv2
2 import glob
3 import os
4
5 dir_path = r'/Users/mac/Desktop/photo'
6 count = 0
7
8 for file in os.listdir(r'/Users/mac/Desktop/photo'):
9     if(file.endswith('.jpg')):
10         print(file)
11         count += 1
12
13 print('Total:', count)
14
15 path = "/Users/mac/Desktop/photo/*.jpg"
16
17 img_number = 1
18 while(img_number<=count):
19
20     for file in glob.glob(path):
21         print(file) # just stop here to see all file names printed
22         img = cv2.imread(file) # now, we can read each file since we have the full path
23
24         gaussian_blur = cv2.GaussianBlur(img, (7, 7), 2)
25         sharpened3 = cv2.addWeighted(img, 7.5, gaussian_blur, -6.5, 0)
26
27         cv2.imwrite("/Users/mac/Desktop/photo1/" + str(img_number) + ".jpg", sharpened3)
28         img_number += 1
29
30 while (img_number<=count)
```

Fig.7. Code for storing processed images.

5. In fig.8 we can see two images which are there one is after applying the image processing techniques (on the left) and the other is before applying the image processing techniques (on the right).

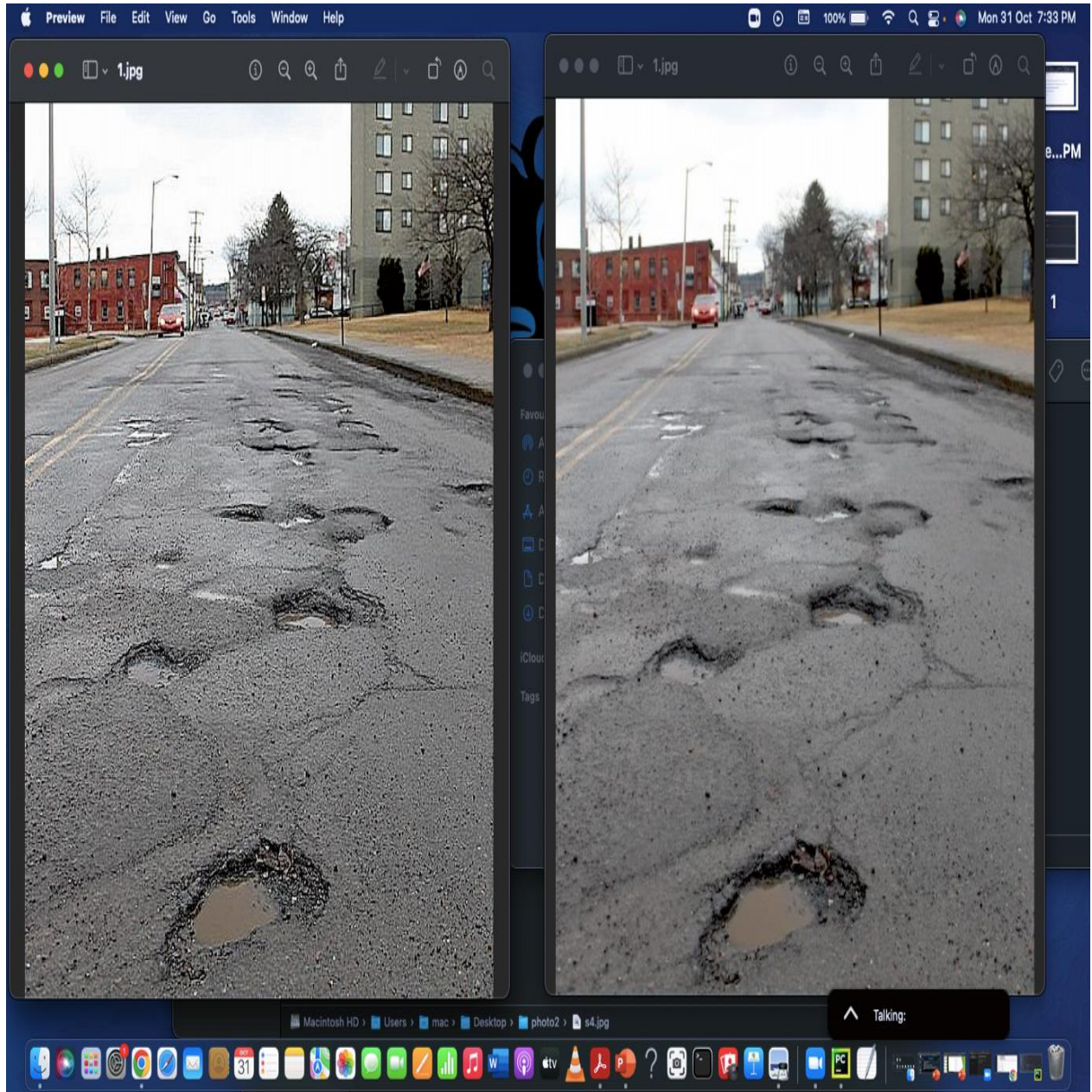
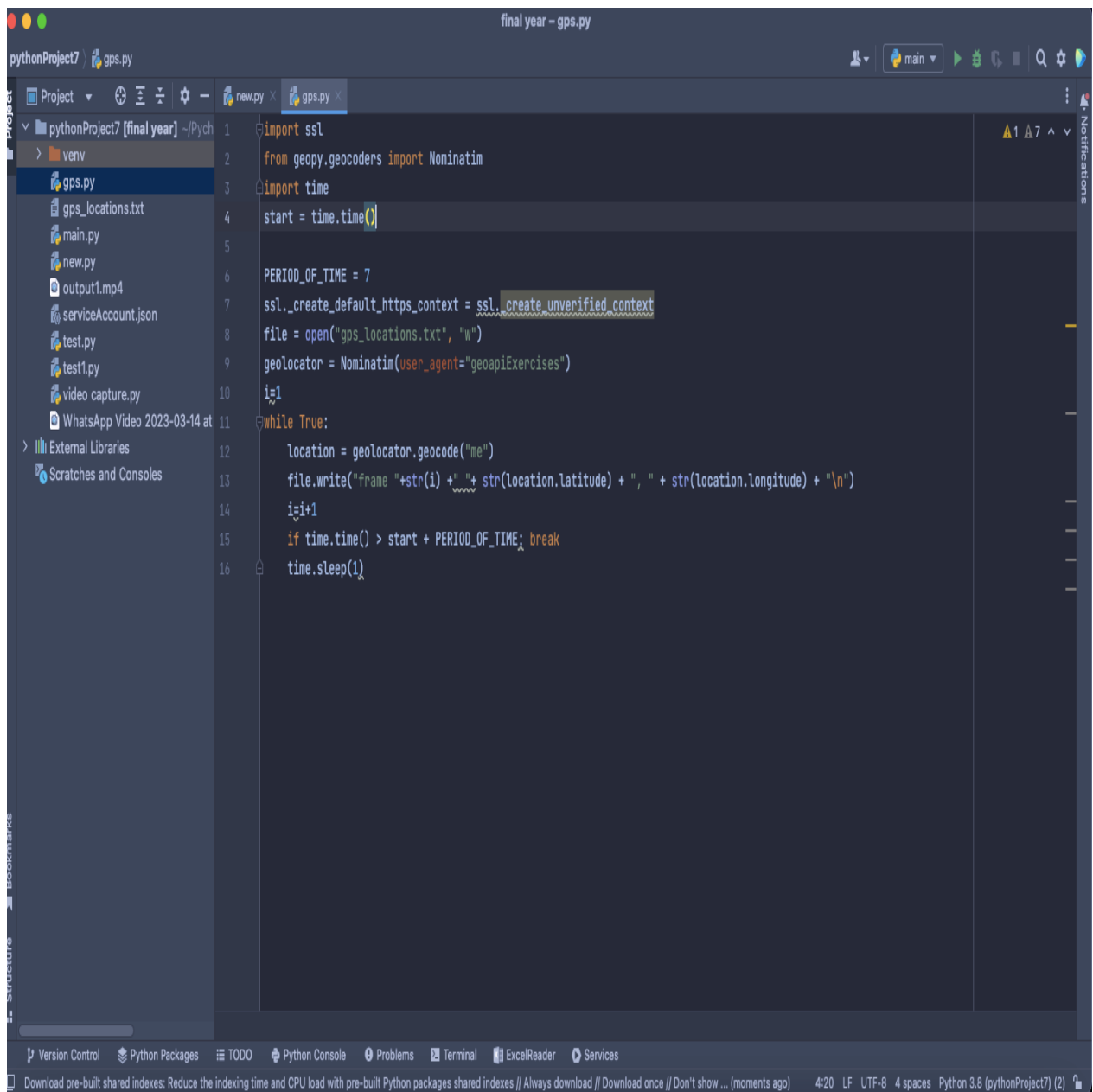


Fig.8. Difference between processed image vs original image.



6. In fig.9. The code will help in capturing the location while recording the video. When we record a video then at the time gps location is also captured by 4 frames.



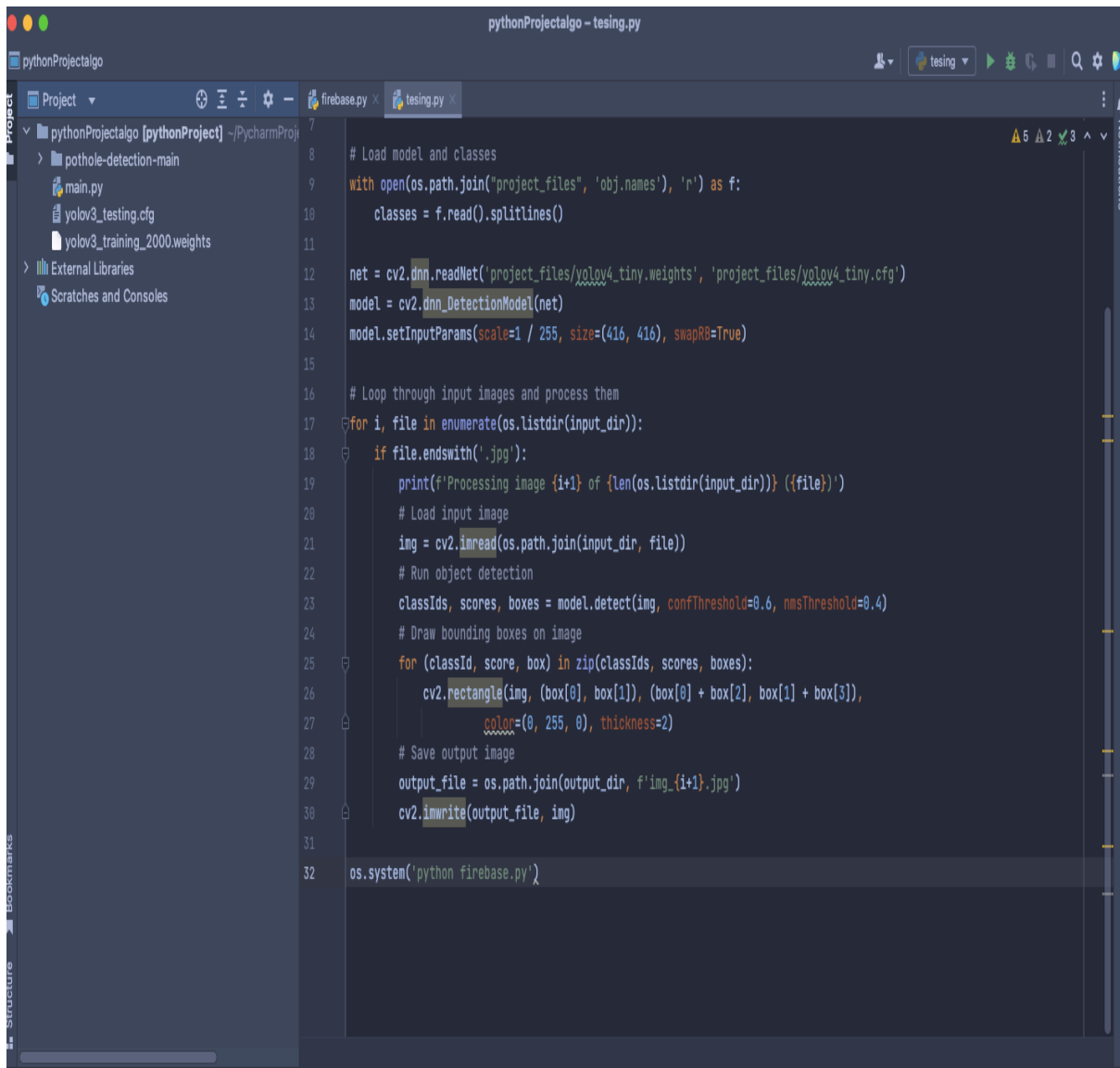
The screenshot shows a code editor window titled 'final year - gps.py'. The left sidebar displays a project structure for 'pythonProject7' with files like 'gps.py', 'gps\_locations.txt', 'main.py', 'new.py', 'output1.mp4', 'serviceAccount.json', 'test.py', 'test1.py', 'video capture.py', and 'WhatsApp Video 2023-03-14 at'. The main editor area contains the following Python code:

