

A Project Report
On
Pothole Detection Using Deep Learning

Submitted in partial fulfilment of the requirement of

University of Mumbai

For the Degree of

Bachelor of Engineering

in

COMPUTER ENGINEERING

Submitted by

**Joel Cherian
Joseph Blessingh
Harun Joe**

Supervised by

Mr. Amroz K. Siddiqui



**Department of Computer Engineering
Fr. Conceicao Rodrigues Institute of Technology
Sector 9A, Vashi, Navi Mumbai - 400703**

UNIVERSITY OF MUMBAI

2019-2020

APPROVAL SHEET

This is to certify that the project entitled
“Pothole Detection Using Deep Learning”

Submitted by

Joel Cherian	101608
Joseph Blessingh	101621
Harun Joe	101660

Supervisors : _____

Project Coordinator : _____

Examiners : 1. _____

2. _____

Head of Department : _____

Date :

Place :

Declaration

We declare that this written submission for B.E. Declaration entitled "**Pothole Detection Using Deep Learning**" represent our ideas in our own words and where others' ideas or words have been included. We have adequately cited and referenced the original sources. We also declared that we have adhere to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any ideas / data / fact / source in our submission. We understand that any violation of the above will cause for disciplinary action by institute and also evoke penal action from the sources which have thus not been properly cited or from whom paper permission have not been taken when needed.

Project Group Members:

1. Joel Cherian, 101608

2. Joseph Blessingh, 101621

3. Harun Joe, 101660

Abstract

Potholes in roads constitute a major problem for citizens acting as pedestrians as well as vehicular drivers. Municipal Bodies which constitute the engineers and workers are responsible to detect damage to roads, distress, etc. Assessing very single part of the road is highly time-consuming, requires a lot of manpower and cannot be done in an efficient way to safety the needs of the citizens.

The system proposed can be applied to make this matter more efficient by automating the detection and classification of damages to roads, potholes and cracks. A mobile application is used for identification of potholes with Convolution Neural Networks and visualization of potholes on map. The method includes installing a camera inside of a vehicle ensuring that the maximum amount of road will be captured. Citizens also contribute in the process of reporting potholes which eases the load of the authorities when facing issues in covering all areas. Municipal Authority will be provided with visualized data on a map having markers to identify each pothole and colour coded roads to indicate severity of the roads with potholes along with the locations which can be used for maintenance.

Contents

Abstract	iii
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Background	2
1.2 Motivation	2
1.3 Aim and Objective	2
1.4 Report Outline	3
2 Study Of the System	4
2.1 Related Works	5
2.1.1 A Real-Time Pothole Detection Approach For Intel- ligent Transportation System	5
2.1.2 Image-Based Pothole Detection System For ITS Ser- vice And Road Management System	5
2.1.3 Pothole Detection System Using a Black-box Camera	5
2.1.4 Automatic Pothole and Speed Breaker Detection us- ing Android System	5
2.1.5 Detecting and counting of pothole using image pro- cessing techniques	6
2.1.6 Automatic Detection and Notification of Potholes and Humps on Roads to Aid Drivers	6
2.2 Literature Review	6
2.2.1 Multi-Agent Deep Reinforcement Learning for Multi- Object Tracker	6
2.2.2 Automated Image Annotation based on YOLOv3 .	7
2.2.3 About Deep Learning	7
2.2.4 Real-time object detection and classification of small figures in image processing in YOLOv2	8

2.2.5	NoSQL Database - Google's Firebase: A Review . .	8
2.3	Deep Learning Approaches	8
2.3.1	R-CNN	8
2.3.2	FAST-RCNN	9
2.3.3	FASTER-RCNN	9
2.3.4	SSD (SINGLE SHOT DETECTOR)	10
2.3.5	YOLO	11
3	Proposed System	13
3.1	Problem Statement	14
3.2	Scope	14
3.3	Proposed System	14
3.3.1	Approach	14
3.3.2	Pothole Detection Model	15
3.3.3	Flask Server for deploying Detection Model	15
3.3.4	Mobile Application	16
3.3.4.1	User Authentication for Municipal and Cit- izen	16
3.3.4.2	Citizen Features	16
3.3.4.3	Municipal Features	17
4	Design Of the System	18
4.1	Requirement Engineering	19
4.1.1	Requirement Elicitation	19
4.1.2	Software lifecycle model	19
4.1.3	Requirement Analysis	20
4.1.3.1	Data Flow Diagram	20
4.1.3.2	Use Case Diagram	22
4.1.3.3	Cost Analysis	22
4.1.3.4	Software Requirements	23
4.1.3.5	Hardware Requirements	23
4.2	System architecture	24
4.2.1	Block Diagram	24
4.2.2	UI/UX diagram	25
5	Result and Discussion	31
5.1	Observations	32
5.1.1	Collection of Data	32
5.1.2	Annotation	32
5.1.3	Training	33

5.1.4	Mobile App Development	34
5.1.5	Application Program Interface (API)	35
5.1.6	Database (Firestore NoSQL)	35
5.2	Important Code Snippets	36
5.2.1	API that returns JSON with classes found in images	36
5.2.2	Function to add Marker objects into a List and used for generating Markers	38
5.2.3	Distance Matrix API	38
5.2.4	Roads API	39
5.3	Testing	41
6	Conclusion & Future Scope	42
	References	44
	Acknowledgement	45
	Appendix A: Timeline Chart	47
	Appendix B: Publication Details	49

List of Figures

2.1	R-CNN	9
2.2	FAST-RCNN	10
2.3	FASTER-RCNN	11
2.4	Single shot detector Vs YOLO	12
2.5	YOLO	12
4.1	Data Flow Diagram LEVEL: 0	20
4.2	Data Flow Diagram LEVEL: 1	20
4.3	Data Flow Diagram LEVEL: 2	21
4.4	Use Case Diagram Of The System	22
4.5	Block Diagram	24
4.6	1. Front Page Of Application	25
4.7	2. Create A Free Account	25
4.8	3. Login/SignUp	26
4.9	4. Municipal Authority/Citizens Page	26
4.10	5. Enable Location Before Capturing Image	27
4.11	6. Capture Image	27
4.12	7. Select Image From Gallery	28
4.13	8. Selected Image Results From Server	28
4.14	9. Selected Image Details	29
4.15	10. View Map (Vashi Area)	29
4.16	11. View Map (Vidyavihar Area)	30
6.1	Timeline Chart For Project	48

List of Tables

4.1	Requirement Elicitation	19
5.1	Dataset information for the detector	33
5.2	Training statistics	34
5.3	Testing Report	41

Chapter 1

Introduction

1.1 Background

Potholes are simply areas of road surface that have cracked, worn away, and eventually formed a hole and start out as tiny cracks. Potholes make a ride bumpy and potentially dangerous, as potholes can damage vehicle tires and even affect the alignment of a vehicle's wheels. Potholes develop naturally as a result of many scientific forces at work. The friction of the tires with the roadway surface heats up the road and causes it to expand. Over time, this expansion can lead to the formation of cracks in the roadway surface. Water can then seep into these cracks. Fixing a pothole requires a crew to clean all the loose rock and dirt out of the pothole before filling it with a mixture of hot or cold asphalt (depending upon the time of year) to create a patch in the roadway surface. The manual identification of potholes can be avoided by introducing an automated approach like object detection, image pre-processing and convolutional neural network algorithms.