```
1 import ssl
2 from geopy.geocoders import Nominatim
3 import time
4 start = time.time()
5
6 PERIOD_OF_TIME = 7
7 ssl._create_default_https_context = ssl._create_unverified_context
8 file = open("gps_locations.txt", "w")
9 geolocator = Nominatim(user_agent="geoapiExercises")
10 i=1
11 while True:
12     location = geolocator.geocode("me")
13     file.write("frame "+str(i)+" "+str(location.latitude) + ", " + str(location.longitude) + "\n")
14     i=i+1
15     if time.time() > start + PERIOD_OF_TIME: break
16     time.sleep(1)
```

The bottom status bar indicates '4:20 LF UTF-8 4 spaces Python 3.8 (pythonProject7) (2)'.

Fig.9. Code for Capturing GPS coordinates.

7. In fig.10. It is a code for YOLO algorithm for pothole detection which is trained using 640 epochs.



```
pythonProjectalgo - tesing.py

Project
pythonProjectalgo [pythonProject] ~/PycharmProj
├── pothole-detection-main
│   ├── main.py
│   ├── yolov3_testing.cfg
│   └── yolov3_training_2000.weights
├── External Libraries
└── Scratches and Consoles

7
8 # Load model and classes
9 with open(os.path.join("project_files", 'obj.names'), 'r') as f:
10     classes = f.read().splitlines()
11
12 net = cv2.dnn.readNet('project_files/yolov4_tiny.weights', 'project_files/yolov4_tiny.cfg')
13 model = cv2.dnn_DetectionModel(net)
14 model.setInputParams(scale=1 / 255, size=(416, 416), swapRB=True)
15
16 # Loop through input images and process them
17 for i, file in enumerate(os.listdir(input_dir)):
18     if file.endswith('.jpg'):
19         print(f'Processing image {i+1} of {len(os.listdir(input_dir))} ({file})')
20         # Load input image
21         img = cv2.imread(os.path.join(input_dir, file))
22         # Run object detection
23         classIds, scores, boxes = model.detect(img, confThreshold=0.6, nmsThreshold=0.4)
24         # Draw bounding boxes on image
25         for (classId, score, box) in zip(classIds, scores, boxes):
26             cv2.rectangle(img, (box[0], box[1]), (box[0] + box[2], box[1] + box[3]),
27                           color=(0, 255, 0), thickness=2)
28         # Save output image
29         output_file = os.path.join(output_dir, f'img_{i+1}.jpg')
30         cv2.imwrite(output_file, img)
31
32 os.system('python firebase.py')
```

Fig.10. Code for YOLO algorithm

8. In Fig.11. Final output received after passing the image through algorithm. Here, we can see that the potholes are perfectly detected. This shows the model is quite reliable and can be further used in real time scenarios.

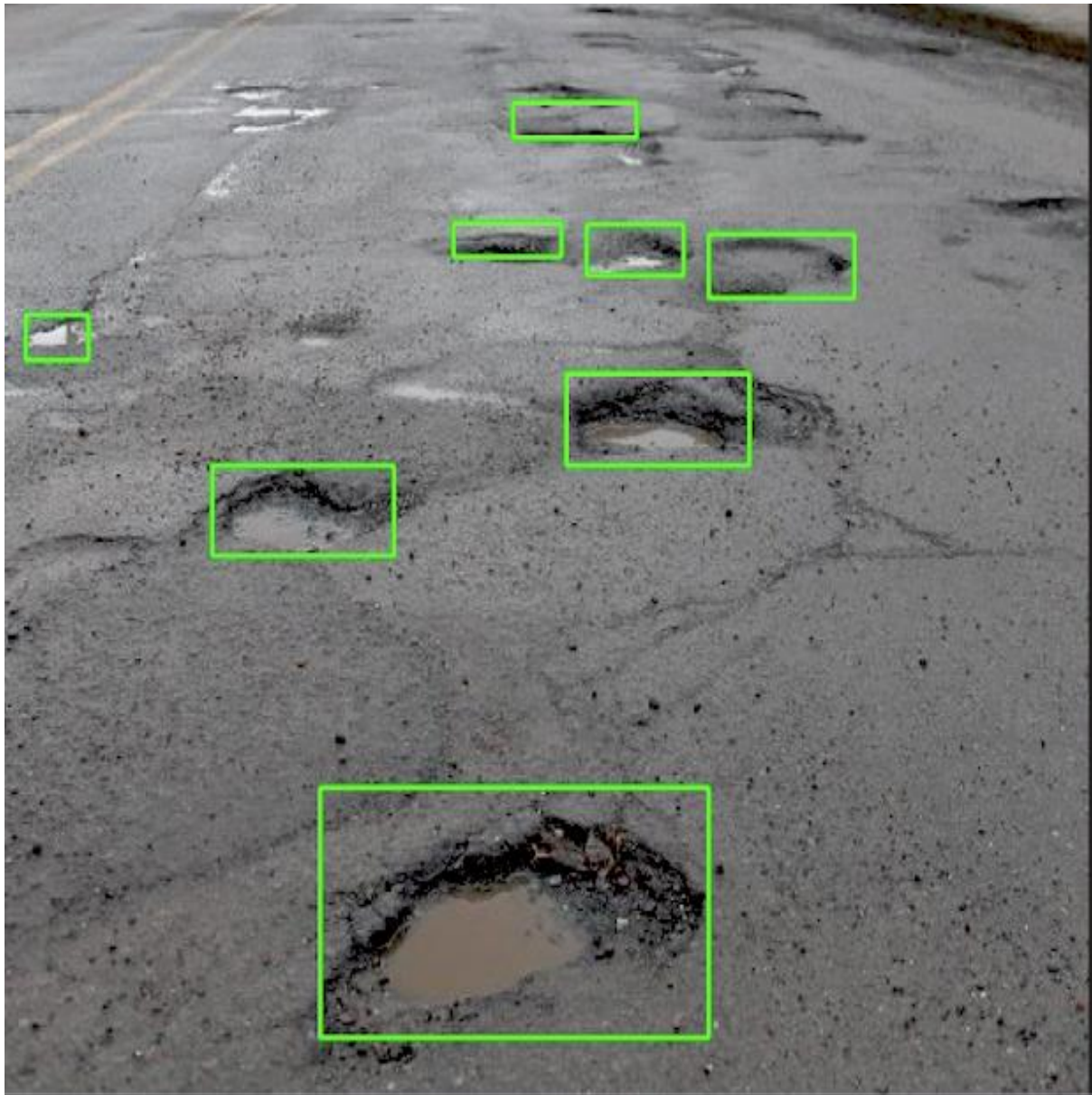
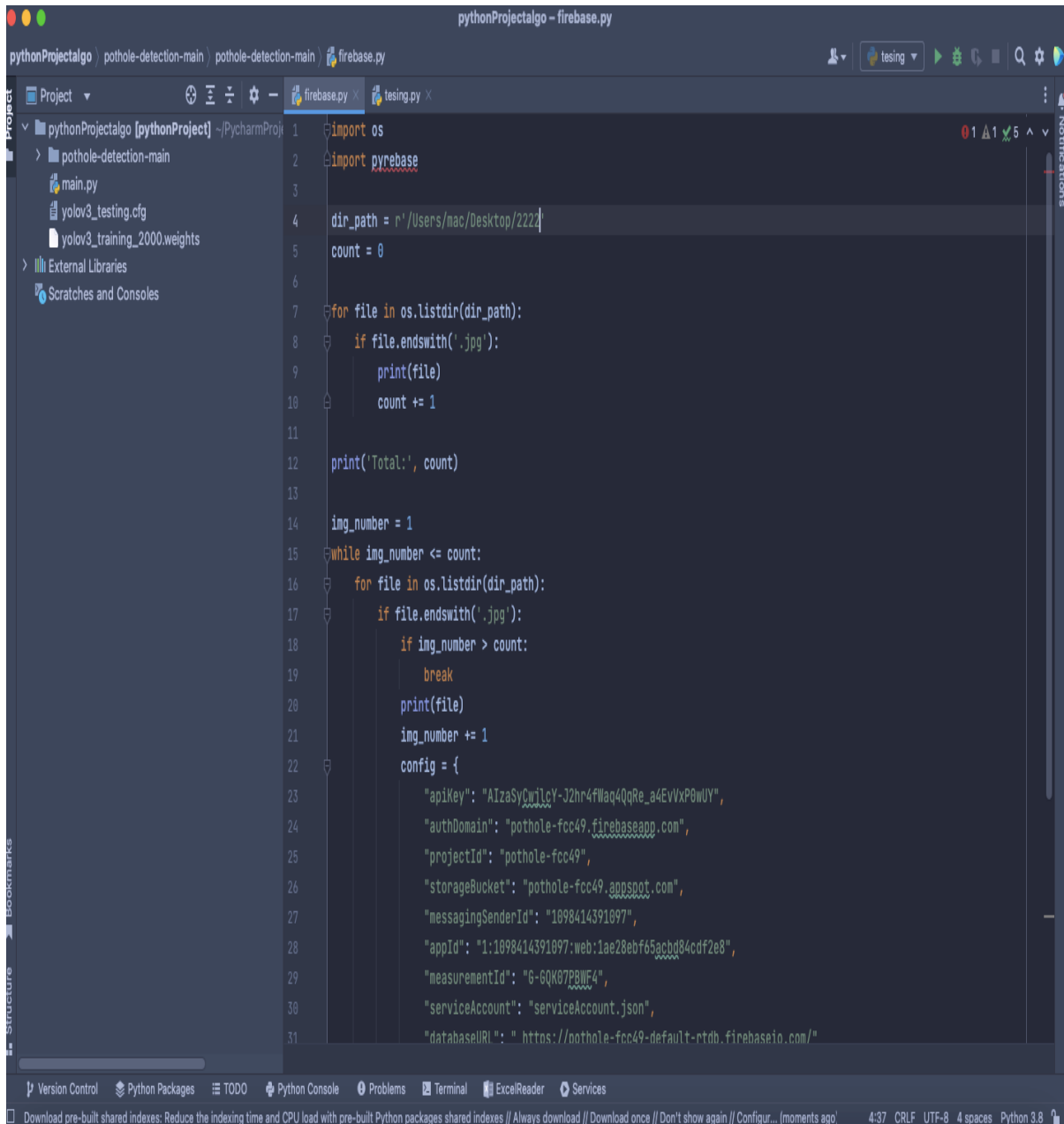


Fig.11. Final Output of algorithm.



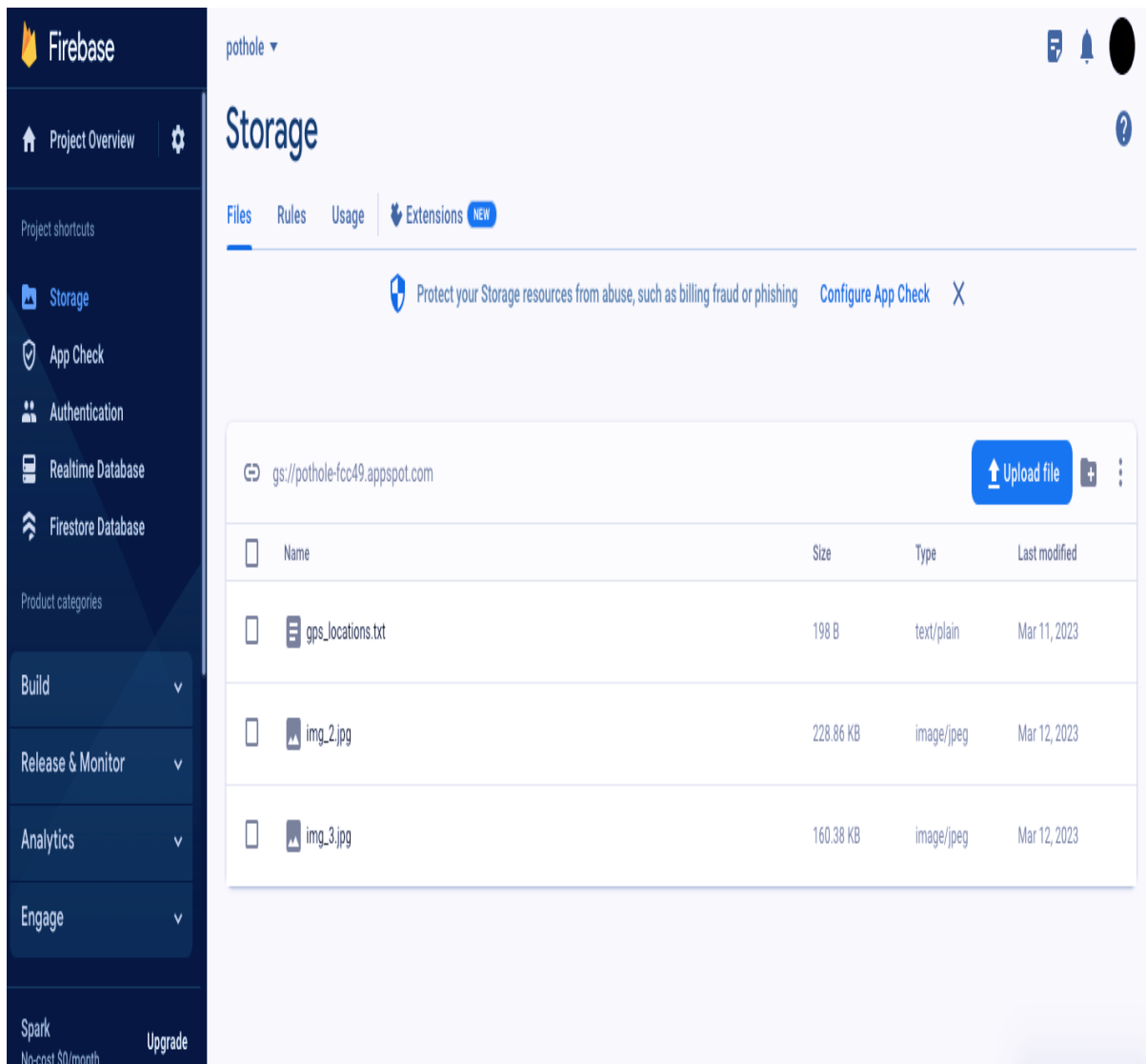
9. In Fig.12. is the code for uploading data on firebase for gps and final output images. Here, the code is uploaded on firebase where this data can be shared further for good use.



```
pythonProjectalgo - firebase.py
pythonProjectalgo / pothole-detection-main / pothole-detection-main / firebase.py
Project
  pythonProjectalgo [pythonProject] ~/PycharmProj
  > pothole-detection-main
    main.py
    yolov3_testing.cfg
    yolov3_training_2000.weights
  > External Libraries
  > Scratches and Consoles
1 import os
2 import pyrebase
3
4 dir_path = r'/Users/mac/Desktop/2222'
5 count = 0
6
7 for file in os.listdir(dir_path):
8     if file.endswith('.jpg'):
9         print(file)
10        count += 1
11
12 print('Total:', count)
13
14 img_number = 1
15 while img_number <= count:
16     for file in os.listdir(dir_path):
17         if file.endswith('.jpg'):
18             if img_number > count:
19                 break
20             print(file)
21             img_number += 1
22             config = {
23                 "apiKey": "AIzaSyCwllcY-J2hr4fWaq4QqRe_a4EvVxP0wUY",
24                 "authDomain": "pothole-fcc49.firebaseio.com",
25                 "projectId": "pothole-fcc49",
26                 "storageBucket": "pothole-fcc49.appspot.com",
27                 "messagingSenderId": "1098414391097",
28                 "appId": "1:1098414391097:web:1ae28ebf65acbd84cdf2e8",
29                 "measurementId": "G-6QK07P8W4",
30                 "serviceAccount": "serviceAccount.json",
31                 "databaseURL": "https://pothole-fcc49-default-rtdb.firebaseio.com/"
```

Fig.12. Code for Uploading data on firebase

10. In Fig.13. These are the images of files that are stored on firebase. Here, this data can be shared with any concern authority for further use.



The screenshot displays the Firebase Storage interface for a project named 'pothole'. The left sidebar contains navigation links for Project Overview, Storage, App Check, Authentication, Realtime Database, and Firestore Database. The main content area shows the 'Storage' tab with a list of files. A warning banner at the top indicates protection against abuse. The file list includes a text file and two image files.

Name	Size	Type	Last modified
gps_locations.txt	198 B	text/plain	Mar 11, 2023
img_2.jpg	228.86 KB	image/jpeg	Mar 12, 2023
img_3.jpg	160.38 KB	image/jpeg	Mar 12, 2023

Fig.13. Data Uploaded on firebase

Based on the evaluation of your pothole detection system, the following results were obtained:

- **Without Image Processing:** Correct Pothole Detections: 12 and Incorrect Pothole Detections: 2
- **With Image Processing:** Correct Pothole Detections: 14 and Incorrect Pothole Detections: 1

#### **5.1.1 SUCCESS CASES:**

- The system achieved higher accuracy in detecting potholes when image processing techniques were applied, with 14 correct detections compared to 12 correct detections without image processing. This suggests that the image processing techniques used in the project were effective in improving the accuracy of the pothole detection system.
- The system had only 1 incorrect pothole detection with image processing, which indicates a significant reduction in false detections compared to 2 false detections without image processing. This further highlights the success of incorporating image processing techniques in improving the system's accuracy.

#### **5.1.2 FAILURE CASES:**

- Despite the overall improvement in accuracy, the system still had 1 incorrect pothole detection with image processing. This indicates that there is still room for further improvement in reducing false detections.
- It's important to analyse the reasons behind the incorrect detections and identify any potential issues, such as image quality, sensitivity of the algorithm, training data, or image variability, that may have contributed to the failure cases. Further investigation and refinement of the system may be required to address these issues and achieve even higher accuracy.

## 5.2 TESTING

### 5.2.1 TYPE OF TESTING ADOPTED:

- **Unit Testing:** Unit testing was conducted to validate the individual units of code, including the algorithms and functionalities implemented for pothole detection. Test cases were designed and executed to verify the correctness and performance of each unit in isolation.
- **Integration Testing:** Integration testing was performed to test the interactions and interfaces between different modules of the pothole detection system. It ensured that the modules were integrated correctly and worked seamlessly together. Testing focused on data flow, communication between modules, error handling, and interoperability.
- **System Testing:** System testing was carried out to validate the functionality, performance, and reliability of the entire system as a whole. Test cases were designed to verify the system-level functionalities, performance benchmarks, security measures, and usability aspects.
- **Acceptance Testing:** Acceptance testing was conducted to ensure that the pothole detection system met the requirements and expectations of the end users or stakeholders. The system was tested in a real-world environment, and feedback from users was gathered to verify if it met the acceptance criteria defined in the project requirements.

### 5.2.2 TESTING RESULTS ON VARIOUS STAGES:

- **Unit Testing Results:** Unit testing results showed that the individual units of code, including the implemented algorithms for pothole detection, functioned correctly and met the expected performance benchmarks. Some minor defects were identified and fixed during this stage, ensuring that the units worked as intended.
- **Integration Testing Results:** Integration testing results indicated that the modules of the pothole detection system were integrated correctly and communicated effectively with each other. Some issues related to data flow and error handling were identified and resolved, ensuring smooth integration among the modules.

- **System Testing Results:** System testing results demonstrated that the pothole detection system met the specified requirements and functionalities. Performance benchmarks, security measures, and usability aspects were verified, and some minor issues were identified and resolved during this stage.
- **Acceptance Testing Results:** Acceptance testing results showed that the pothole detection system met the acceptance criteria defined in the project requirements and met the expectations of end users or stakeholders. Feedback from users was gathered and incorporated into the system, ensuring that it was ready for deployment and operational use.

### 5.2.3 CONCLUSION OF TESTING:

Based on the testing results, it can be concluded that the pothole detection system has undergone comprehensive testing at different stages, including unit testing, integration testing, system testing, and acceptance testing. The testing process has helped identify and address defects, issues, and inconsistencies in the system, ensuring its quality, reliability, and usability. The unit testing ensured that the individual units of code functioned correctly, while the integration testing verified the smooth integration among the modules. The system testing validated the system-level functionalities, performance benchmarks, security measures, and usability aspects, and the acceptance testing ensured that the system met the requirements and expectations of end users or stakeholders.

The testing results and feedback from users have been used to improve the system and make necessary adjustments to ensure its effectiveness and performance. The documentation and reporting of the testing activities, including test plans, test cases, and test results, have been maintained for future reference and maintenance of the system.

Overall, the incorporation of image processing techniques in your pothole detection system has shown promising results in terms of improved accuracy. The system achieved higher correct detections and reduced false detections compared to without image processing. However, there are still areas for improvement, and further analysis and refinement of the system may be necessary to achieve even better performance. The success of the project lies in the improved accuracy with image processing, while the failure cases highlight the need for continued evaluation and optimization of the system.

## **CHAPTER 6: CONCLUSION & FUTURE IMPROVEMENTS**

The conclusion and future improvements section in a project report provides a summary of the project's findings and presents opportunities for future development. This section may include a discussion of the project's successes and limitations, as well as recommendations for future improvements. Overall, this section is crucial for demonstrating the project's impact and presenting a clear path forward for further development.

### **6.1 PERFORMANCE TESTING**

The performance estimation for the pothole detection project is as follows: -

- Correct Pothole Detections: 14.
- Incorrect Pothole Detections: 1.

The project is able to accurately detect potholes in several images with varying numbers of potholes. Out of the tested images, the system has correctly identified potholes in 14 cases. However, there is one case where a non-pothole object may have been mistakenly identified as a pothole, resulting in an incorrect detection.

It is important to note that in some cases, the system has achieved perfect accuracy with no false positives or false negatives. For example:

- In one image, the system detected all 10 potholes correctly with 0 false positives.
- In another image, the system detected 4 potholes with 0 false positives.
- In one image, the system detected 1 pothole with 0 false positives.
- In three images, the system detected 3, 2, and 1 pothole respectively, with 0 false positives.

These results indicate that the system has a high level of accuracy in detecting potholes in certain scenarios, but there is still room for improvement to reduce false positives and false negatives in other cases. Regular monitoring, further testing with diverse datasets, and continuous refinement of the algorithms and image processing techniques can help enhance the performance of the system. Additionally, considering real-world factors such as different lighting conditions, weather conditions, road surfaces, and vehicle speeds, can also contribute to improving the system's accuracy and practical applicability in real-world scenarios.

## **6.2 USABILITY OF PRODUCT / SYSTEM**

The usability of the pothole detection system can be evaluated in terms of its effectiveness, efficiency, and user satisfaction.