1.2 Motivation

Hitting a pothole can not only cause damage to a vehicle's shocks and suspension, it can also cause the driver to lose control of his or her car. Potholes can cause truck accidents and motorcycle accidents in addition to poor road conditions that often result in serious car accident injuries. Moreover, motorcycle drivers are at an increased risk for serious and fatal injuries due to the dangerous road conditions potholes create. When a vehicle hits a deep pothole, the impact is similar to that of a collision at 56 km/h. The idea drives to lend a helping hand to the municipal authorities and citizens to automate the process of detecting and reporting it.

1.3 Aim and Objective

Our aim is to develop a mobile application for the citizens and municipal authorities which is capable of detecting potholes and represent the detection on a Map. This map can be viewed by the citizen and authority to identify the location of potholes as well as the current condition of the road. The objectives are to develop an object detection model for detecting Potholes using Deep Learning. It also includes the integration of the model with the mobile application. The mobile application will also be able to give the user a visualized view of potholes using a map. This is to accomplish the aim of reducing efforts and time on manually checking every single part of the road by developing a reliable system.

1.4 Report Outline

The report of our project mentions our approach for a solution to the problem of manual detection and fixing of potholes conducted by Municipal Authorities. The methods used have been explained along with the individual components used have been described to the fullest. It also provides detailed information on the design of the system.

The implementation section for this project is done by understanding image processing and deep learning techniques with the help of Research Papers. A comparison was made to understand the advantages and disadvantages of using different deep learning algorithms. We conclude that Convolution Neural Network is the best algorithm as it gives the highest accuracy. A mobile application is developed for the Citizens and Municipal Authority for using the functionalities. A server is handled to process object detection requests for detecting potholes. This server is used by the mobile application. The data gathered related to potholes are stored in Firebase acting as a cloud storage.

Chapter 2

Study Of the System

2.1 Related Works

2.1.1 A Real-Time Pothole Detection Approach For Intelligent Transportation System

The location of pothole is an important aspect. The advantage of marking the location of pothole is to identify and fix the pothole as soon as possible without the traditional means of repairing the pothole. In[1], space interpolation method is used to obtain exact location of the pothole, the distance between the start point and the pothole followed by the distance between end point and the pothole is obtained by using a GPS module on the mobile.

2.1.2 Image-Based Pothole Detection System For ITS Service And Road Management System

A system to detect the pothole using a optical device mounted on the vehicle is explained by[2]. This approach uses candidate region extraction to identify the pothole and it is first segmented by various techniques like Otsu,Histogram shape based thresholding. This proposed system gave an accuracy of 73.5 percentage with a precision of 80 percentage, the location and severity of the pothole is given to the center i.e pothole alert service and road management system.

2.1.3 Pothole Detection System Using a Black-box Camera

It is important to differentiate the lanes in road from the pothole, the continuous shades of lane on the road can be avoided and the potholes can be obtained. In this paper [3], Cascade detector method is employed to distinguish pothole from similar types of shades and road patches. This method makes use of a black box camera fitted on top of a vehicle which removes objects like shades that changes its shape irrespective of a pothole.

2.1.4 Automatic Pothole and Speed Breaker Detection using Android System

This system make use of accelerometer to detect pothole and its severity within a scale(1-3). The andriod application records using the 3-axis acceleration to monitor the roads. These events along with their severity and the location co-ordinates are stored in the local database of the mobile device[4]. This local database is synchronized with the global database as

soon as the session ends. The synchronization is done in such a way that the previous road surface condition is replaced by the current one.

2.1.5 Detecting and counting of pothole using image processing techniques

The detection of pothole was using thresholding technique of Otsu's method. The image was converted into lower resolution for faster computation and then transformed into grayscale to lower the pixel intensity as compared to the RGB scale. Then the median and Gaussian filter is applied to reduce the noise, edge detection like canny edge was used to detect the edges of pothole. Black and white-Convexhull was used for generation of convex hull image from binary image. It computes the convex hull of each connected component of black and white individually. The convex hulling helps in shaping the connected black and white components i.e. pothole segmented area which provides better results for pothole detection[5]. By find the number of connected components in binary image i.e. the segmented image we can find number of pothole present. Eight neighborhood pixels where considered for identifying connected components.

2.1.6 Automatic Detection and Notification of Potholes and Humps on Roads to Aid Drivers

The system is using a micro-controller to calculate the threshold value. Vehicle was used to record all the threshold value of roads to be send to the driver as a notification. The proposed system considers the presence of potholes and humps. However, it does not consider the fact that potholes or humps get repaired by concerned authorities periodically[6]. This system can be further improved to consider the above fact and update server database accordingly. Also, Google maps and SATNAV can be integrated in the proposed system to improve user experience.

2.2 Literature Review

2.2.1 Multi-Agent Deep Reinforcement Learning for Multi-Object Tracker

Multi-object tracking has been a key research subjects in many computer vision applications. It proposes a novel approach based on multi-agent deep reinforcement learning (MADRL) for multi-object tracking to solve the problems in the existing tracking methods, such as a varying number of targets, noncausal, non-realtime, etc. At first it chooses YOLO V3 to

detect the objects included in each frame. And unsuitable candidates were screened out and the rest of detection results are regarded as multiple agents and forming a multi-agent system. Independent Q-Learners (IQL) is used to learn the agents' policy, in which each agent treats other agents as part of the environment. Then conducted offline learning in the training and online learning during the tracking[7]. These experiments demonstrate that the use of MADRL achieves better performance than the other state-of-art methods in precision, accuracy and robustness.

2.2.2 Automated Image Annotation based on YOLOv3

A typical pedestrian protection system requires sophisticated hardware and robust detection algorithms. To solve these problems the existing systems use hybrid sensors where mono and stereo vision merged with active sensors. One of the most assuring pedestrian detection sensors is far infrared range camera. The classical pedestrian detection approach based on Histogram of oriented gradients is not robust enough to be applied in devices which consumers can trust. An application of deep neural network-based approach is able to perform with significantly higher accuracy. However, the deep learning approach requires a high number of labeled data examples. The investigation presented in this paper aimed the acceleration of pedestrian labeling in far-infrared image sequences. In order to accelerate pedestrian labeling in far-infrared camera videos, we have integrated the YOLOv3 object detector into labeling software. The verification of the pre-labeled results was around eleven times faster than manual labeling of every single frame[8]. To increase the accuracy of the detector it needs to decrease the threshold of probability because it is filtering low prediction samples.

2.2.3 About Deep Learning

Deep learning is a machine learning technique which teaches computers to perform activities that comes naturally to humans and by learning every day to day ventures. In deep learning, a computer model learns to classify tasks directly from text, sound, images or video clips. Deep learning models can achieve state-of-the-art accuracy, sometimes even surpassing human level performance. Models are trained by using huge labeled data sets and neural network architectures that contain various layers. Nowadays, deep learning is achieving higher levels of recognition accuracy than ever before. Recent advancements in deep learning has improved to such stage that it outperforms humans in classifying objects in images[9].

2.2.4 Real-time object detection and classification of small figures in image processing in YOLOv2

The paper uses YOLO V2 and cnn to identify small objects in traffic. The time taken for this model is more compared to other YOLO models. To improve the YOLO aspects four stage process was implemented where first stage is the partial implementation of YOLO V2, second stage where the RoIs are cropped and the third and fourth stage performs classification and filtering. The results were computed in a video and google map to view snapshots[10]. YOLO V2 can be applied were only single class is to be trained and lightweight classifier like CNN to subclassify and improve the quality of the detection with minimal processing.