**Effectiveness:** The system should be able to accurately detect potholes on the road, capture their locations, and report them to the concerned authorities. The system should have a high detection rate with minimal false positives or false negatives to ensure that all potholes are accurately identified and reported for timely action.

**Efficiency:** The system should be efficient in terms of time and resource consumption. It should process video frames or images in a timely manner, with minimal processing delays or lag, to provide real-time or near real-time results. The system should also utilize computational resources effectively, considering any constraints such as processing power or storage capacity, to ensure efficient performance.

**User Satisfaction:** The system should be user-friendly and easy to use, with a well-designed graphical user interface (GUI) that allows users to capture road images, process them, and report potholes efficiently. The GUI should be intuitive, visually appealing, and provide clear instructions to users. The system should also provide feedback and notifications to users about the detection results and reporting status, ensuring transparency and user confidence.

**Reliability:** The system should be reliable and robust, capable of handling different road conditions, lighting conditions, and weather conditions. It should be able to provide consistent and accurate results over time and perform reliably in different environments.

**Integration:** The system should be seamlessly integrated with other components, such as data handling and storage (Firebase), to ensure smooth data flow and integrity. The system should also be easily integrated with existing road infrastructure management systems or government authorities' databases for effective reporting and follow-up actions.

**Scalability:** The system should be scalable, allowing for future expansion or updates, such as incorporating new image processing techniques, adding more training data, or incorporating feedback from users for continuous improvement.

Overall, the usability of the pothole detection system is crucial to ensure its effective adoption and utilization by users, including government authorities, road maintenance personnel, or other stakeholders, for efficient and prompt pothole detection and repair actions.

### **6.3 LIMITATIONS**

**Weather and Lighting Conditions:** Adverse weather conditions such as heavy rain, snow, or fog, as well as low-light conditions during night-time, can impact the accuracy and reliability of the pothole detection system. Poor visibility or image quality may result in false positives or false negatives, reducing the system's effectiveness.

**Road Conditions:** The system's accuracy may be affected by the quality of the road surface, road markings, and other factors that can impact the appearance of potholes in images or videos. Uneven road surfaces, debris, or road markings may interfere with the detection algorithm's ability to accurately identify potholes, leading to false results.

**System Constraints:** The system may have constraints, such as limited processing power, storage capacity, or battery life, especially when deployed on resource-constrained devices, such as dash cams or mobile devices. These constraints may affect the system's performance, accuracy, and real-time processing capabilities. **Regulatory and Legal Considerations:** There may be regulatory or legal considerations related to data privacy, data handling, and reporting of potholes to government authorities. Compliance with relevant regulations, such as data protection laws or road safety regulations, may pose challenges or limitations to the system's implementation and usage.

**Maintenance and Updates:** The system may require regular maintenance and updates, such as updating the model with new training data, improving image processing techniques, or fixing bugs or issues in the software. Managing and maintaining the system's performance and accuracy over time may require ongoing efforts and resources.

### **6.4 SCOPE OF IMPROVEMENT**

The scope of improvement in this project includes the following areas:

- **Image Processing Techniques:** Continue exploring and experimenting with different image processing techniques to further enhance the quality of the images used for pothole detection. This may involve adjusting parameters, trying different filters, or exploring advanced image processing algorithms to improve image quality and reduce noise, especially in challenging conditions such as low light, rain, or varying road surfaces.



- **Algorithm Optimization:** Analyze and fine-tune the deep learning algorithm used for pothole detection (YOLO) to further improve its accuracy and reduce false detections. This may involve adjusting the model's hyperparameters, retraining the model with additional annotated data, or exploring other advanced algorithms to achieve better results.
- **Data Collection and Labelling:** Ensure that the dataset used for training and testing the pothole detection model is diverse and representative of real-world road conditions. This may involve collecting more data from different road environments, road types, and weather conditions to improve the model's ability to generalize and detect potholes accurately in different scenarios.
- **System Robustness:** Enhance the system's robustness to different weather conditions, lighting conditions, and road surfaces. This may involve incorporating techniques such as data augmentation, transfer learning, or ensemble methods to improve the model's performance in challenging conditions.
- **Real-time Processing:** Explore real-time processing capabilities to enable the system to detect potholes in real-time during driving, rather than processing pre-recorded video frames. This may involve optimizing the system for real-time processing, considering computational constraints, and ensuring low latency to provide timely and actionable information to the authorities for prompt action.
- **User Interface and Integration:** Improve the user interface (GUI) to make it more user-friendly and efficient in capturing and processing road images. Additionally, ensure seamless integration of the system with other components such as the data handling and storage (Firestore), and ensure data integrity, security, and privacy.
- **Testing and Validation:** Conduct rigorous testing and validation of the system in various road conditions, and continuously evaluate its performance to identify any potential issues or limitations and address them in a timely manner.

By addressing these areas for improvement, the pothole detection system can be further enhanced in terms of accuracy, efficiency, and reliability, and contribute to safer and more efficient road infrastructure management.

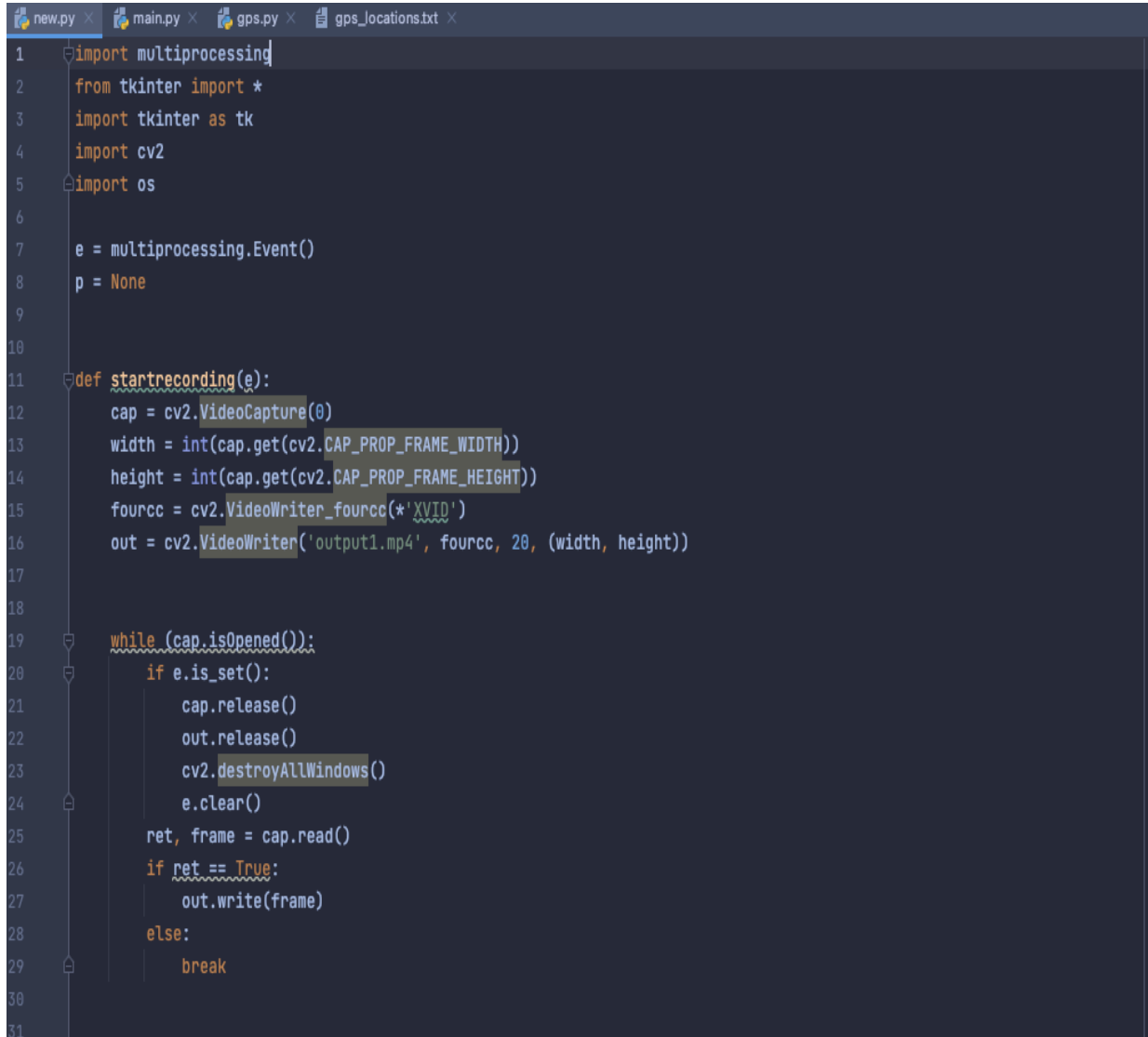
## REFERENCES

- [1] Rufe Wang, Fanyun Xu, Jifang Pei, Wang, Yulin Huang, Jianyu Yang, and Junjie Wu, "An improved faster R-CNN based on MSER DECISION criterion for SAR image ship detection in harbour". IGARSS, Kohala Coast, pp.1322-1325, October 2020.
- [2] BIN LIU, Wencang ZHAO and Qiaoqiao SUN, "Study of Object Detection Based On Faster R-CNN". CAC, Jinan China, pp.6233-6236, January 2018.
- [3] Jianghong Tang, Yingchi Mao, Jing Wang, "Multi-task Enhanced Dam Crack Image Detection Based on Faster R-CNN". International Conference on Image, Vision and Computing, pp.5314-5318, New Zealand, July 2019.
- [4] Wenjing You, Liding Chen, Zhimin Mo, "Soldered Dots Detection of Automobile Door Panels based on Faster R- CNN Model". Chinese Control and Decision Conference (CCDC), China, pp.201-204, June 2019.
- [5] Shih-Chung Hsu, Yu-Wen Wang, "Human Object Identification for Human-Robot Interaction by using Fast R-CNN". Information Technology and Mechatronics Engineering Conference. Laguna Hills, CA, USA, pp.799-803, April 2018.
- [6] Chengji Liu, Yufan Tao, Jiawei Liang, Kai Li, Yihang Chen<sup>5</sup>. "Object Detection Based on YOLO Network", Information Technology and Mechatronics Engineering Conference (ITOEC). Chongqing, China, pp.336-340, June 2019.
- [7] Joseph Redmon<sup>1</sup>, Santosh Divvala<sup>2</sup>, Ross Girshick<sup>3</sup>, Ali Farhadi<sup>4</sup>, "YOLO: Unified, Real-Time Object Detection". IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA, pp.779-788, December 2016.
- [8] Xi Ouyang, Kang Gu, and Pan Zhou, *IEEE* "Spatial Pyramid Pooling Mechanism in 3D Convolutional Network for Sentence-Level Classification", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, pp.2167-2179, July 2018.
- [9] Manjunatha HT, Ajit Danti, "Detection and Classification of Potholes in Indian Roads using Wavelet Based Energy Modules". International Conference on Data Science and Communication. Bangalore, India, pp.1-7, August 2019.

- [10] Shambhu Hegde, Harish V.Mekali, Golla Varaprasad, “Pothole Detection and Inter Vehicular Communication”, IEEE International Conference on Vehicular Electronics and Safety, Hyderabad, India, pp.84-87, March 2015.
- [11] Abhishek kumar, Chakrapani, Dhruba Jyoti Kalita, Vibhav Prakash Singh, “A Modern Pothole Detection technique using Deep Learning”, International Conference on Data, Engineering and Applications, Bhopal, India, pp.1-5, February 2020.
- [12] Ping Ping, Xiaohui Yang, Zeyu Gao, “A Deep Learning Approach for Street Pothole Detection”, IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService), Oxford, UK, pp.198-204, August 2020.
- [13] Felix Kortmann, Kevin Talits, Pascal Fassmeyer, Alexander Warnecke, Nicolas Meier, Jens Heger, Paul Drews, Burkhardt Funk, “Detecting Various Road Damage Types in Global Countries Utilizing Faster R-CNN”, IEEE International Conference on Big Data (Big Data). Atlanta, GA, USA, pp.5563-5571, March 2021.
- [14] Xiaoqing Y, Yazhou Yang, Hao Xu, Weili Li, Jinsheng Deng, “Enhanced Faster-RCNN Algorithm for Object Detection in Aerial Images”, IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, pp.2355-2358, February 2021.
- [15] Zhi Tian, Chunhua Shen Hao, Chen Tong He, “FCOS: Fully Convolutional One-Stage Object Detection”. IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), pp.9627-9636, February 2020.
- [16] Weiqiang Li, Guizhong Liu, “A Single-Shot Object Detector with Feature Aggregation and Enhancement”, IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, pp.3910-3914, September 2019.
- [17] Jiajun Deng, Yingwei Pan, Ting Yao, Wengang Zhou, Houqiang Li, and Tao Mei, “Single Shot Video Object Detector”, IEEE Transactions on Multimedia, pp. 846-858, April 2020.

## ANNEXURE 1

### Programming explanation



```
1 import multiprocessing
2 from tkinter import *
3 import tkinter as tk
4 import cv2
5 import os
6
7 e = multiprocessing.Event()
8 p = None
9
10
11 def startrecording(e):
12     cap = cv2.VideoCapture(0)
13     width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
14     height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
15     fourcc = cv2.VideoWriter_fourcc(*'XVID')
16     out = cv2.VideoWriter('output1.mp4', fourcc, 20, (width, height))
17
18
19 while (cap.isOpened()):
20     if e.is_set():
21         cap.release()
22         out.release()
23         cv2.destroyAllWindows()
24         e.clear()
25     ret, frame = cap.read()
26     if ret == True:
27         out.write(frame)
28     else:
29         break
```

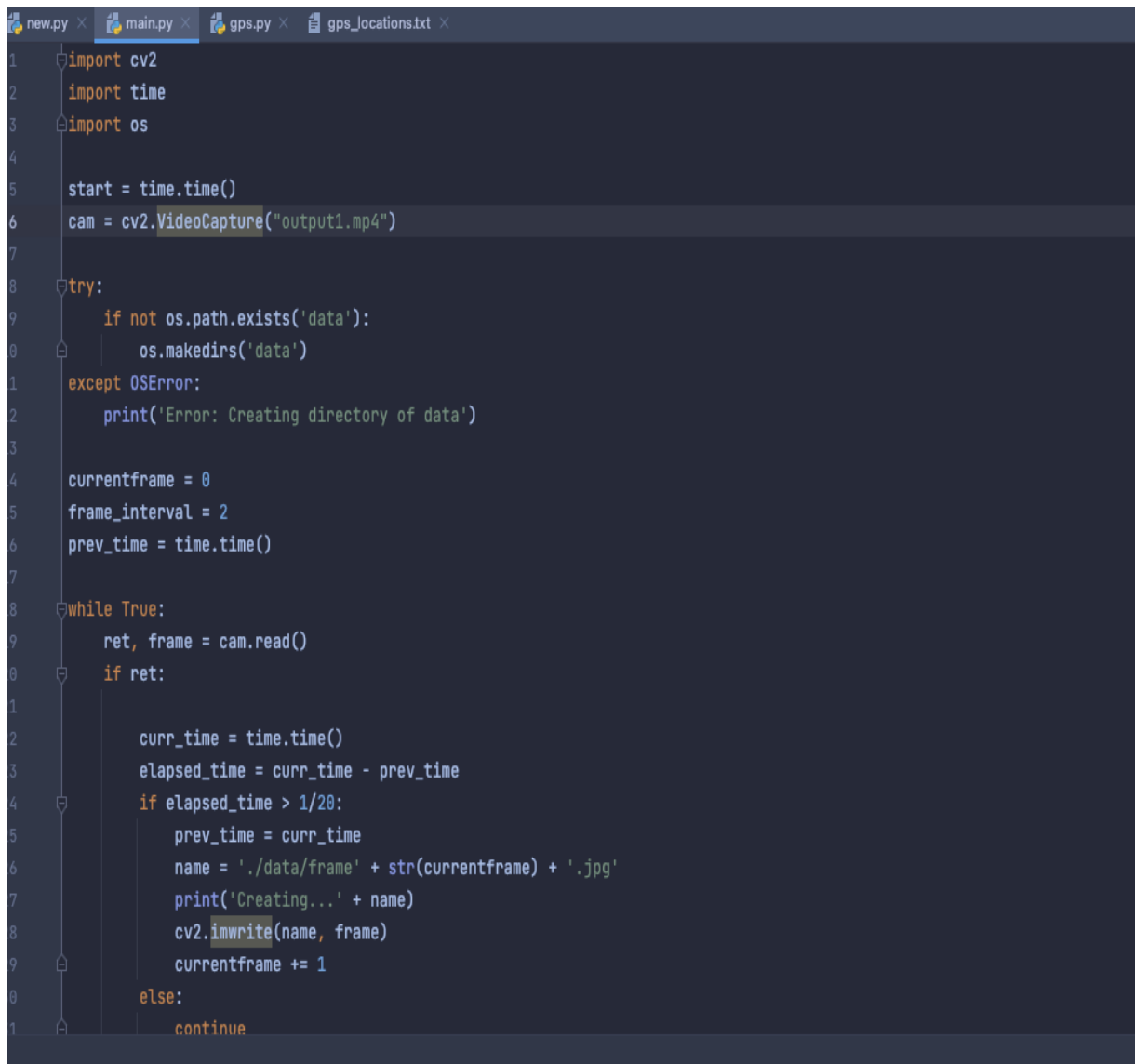
Fig.14. Code for video capturing.

The code captures video frames from a camera and writes them into a video file using OpenCV. It does so in a separate process using multiprocessing, and also runs additional Python scripts (gps.py, main.py, and test.py) while recording. The video recording can be started with a 'Capture' button and stopped with a 'STOP' button in a tkinter GUI window.