2.2.5 NoSQL Database - Google's Firebase: A Review

The need to process large amount of data is necessary and the traditional means of processing the required or all data is impossible. A real time database solution is put forward by Google for cloud storage, authentication and many other security aspects containing the efficiency of the system. The paper[11] explains the NoSQL types of databases like key-value store, document databases, wide-column Store, graph databases. Firebase provides multiple services so that the database don't have got to external links for storage, validation and hosting individually.

2.3 Deep Learning Approaches

2.3.1 R-CNN

RCNN algorithm proposes a bunch of boxes in the image and checks if any of these boxes contain any object. RCNN uses selective search to extract these boxes from an image (these boxes are called regions). We first take a pre-trained convolutional neural network. Then, this model is retrained. We train the last layer of the network based on the number of classes that need to be detected. The third step is to get the Region of Interest for each image. We then reshape all these regions so that they can match the CNN input size. After getting the regions, we train SVM to classify objects and background. For each class, we train one binary SVM. Finally, we train a linear regression model to generate tighter bounding boxes for each identified object in the image.

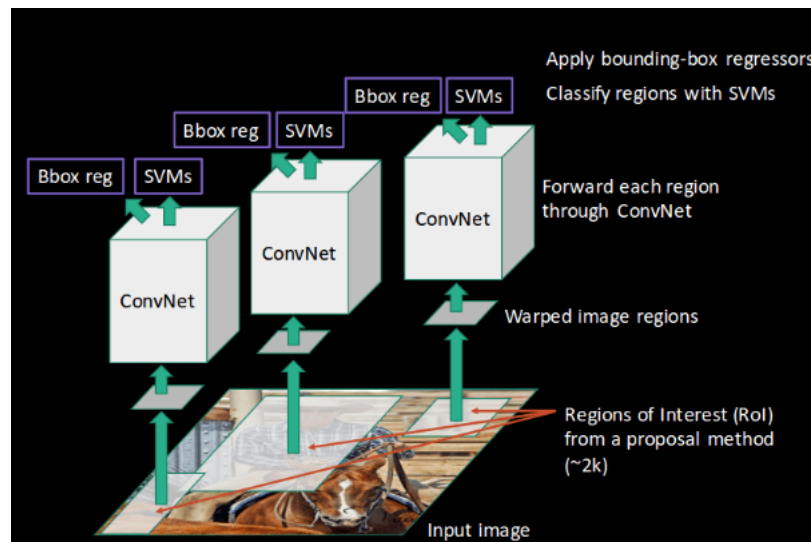


Figure 2.1: R-CNN

2.3.2 FAST-RCNN

Fast RCNN uses a single model which extracts features from the regions, divides them into different classes, and returns the boundary boxes for the identified classes simultaneously. We take an image as an input. This image is passed to a ConvNet which in turns generates the Regions of Interest. A RoI pooling layer is applied on all of these regions to reshape them as per the input of the ConvNet. Then, each region is passed on to a fully connected network. A softmax layer is used on top of the fully connected network to output classes. Along with the softmax layer, a linear regression layer is also used parallelly to output bounding box coordinates for predicted classes.

2.3.3 FASTER-RCNN

Fast R-CNN, on the other hand, passes the entire image to ConvNet which generates regions of interest (instead of passing the extracted regions from the image). Also, instead of using three different models (as we saw in R-CNN), it uses a single model which extracts features from the regions, classifies them into different classes, and returns the bounding boxes. All these steps are done simultaneously, thus making it execute faster as compared to R-CNN. Fast R-CNN is, however, not fast enough when applied on a large dataset as it also uses selective search for extracting the regions. Faster R-CNN fixes the problem of selective search by replacing it with Region Proposal Network (RPN). We first extract feature maps from the input image using ConvNet and then pass those maps through a RPN which returns object proposals. Finally, these maps are classified and the

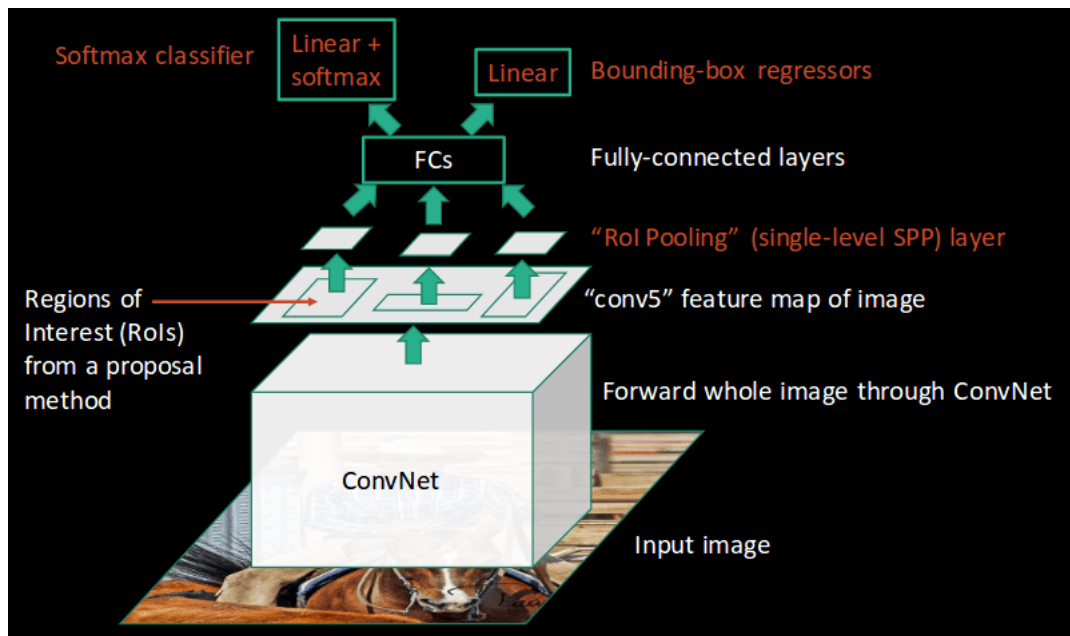


Figure 2.2: FAST-RCNN

bounding boxes are predicted.

2.3.4 SSD (SINGLE SHOT DETECTOR)

SSD is designed for object detection in real-time. Faster R-CNN uses a region proposal network to create boundary boxes and utilizes those boxes to classify objects. While it is considered the start-of-the-art in accuracy, the whole process runs at 7 frames per second. Far below what a real-time processing needs. SSD speeds up the process by eliminating the need of the region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN's accuracy using lower resolution images, which further pushes the speed higher. According to the following comparison, it achieves the real-time processing speed and even beats the accuracy of the Faster R-CNN. (Accuracy is measured as the mean average precision mAP: the precision of the predictions.