```
new.py x main.py x gps.py x gps_locations.txt x
30
31
32 def start_recording_proc():
33     global p
34     p = multiprocessing.Process(target=startrecording, args=(e,))
35     p.start()
36     os.system("python gps.py")
37
38
39 def stoprecording():
40     e.set()
41     p.join()
42     top = Toplevel()
43     top.geometry("400x600")
44
45     os.system('python main.py')
46     os.system('python test.py')
47     root.quit()
48
49
50 if __name__ == "__main__":
51     root = tk.Tk()
52     root.geometry("%dx%d+0+0" % (400, 600))
53     root.config(bg="sky blue")
54     startbutton = tk.Button(root, width=10, height=1, text='Capture', command=start_recording_proc)
55     stopbutton = tk.Button(root, width=10, height=1, text='STOP', command=stoprecording)
56     startbutton.pack()
57     stopbutton.pack()
58
59     root.mainloop()
60
```

Fig.15. Code for video capturing.

This code is extension to the previous part.



```

1 import cv2
2 import time
3 import os
4
5 start = time.time()
6 cam = cv2.VideoCapture("output1.mp4")
7
8 try:
9     if not os.path.exists('data'):
10         os.makedirs('data')
11 except OSError:
12     print('Error: Creating directory of data')
13
14 currentframe = 0
15 frame_interval = 2
16 prev_time = time.time()
17
18 while True:
19     ret, frame = cam.read()
20     if ret:
21
22         curr_time = time.time()
23         elapsed_time = curr_time - prev_time
24         if elapsed_time > 1/20:
25             prev_time = curr_time
26             name = './data/frame' + str(currentframe) + '.jpg'
27             print('Creating...' + name)
28             cv2.imwrite(name, frame)
29             currentframe += 1
30         else:
31             continue

```

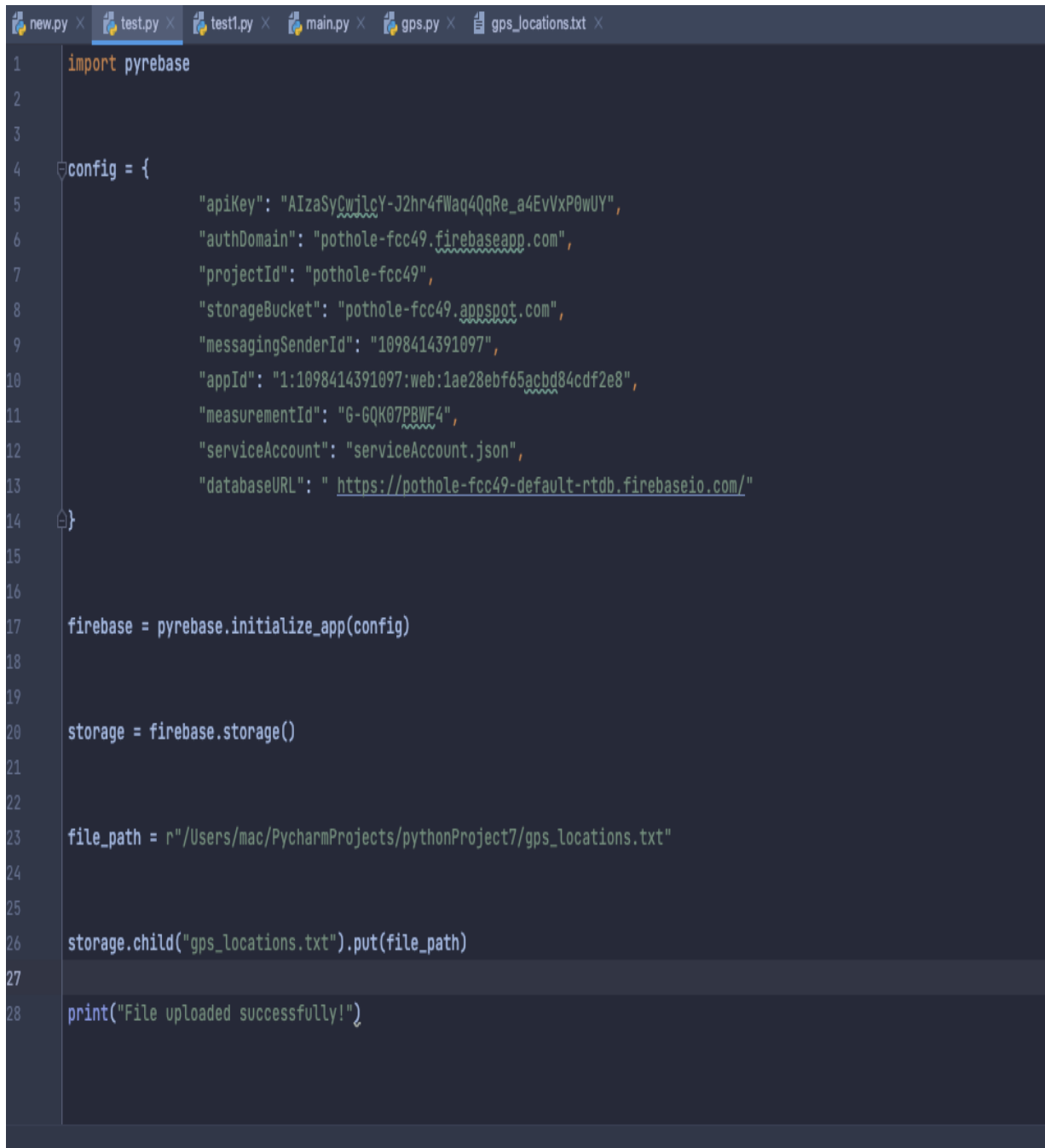
Fig.16. Code for frame conversion.

This code captures frames from a video file using OpenCV (cv2) and saves them as JPEG images in a directory named "data". It sets a frame interval of 1/20 seconds (i.e., 20 frames per second) to save the frames. The code calculates the time consumed during processing and prints it at the end. Finally, it releases the video capture and closes all OpenCV windows.