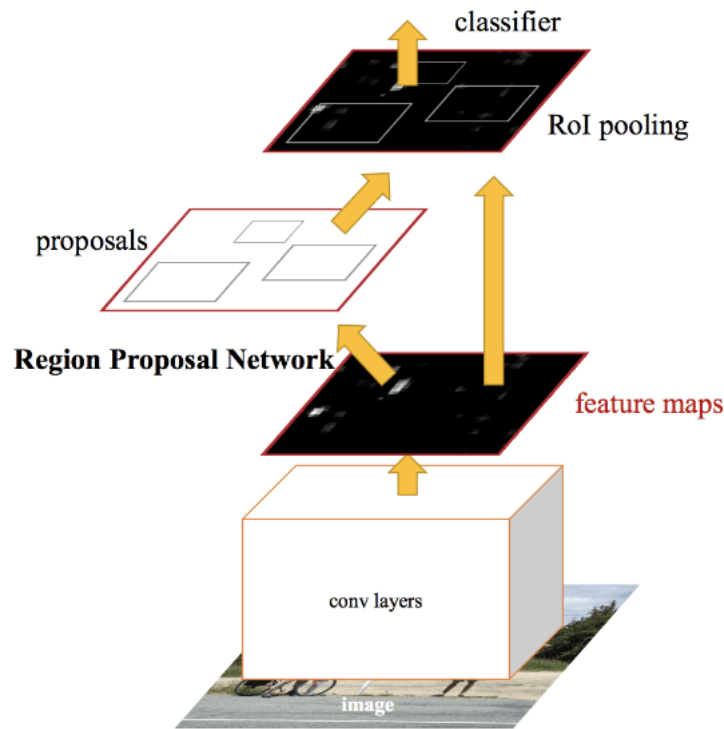


Figure 2.3: FASTER-RCNN

2.3.5 YOLO

Compared to other region proposal classification networks (fast RCNN) which perform detection on various region proposals and thus end up performing prediction multiple times for various regions in a image, Yolo architecture is more like FCNN (fully convolutional neural network) and passes the image ($n \times n$) once through the FCNN and output is ($m \times m$) prediction. This the architecture is splitting the input image in $m \times m$ grid and for each grid generation 2 bounding boxes and class probabilities for those bounding boxes. Note that bounding box is more likely to be larger than the grid itself.

Biggest advantages: 1.Speed (45 frames per second — better than realtime) 2.Network understands generalized object representation (This allowed them to train the network on real world images and predictions on artwork was still fairly accurate). 3.faster version (with smaller architecture) — 155 frames per sec but is less accurate.

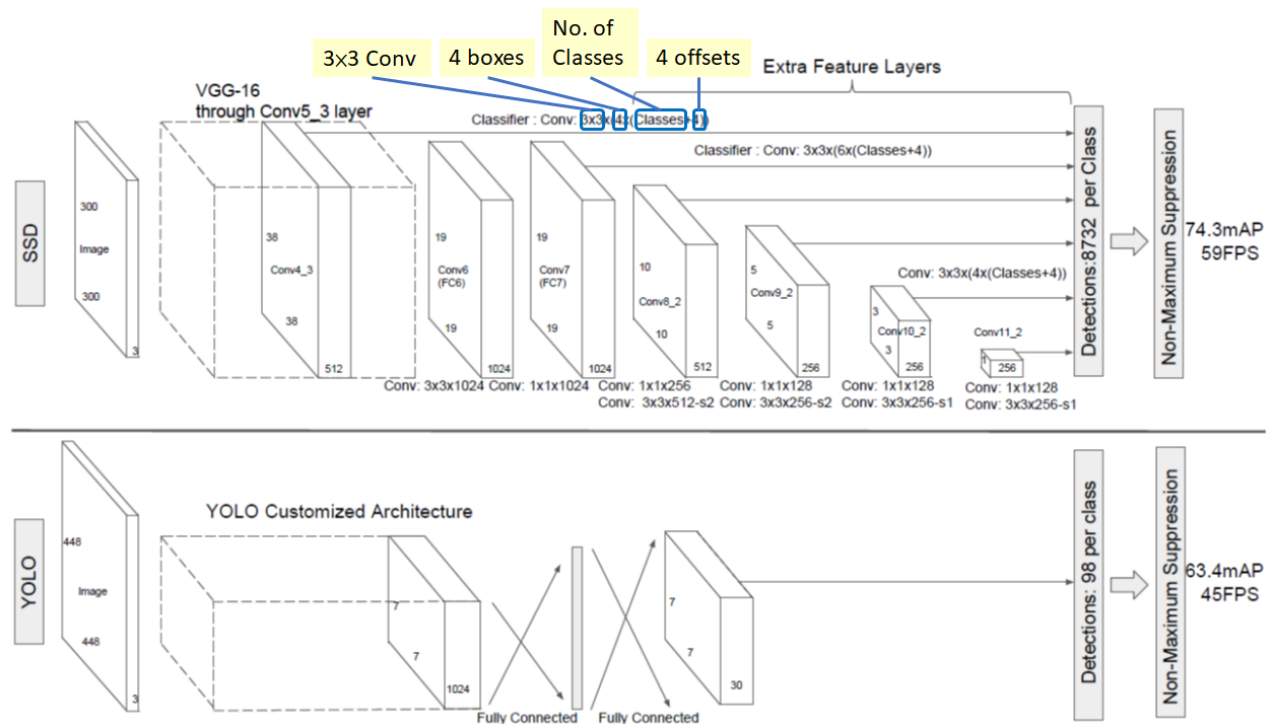


Figure 2.4: Single shot detector Vs YOLO

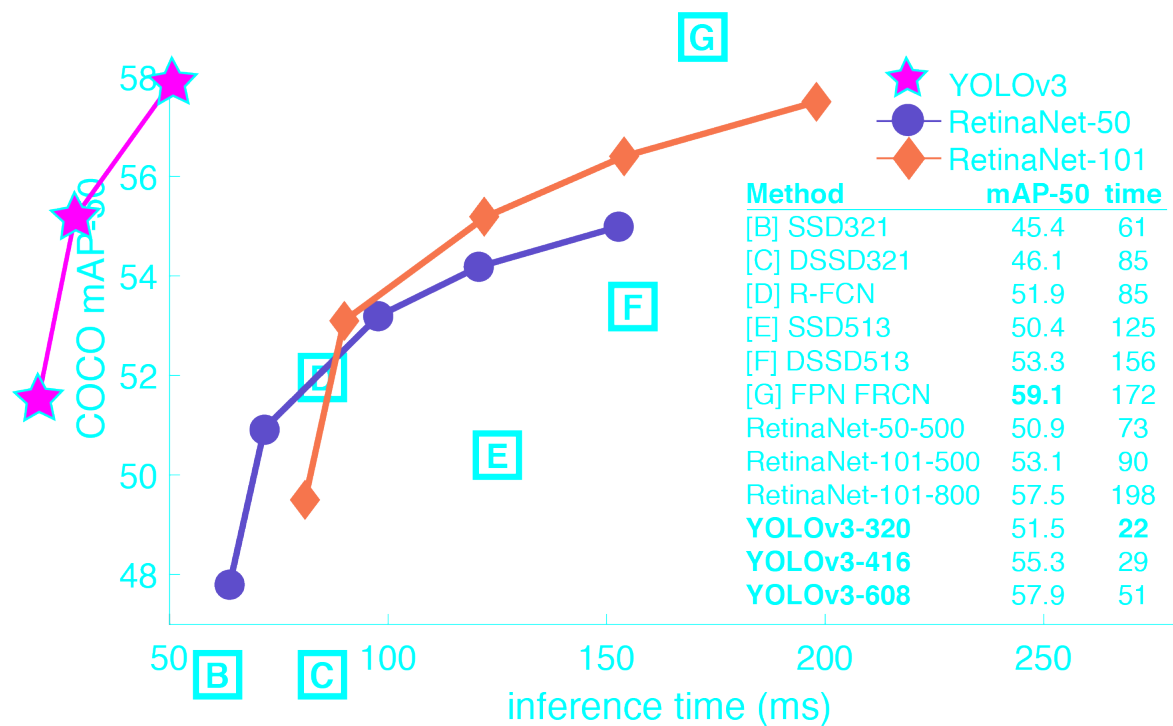


Figure 2.5: YOLO

Chapter 3

Proposed System

3.1 Problem Statement

The procedure followed for repairing potholes by the Municipal Authorities requires them to inspect the roads with manual labour. The detected potholes then need to be fixed which takes up a lot of time. Considering time to be an essential factor when monsoons are up, the process of detecting and repairing potholes needs to be agile. The solution offered to the problem should be capable of automating the process of detecting potholes. It should also include the inputs of citizens to help with the detection of potholes and reporting it which can make the process of detection easier for the authorities. The system should be able to represent the potholes identified with markers on map along with the state of severity of the roads determined by the count of potholes. Municipal body will be able to maintain the roads with data about the conditions and location of roads.

3.2 Scope

The scope of the project involves detection of potholes by Deep Learning algorithm using a mobile application for the Municipal Authorities and Citizens. It is an improvement over the conventional methods of detecting potholes which requires manual labour. The mobile acts as the entire environment to run the application which is capable of capturing images. A good camera will ensure accurate detection. The GUI of the system is required for the end user interaction to access the functionalities of the application. A server is required which hosts the pothole detection model. It responds to requests of images sent by the mobile application. The mobile should be mounted on the dashboard or windscreen by the municipal authorities while scanning the roads for potholes to get the optimum result.

3.3 Proposed System

3.3.1 Approach

The proposed system uses a convolution network framework YOLOv3 for detecting the objects in an image. The network is trained to detect potholes by training the neural network. This custom trained object detection model is used for detecting potholes by deploying the model on a server supported by the micro-web service framework Flask. Flask allows to generate Application Programming interfaces (APIs) which can be used for requesting services and fetching responses. The API allows to request for

detecting potholes in an image sent. The corresponding detection and confidence are sent back as response for the request by the server. The mobile application is used by the Municipal Authorities and Citizen. The application for the citizens will be able to capture image from camera as well as upload image from gallery. It works on the assumption that the citizen has Global Positioning System (GPS) working on the device. The image is sent to the server to identify if the image constitutes of potholes or not. The server then sends back the response. The citizen reports the pothole if the model identifies the pothole. Corresponding data linked with the pothole is prepared and registered to the database. The data includes the coordinates of GPS of the device, count of potholes, and the image captured. These findings are directly plotted on to the map. The application for the municipal deals with the ability of scanning the roads with the application. The device when mounted on the windscreen of a car, can identify potholes on the roads. The car needs to be moving in order to capture images in an interval of few metres and send the image to server. The response is sent containing the detection of potholes. The data is prepared accordingly which is sent to database. The end result helps to form coloured lines called polylines on the roads of map which denotes the severity of the situation and count of potholes.

3.3.2 Pothole Detection Model

The Pothole Detection Model uses 75 CNN layers for object detection. The object detection framework used is YOLOv3. The training of the custom object detection model requires the annotation or labelling of potholes. These annotated images are used for training the model. The trained model is ready to be used for detecting potholes.

3.3.3 Flask Server for deploying Detection Model

Deploying the object detection model directly on the mobile device is possible but the device requires very high specifications, which becomes a constraint to feasibility and availability of such high end mobile devices. So, the best round about solution was to deploy the object detection model on to a server using Flask Web Framework. Flask allows to create APIs through the application. Theses APIs are used directly on the mobile application to request and fetch data.

3.3.4 Mobile Application

The mobile application consists of all the functionalities which is required for the citizens and municipal authorities. It is developed and designed with Flutter to support cross-platforms. The application is supported by Firebase as back-end used for storing data relevant to the users and pothole Data.

3.3.4.1 User Authentication for Municipal and Citizen

Authenticating the user is required to check the role of the user. There are 2 roles laid out for this application, namely Municipal and Citizen. The municipal authority can sign-in to access a different set of features when compared to the citizen. Firebase Authentication is used for storing the users' data. Also, another collection is stored which consists of the role of the user.

The municipal authentication is simple and consists of Sign Up, Sign In, and Forgot Password. The Create Account feature lets the user to create a new account with a valid email id, name and password. Once registered, the user can login with credentials.

The citizen authentication consists of Sign Up, Sign In, Sign In with Google, Sign In Anonymously and Forgot Password. The variations in Sign In helps the citizens to authenticate in an easier fashion rather than going through the cumbersome process of filling up the name, email id and password. The anonymous login helps the citizen to act faster in some cases where reporting the pothole is the need of the hour.

3.3.4.2 Citizen Features

The citizen plays a crucial part in using the application. The potholes reported by a vigilant citizen help in plotting markers on map. These markers when clicked shows the sighting of the pothole. The citizen also gets rewards for finding potholes. The rewards are stored in the form of virtual credits in the application. The many features of the citizen are:

1. Capture Image from Camera

The citizen is able to capture an image from the camera of the mobile device. This image will be sent to the server to identify potholes.

2. Capture Image from Gallery

The citizen is able to import an image from the gallery of the mobile device. This image will be sent to the server to identify potholes.

3. View Maps

The citizen uses this feature to view the markers laid out by fellow citizens as well as the municipal authorities. Polylines on roads also visualized on the map which are colored lines signifying the severity of the condition of the road. The colour codes range from Yellow, Orange, Red indicating low, medium and high severity.

3.3.4.3 Municipal Features

1. Capture Image from Camera

The municipal authority is able to capture an image from the camera of the mobile device. This image will be sent to the server to identify potholes.

2. Scan the Roads

This feature is accessible only to the Municipal Authorities. It allows the Municipal Authorities to represent the severity of road conditions threatened by potholes. The manual inspection of roads can be eliminated with the help of the feature. The process followed by the municipal requires the need of a mobile holder mount to mount the device to the dashboard or the windscreen. The car starts from a point and the application is started with the feature turned on. The application captures an image in every 30 metres to the server to detect for potholes. If the server detects potholes, then application prepares the data accordingly and stores it in the database. This feature helps in showing polylines on maps. Polylines are colored lines which can be drawn on roads. Polylines are useful to show the severity of condition of the road. It denotes the urgency to maintain and repair the potholes on roads.

3. View Maps

The functionality of the municipal for view maps is similar to that of the citizen. The municipal authority is able to view the markers and polylines on the road which are colored roads based on the severity of the road conditions. The municipal also get the additional feature of closing down the reports of open potholes registered by both the citizens as well as the authorities.

Chapter 4

Design Of the System

4.1 Requirement Engineering

4.1.1 Requirement Elicitation

Table 4.1: Requirement Elicitation

Sr.No	Type	Short Description
I1	Informational	The system displays two logins (Citizen and Municipal Authority)
F1	Functional	Citizen should be able to login into the system
I2	Informational	The system displays select image from gallery
F2	Functional	Citizen picks pothole image from the device
I3	Informational	The system displays capture image
F3	Functional	Citizen should able to capture image of the pothole
I4	Informational	The system displays show map
F4	Functional	Citizen can visualize the map having polylines and markers on the road
F5	Functional	Municipal Authority should be able to login into the system
I5	Informational	The system displays select image from gallery
F6	Functional	Municipal Authority picks pothole image from gallery
I6	Informational	The system displays scan the road
F7	Functional	Municipal Authority should be able to get image frame containing pothole
I6	Informational	The system displays show map
F7	Functional	Municipal Authority can visualize the map having polylines and markers on the road

4.1.2 Software lifecycle model

The Software life cycle model used was Agile model. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release. Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the municipal authority as well as the citizens. Reasons to use Agile model:-

Individuals and Interactions: In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.