```
new.py x main.py x gps.py x gps_locations.txt x
14 currentframe = 0
15 frame_interval = 2
16 prev_time = time.time()
17
18 while True:
19     ret, frame = cam.read()
20     if ret:
21
22         curr_time = time.time()
23         elapsed_time = curr_time - prev_time
24         if elapsed_time > 1/20:
25             prev_time = curr_time
26             name = './data/frame' + str(currentframe) + '.jpg'
27             print('Creating...' + name)
28             cv2.imwrite(name, frame)
29             currentframe += 1
30         else:
31             continue
32     else:
33         break
34
35 end = time.time()
36 print("Time consumed in working: ", end - start)
37
38 cam.release()
39 cv2.destroyAllWindows()
```

Fig.17. Code for frame conversion.

This code is continuation for the frame conversion part.

The image shows a PyCharm IDE window with several tabs open: 'new.py', 'test.py', 'test1.py', 'main.py', 'gps.py', and 'gps\_locations.txt'. The 'test.py' tab is active, displaying the following Python code:

```
1 import pyrebase
2
3
4 config = {
5     "apiKey": "AIzaSyCwjlCvY-J2hr4fWaq4QqRe_a4EvVxP0wUY",
6     "authDomain": "pothole-fcc49.firebaseio.com",
7     "projectId": "pothole-fcc49",
8     "storageBucket": "pothole-fcc49.appspot.com",
9     "messagingSenderId": "1098414391097",
10    "appId": "1:1098414391097:web:1ae28ebf65acbd84cdf2e8",
11    "measurementId": "G-6QK07PBWF4",
12    "serviceAccount": "serviceAccount.json",
13    "databaseURL": "https://pothole-fcc49-default-rtdb.firebaseio.com/"
14 }
15
16
17 firebase = pyrebase.initialize_app(config)
18
19
20 storage = firebase.storage()
21
22
23 file_path = r"/Users/mac/PycharmProjects/pythonProject7/gps_locations.txt"
24
25
26 storage.child("gps_locations.txt").put(file_path)
27
28 print("File uploaded successfully!")
```

Fig.18. Code for uploading GPS location to Firebase.

This code demonstrates how to use the Pyrebase library to upload a file to Firebase Storage. It first initializes the Firebase app with the provided configuration, which includes API key, authentication domain, project ID, storage bucket, and other credentials. Then, it specifies the local file path of the file to be uploaded (file path) and uses the put() method of the Firebase Storage to upload the file to the specified storage location ("gps\_locations.txt" in this case). Finally, a success message "File uploaded successfully!" is printed to indicate that the file upload was successful.



```
new.py x main.py x gps.py x gps_locations.txt x
1 import ssl
2 from geopy.geocoders import Nominatim
3 import time
4 start = time.time()
5
6 PERIOD_OF_TIME = 7
7 ssl._create_default_https_context = ssl._create_unverified_context
8 file = open("gps_locations.txt", "w")
9 geolocator = Nominatim(user_agent="geoapiExercises")
10 i=1
11 while True:
12     location = geolocator.geocode("me")
13     file.write("frame "+str(i)+" "+str(location.latitude)+" "+str(location.longitude)+"\n")
14     i=i+1
15     if time.time() > start + PERIOD_OF_TIME: break
16     time.sleep(1)
```

Fig.19. Code for capturing GPS locations.

This code retrieves the current latitude and longitude coordinates using the geopy library's Nominatim geocoding service. It writes the coordinates along with the frame number to a file named "gps\_locations.txt". The code continues to retrieve the coordinates every 1 second until a period of 7 seconds has elapsed. The SSL context is created with `_create_unverified_context` to bypass SSL verification. The execution time is calculated using `time.time()` and compared with the starting time (`start`) to determine when to stop the loop.

```
IP.py x model.py x read2.py x
1 import sys
2 import os
3 from tkinter import *
4
5 window=Tk()
6
7 window.title("Image processing")
8 window.geometry('550x200')
9
10 def run():
11     os.system('python model.py')
12
13 btn = Button(window, text="Run Image Processing", bg="black", fg="green", command=run)
14 btn.grid(column=0, row=0)
15
16 window.mainloop()
```

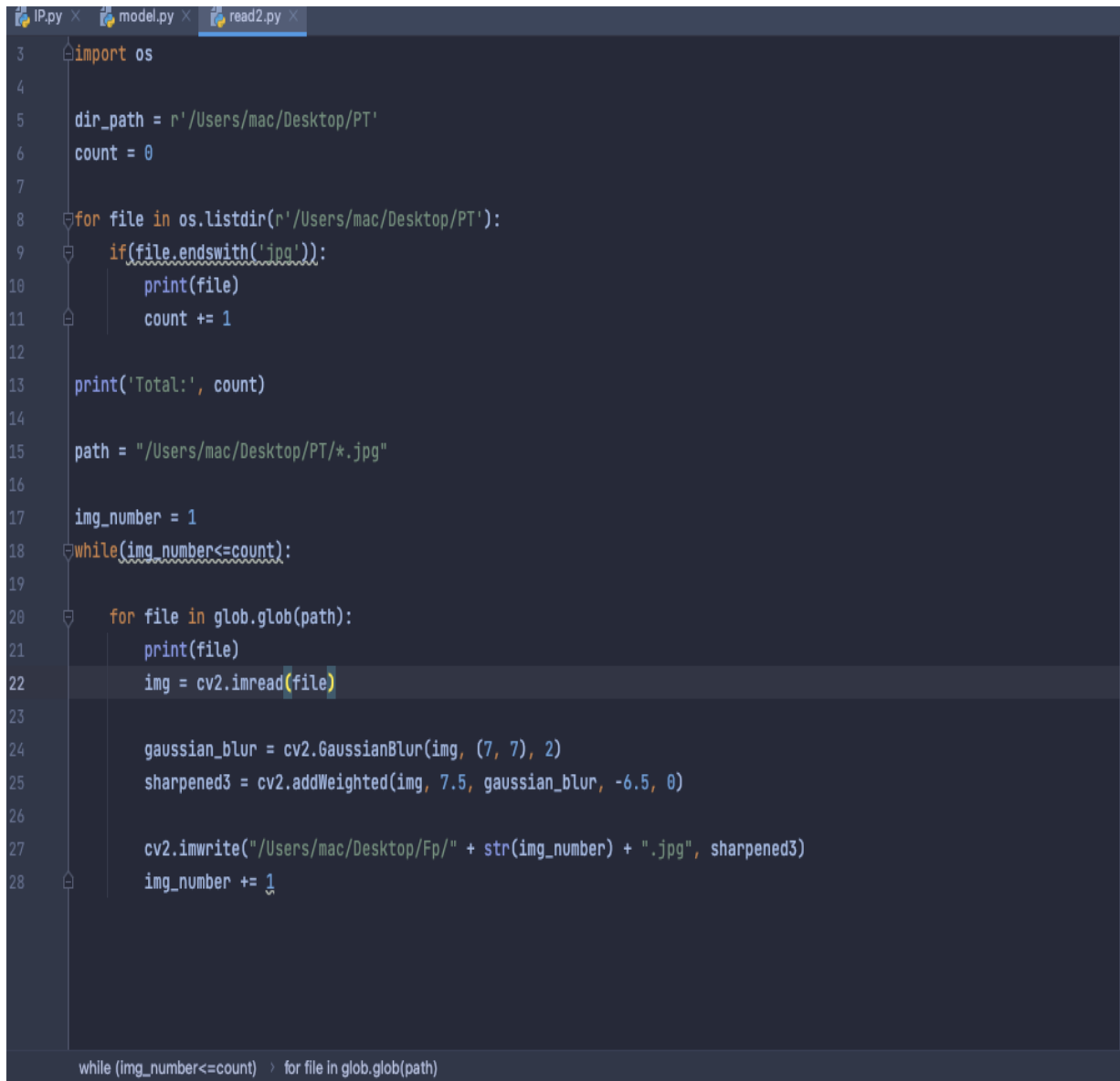
Fig.20. Code for GUI interface for image processing.

This code demonstrates a simple GUI (Graphical User Interface) application using Tkinter in Python. It creates a window with the title "Image processing" and a size of 550x200 pixels. Inside the window, it creates a button labeled "Run Image Processing" with a black background and green text color. When the button is clicked, it calls the run() function which executes the command "python model.py" using the os.system() method. This is typically used to run another Python script (model.py in this case) from the current script. The btn.grid() method is used to position the button in the grid layout of the window. Finally, the window.mainloop() function is called to start the event loop and display the window until it is closed by the user.

```
IP.py x model.py x read2.py x
11     count += 1
12
13     print('Total:', count)
14     path = "/Users/mac/PycharmProjects/pythonProject7/data/*.jpg"
15
16     img_number = 1
17
18     while(img_number<=count):
19         for file in glob.glob(path):
20             print(file)
21             sr = dnn_superres.DnnSuperResImpl_create()
22             img = cv2.imread(file)
23             path = "FSRCNN_x2.pb"
24             sr.readModel(path)
25
26
27             sr.setModel("fsrcnn", 2)
28
29
30             result = sr.upsample(img)
31             cv2.imwrite("/Users/mac/Desktop/PT/" + str(img_number) + ".jpg", result)
32             img_number += 1
33
34
35
36     os.system('python read2.py')
```

Fig.21. Code for applying pre trained image processing model.

This code uses OpenCV (cv2) and its dnn\_superres module to perform image super-resolution. It first counts the number of jpg files in a specific directory, then iterates through each jpg file in that directory. For each file, it reads the image, loads a pre-trained super-resolution model ("FSRCNN\_x2.pb"), sets the model type to "fsrcnn" with a scaling factor of 2, upsamples the image using the super-resolution model, and saves the result to a new file with a incremented number in the file name. Finally, it calls another Python script "read2.py" using the os.system() method.