Working software: Demo working software is considered the best means of communication with the municipal authority to understand their requirements, instead of just depending on documentation.

Municipal Authority collaboration: As the requirements cannot be gathered completely in the beginning of the project due to various factors,

continuous interaction with municipal authority is very important to get proper product requirements.

Responding to change: Agile Development is focused on quick responses to change and continuous development.

4.1.3 Requirement Analysis

4.1.3.1 Data Flow Diagram

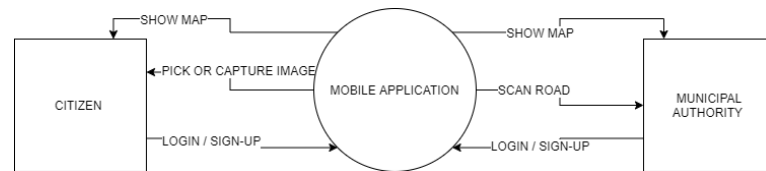


Figure 4.1: Data Flow Diagram LEVEL: 0

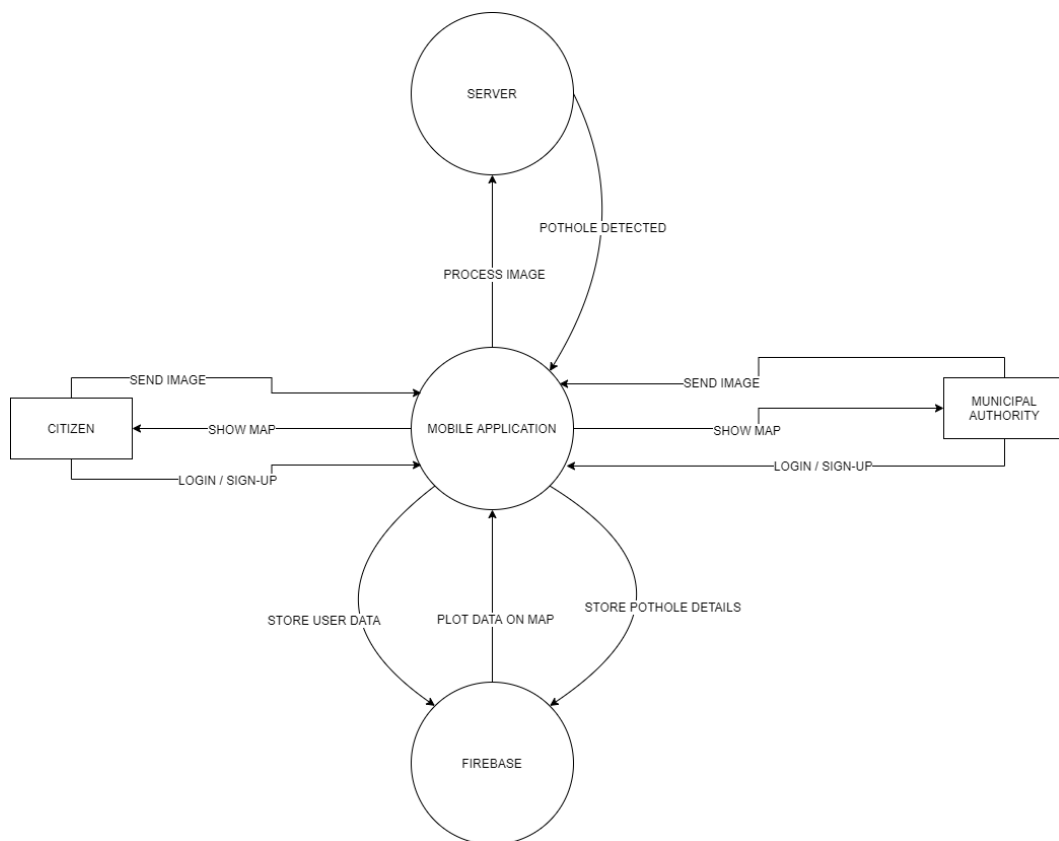


Figure 4.2: Data Flow Diagram LEVEL: 1

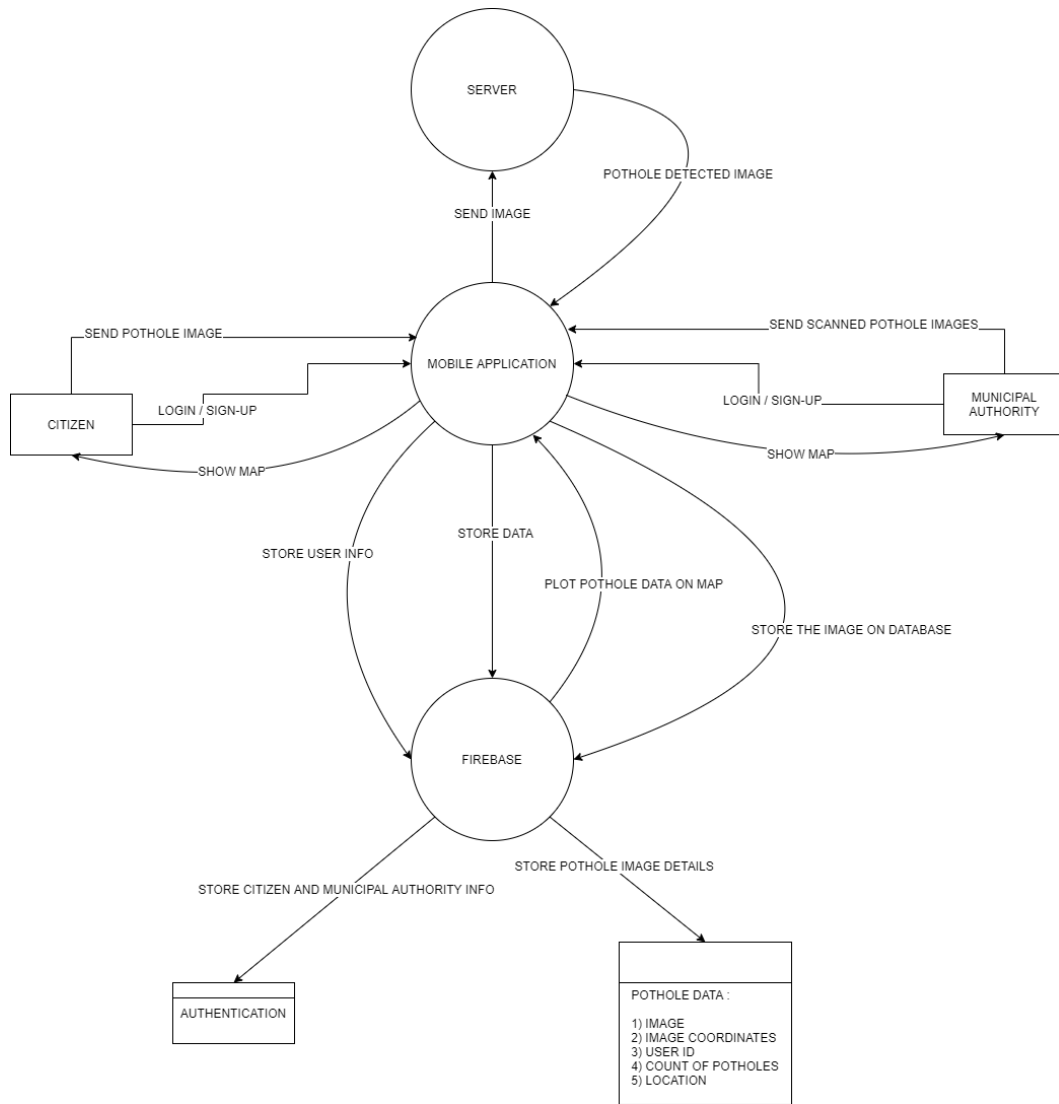


Figure 4.3: Data Flow Diagram LEVEL: 2

4.1.3.2 Use Case Diagram

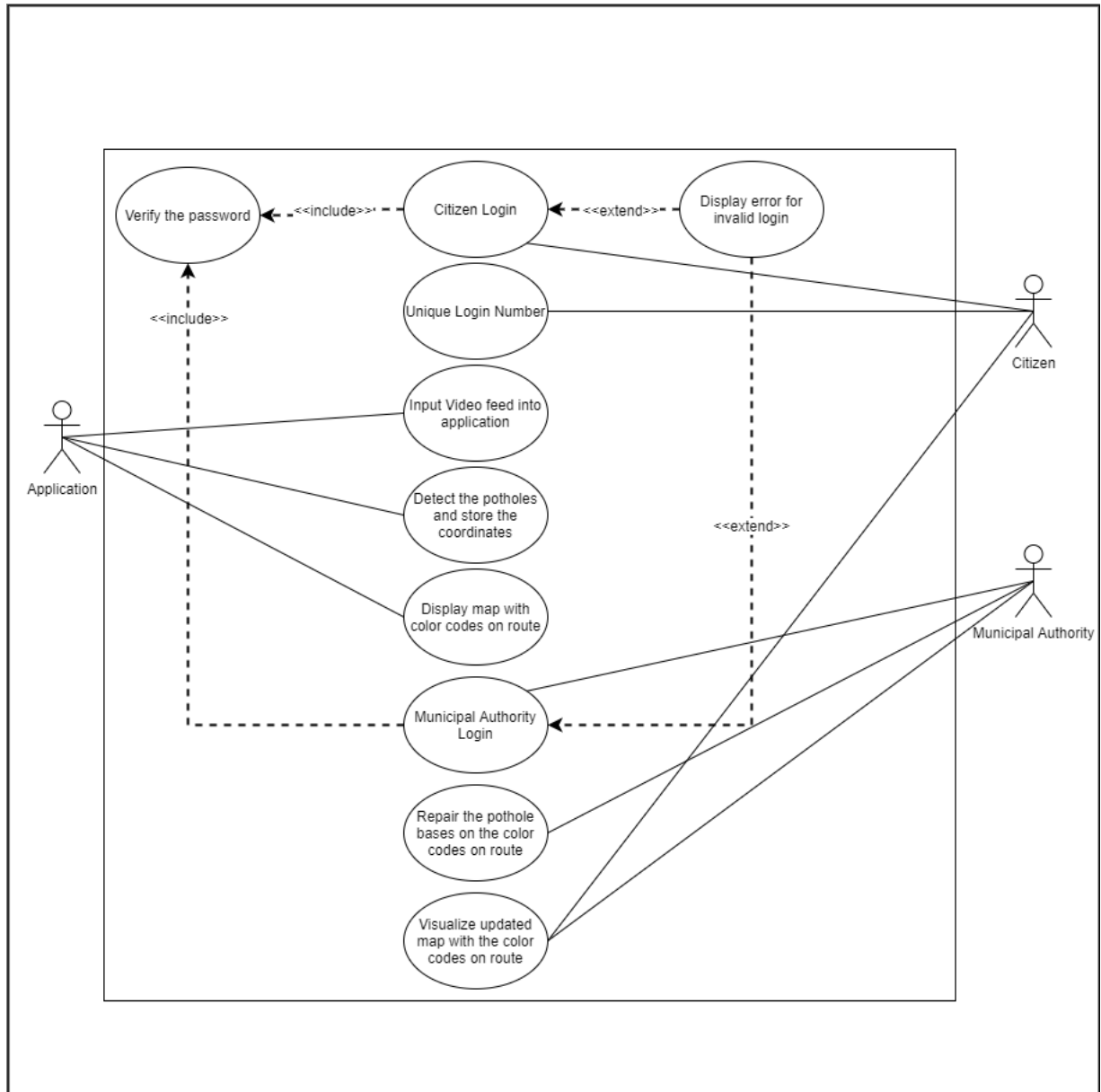


Figure 4.4: Use Case Diagram Of The System

4.1.3.3 Cost Analysis

1. Citizen

The citizen requires to have a mobile device that meets the minimum Software and Hardware requirements. The application will be free of cost. Hence, the total cost incurred by the citizen depends on the type of mobile device owned. An active internet connectivity is required to send the image to the server and Firebase.

2. Municipal Authority

- (a) Mobile Device: In order to capture images of good quality, it is required to have a mobile device which meets the minimum requirements. Such a device can cost between INR 15,000 to INR 25,000.
- (b) Municipal Vehicle: The vehicle used for scanning the roads can be any car, semi-truck or a truck. The cost incurred will include the fuel charges needed to make the inspection possible.
- (c) Mobile Holder and Mount for Car : The mobile needs to be carried by holder to capture the images. Such a holder will cost around INR 500 TO INR 1000.

4.1.3.4 Software Requirements

- Mobile Software Requirements:
 - 1. Mobile Operating System : Android 7.0.0 and above or iOS 6 and above
 - 2. Mobile Application Front-End : Flutter
 - 3. Mobile Application Back-End : Firebase
- Server Requirements:
 - 1. Server Operating System: Windows 7/8/10 or Ubuntu 18.04 and above
 - 2. Server Web-Framework : Flask
 - 3. Server Back-End : Tensorflow-GPU, OpenCV
- Training Requirements:
 - 1. Google Chrome v70.0.3538.77 and above

4.1.3.5 Hardware Requirements

- Training System Requirements:
 - 1. Intel i3 Processor or above
 - 2. \geq 6GB RAM
 - 3. \geq 5 GB HDD Memory

4. Internet Connectivity

- Mobile Device Requirements:

1. Camera with 16 MP and above
2. ≥ 4 GB RAM
3. Free space ≥ 100 MB Memory
4. Internet Connectivity
5. GPS

- Server Requirements:

1. NVIDIA GPU 1000 series and above with ≥ 4 GB memory
2. ≥ 4 GB RAM
3. Free space ≥ 1024 MB Memory
4. Internet Connectivity