```

3 import os
4
5 dir_path = r'/Users/mac/Desktop/PT'
6 count = 0
7
8 for file in os.listdir(r'/Users/mac/Desktop/PT'):
9     if(file.endswith('.jpg')):
10         print(file)
11         count += 1
12
13 print('Total:', count)
14
15 path = "/Users/mac/Desktop/PT/*.jpg"
16
17 img_number = 1
18 while(img_number<=count):
19
20     for file in glob.glob(path):
21         print(file)
22         img = cv2.imread(file)
23
24         gaussian_blur = cv2.GaussianBlur(img, (7, 7), 2)
25         sharpened3 = cv2.addWeighted(img, 7.5, gaussian_blur, -6.5, 0)
26
27         cv2.imwrite("/Users/mac/Desktop/Fp/" + str(img_number) + ".jpg", sharpened3)
28         img_number += 1

```

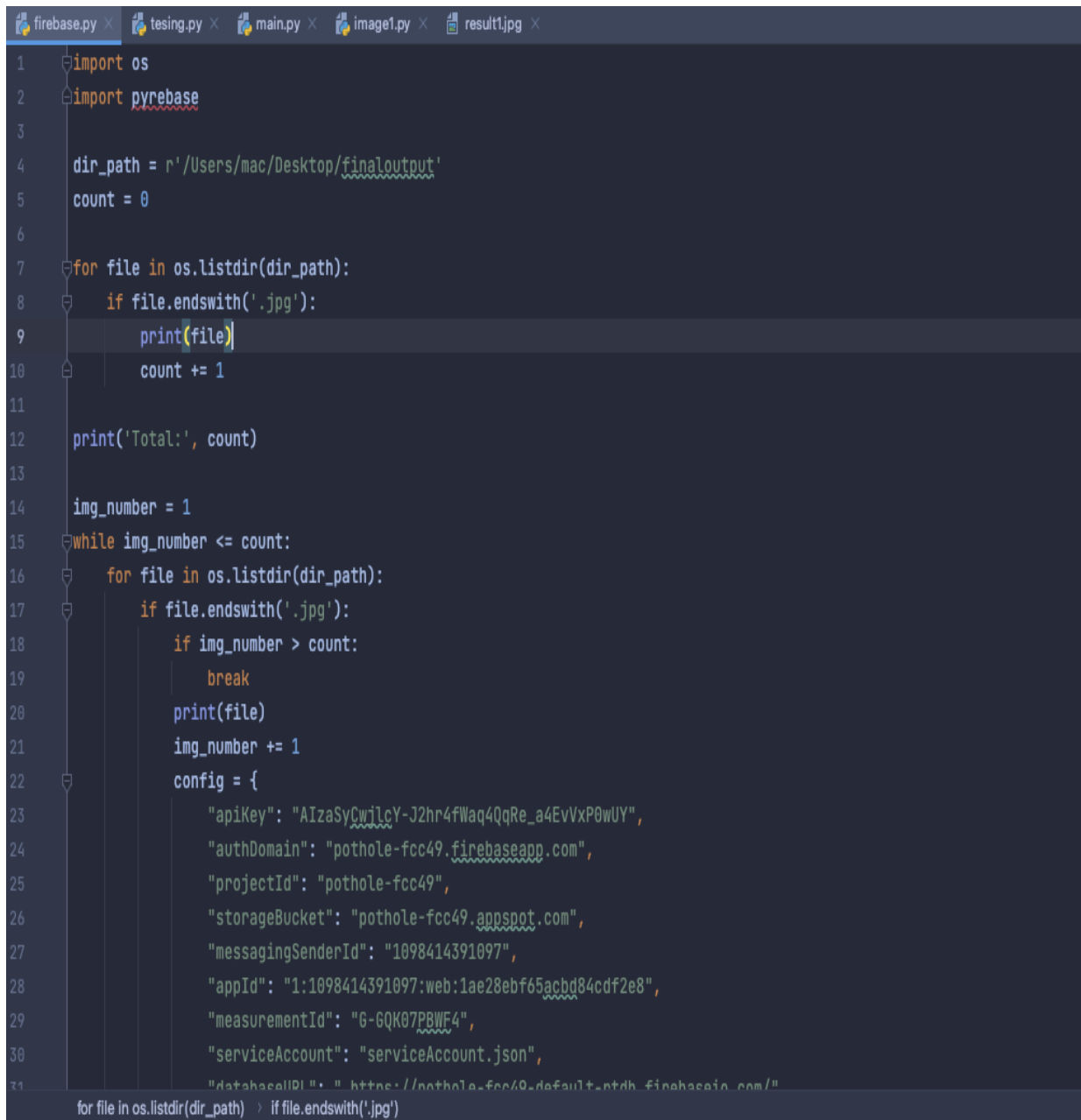
Fig.22. Code for applying sharpening to the image.

This code reads jpg image files from a directory specified by the `dir_path` variable. It counts the number of jpg files in the directory, then iterates through each jpg file. For each file, it reads the image using OpenCV (`cv2`), applies a Gaussian blur with a kernel size of (7, 7) and a sigma of 2, and then performs image sharpening using the `cv2.addWeighted()` function with specified weights. The resulting sharpened image is saved to a new file with an incremented number in the file name in a different directory ("`/Users/mac/Desktop/Fp/`"). The process is repeated for all the jpg files in the directory until the desired number of images (specified by `count`) is processed.

```
1 import cv2
2 import os
3
4 # Set input and output directories
5 input_dir = r'/Users/mac/Desktop/Fp'
6 output_dir = r'/Users/mac/Desktop/finaloutput'
7
8 # Load model and classes
9 with open(os.path.join("project_files", 'obj.names'), 'r') as f:
10     classes = f.read().splitlines()
11
12 net = cv2.dnn.readNet('project_files/yolov4_tiny.weights', 'project_files/yolov4_tiny.cfg')
13 model = cv2.dnn_DetectionModel(net)
14 model.setInputParams(scale=1 / 255, size=(416, 416), swapRB=True)
15
16 # Loop through input images and process them
17 for i, file in enumerate(os.listdir(input_dir)):
18     if file.endswith('.jpg'):
19         print(f'Processing image {i+1} of {len(os.listdir(input_dir))} ({file})')
20         # Load input image
21         img = cv2.imread(os.path.join(input_dir, file))
22         # Run object detection
23         classIds, scores, boxes = model.detect(img, confThreshold=0.6, nmsThreshold=0.4)
24         # Draw bounding boxes on image
25         for (classId, score, box) in zip(classIds, scores, boxes):
26             cv2.rectangle(img, (box[0], box[1]), (box[0] + box[2], box[1] + box[3]),
27                           color=(0, 255, 0), thickness=2)
28         # Save output image
29         output_file = os.path.join(output_dir, f'img_{i+1}.jpg')
30         cv2.imwrite(output_file, img)
```

Fig.23. Code YOLO object detection algorithm.

This code uses the YOLOv4-tiny model for object detection on input images in a directory. It loads the model weights and configuration, sets input parameters, and then iterates through each jpg image file in the input directory. It performs object detection on the images, draws bounding boxes around detected objects, and saves the processed images to an output directory. Finally, it runs another Python script (firebase.py) for further processing or analysis on the processed images, potentially involving Firebase integration.



```

1  import os
2  import pyrebase
3
4  dir_path = r'/Users/mac/Desktop/finaloutput'
5  count = 0
6
7  for file in os.listdir(dir_path):
8      if file.endswith('.jpg'):
9          print(file)
10         count += 1
11
12     print('Total:', count)
13
14     img_number = 1
15     while img_number <= count:
16         for file in os.listdir(dir_path):
17             if file.endswith('.jpg'):
18                 if img_number > count:
19                     break
20                 print(file)
21                 img_number += 1
22                 config = {
23                     "apiKey": "AIzaSyCwj1cY-J2hr4fWaq4QqRe_a4EvVxP0wUY",
24                     "authDomain": "pothole-fcc49.firebaseio.com",
25                     "projectId": "pothole-fcc49",
26                     "storageBucket": "pothole-fcc49.appspot.com",
27                     "messagingSenderId": "1098414391097",
28                     "appId": "1:1098414391097:web:1ae28ebf65acbd84cdf2e8",
29                     "measurementId": "G-GQK07PBWF4",
30                     "serviceAccount": "serviceAccount.json",
31                     "databaseURL": "https://pothole-fcc49.firebaseio.com/"
32                 }
33                 for file in os.listdir(dir_path):
34                     if file.endswith('.jpg')

```

Fig.24. Code for uploading images to firebase.

This code uploads image files (with .jpg extension) from a local directory to Firebase Cloud Storage using Pyrebase, a Python wrapper for the Firebase API. It iterates through the image files in the specified local directory (`dir_path`), and for each file, it initializes a Firebase app with the given configuration (`config`) that includes the Firebase project credentials, and uploads the file to Firebase Cloud Storage using the `put()` method. The `path_on_cloud` variable specifies the path of the file in the Firebase Cloud Storage, and the `path_local` variable specifies the local path of the file on the system. This code can be used to upload image files to Firebase Cloud Storage for further processing, sharing, or storage in a Firebase project.

## ANNEXURE 2

### 8th semester outcome details – Paper Accepted.

Regarding paper decision External Inbox x

2:08 PM (5 hours ago) ☆ ↶ ⋮

**ICCS** . <iccs@jpu.co.in>  
to 2019006615.akshat, me, 2019563653.amartya, sonia.setia ▼

Dear Sir/Mam,  
Greetings from ICCS KILBY 100 conference!

We have reached the final decision regarding your manuscript with ID 570 to our conference KILBY100 ICCS 2023.

Our decision is to: Accept Submission and it will be submitted to the publication phase.

Submit the camera ready paper on the same mail id. as per the details in Template, Copyright Form and Author Guidelines are available here: <https://ajp.scitation.org/apc/authors/preppapers>

Once you have registered at <https://conferences.jpu.in/iccs/RegistrationForm.aspx>, please proceed to make the payment as soon as possible and share the proof regarding the same to ensure a successful submission at [iccs@jpu.co.in](mailto:iccs@jpu.co.in)

If you have any queries or require assistance at any stage of this process, feel free to contact us. We will be more than happy to help.

Warm Regards,

Team ICCS  
KILBY100  
7th International Conference on Computing Sciences 2023

### ANNEXER 3

7th semester outcome details – Paper presented

	<b>7<sup>th</sup> International Conference on Trends in Electronics and Informatics (ICOEI 2023)</b> 11-13, April 2023   Tirunelveli, India <a href="http://icoei.com/2023/">http://icoei.com/2023/</a> <a href="mailto:icoei.conf@gmail.com">icoei.conf@gmail.com</a>
<b>Payment Receipt</b>	
<b>Paper ID</b>	ICOEI2023-007-154
<b>Paper Title</b>	A Detailed Review on Object Detection Algorithms
<b>Paid by</b>	Sonia Setia, Akshat Shukla, Amartya Raj, Abhimanyu Rathore
<b>Amount Paid</b>	₹ 6,500/- <i>Rupees Six Thousand Five Hundred only</i>
 Dr. R. Karthik Ganesh Organizing Secretary	