4.2 System architecture

4.2.1 Block Diagram

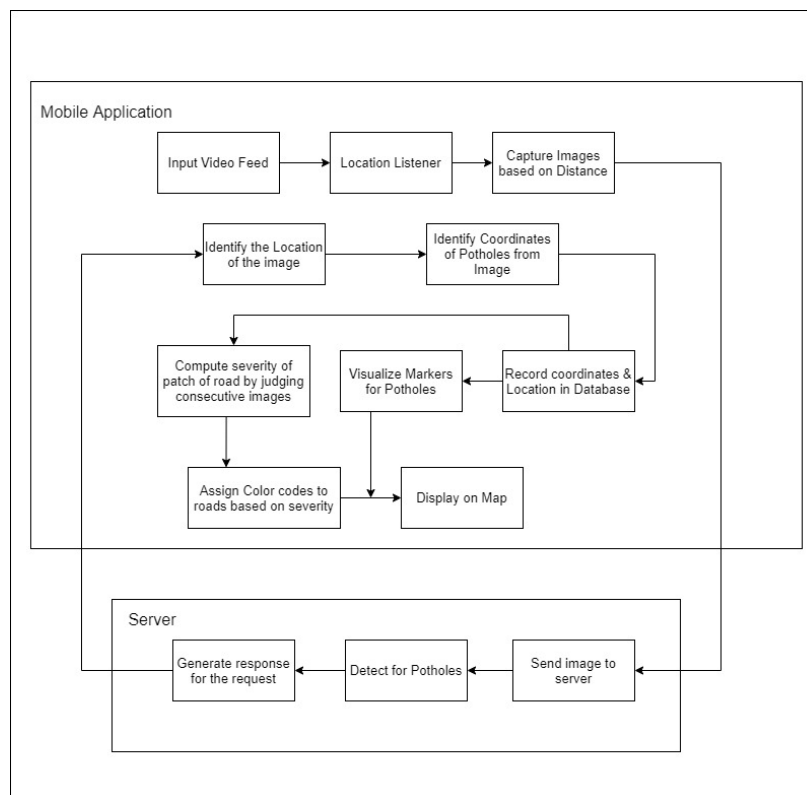


Figure 4.5: Block Diagram

4.2.2 UI/UX diagram

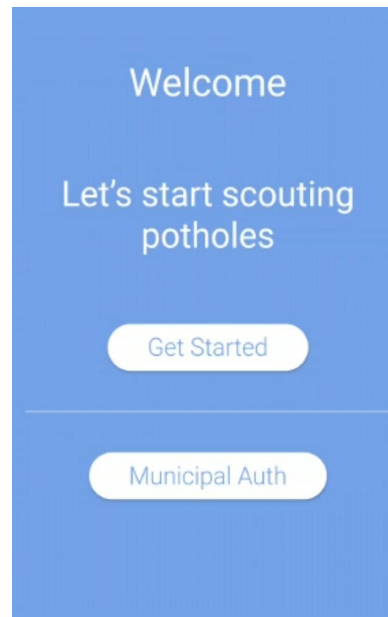


Figure 4.6: 1. Front Page Of Application

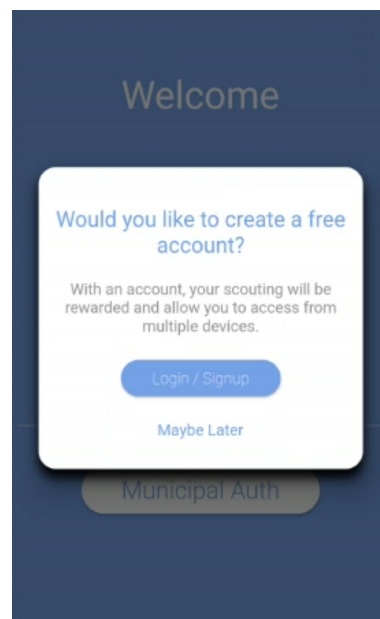
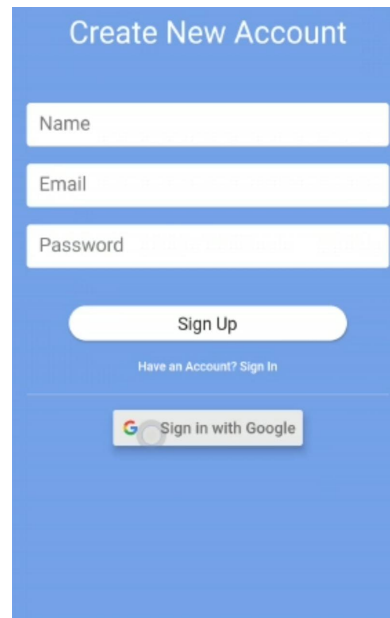
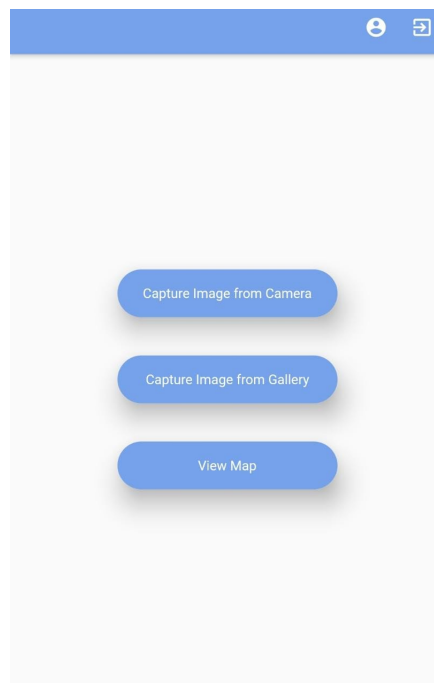


Figure 4.7: 2. Create A Free Account



A mobile app interface for creating a new account. The background is a solid blue color. At the top, the text "Create New Account" is displayed in white. Below this, there are three white input fields with blue borders, labeled "Name", "Email", and "Password". Under the "Password" field is a white "Sign Up" button with blue text. Below the button is a link that says "Have an Account? Sign In" in small white text. At the bottom, there is a white button with the Google logo and the text "Sign in with Google".

Figure 4.8: 3. Login/SignUp



A mobile app interface for the Municipal Authority/Citizens Page. The background is a light gray color. At the top, there is a blue header bar with a white user profile icon and a white share icon. Below the header, there are three blue buttons with white text, stacked vertically: "Capture Image from Camera", "Capture Image from Gallery", and "View Map".

Figure 4.9: 4. Municipal Authority/Citizens Page

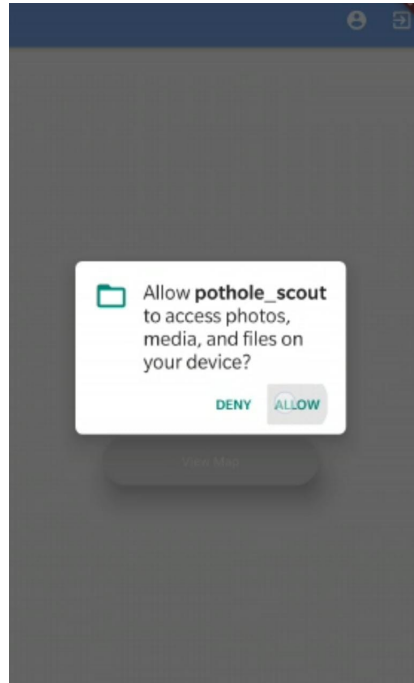


Figure 4.10: 5. Enable Location Before Capturing Image

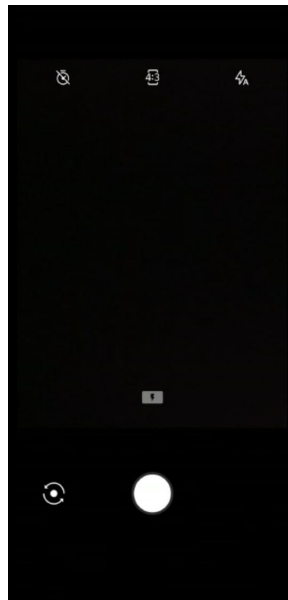


Figure 4.11: 6. Capture Image

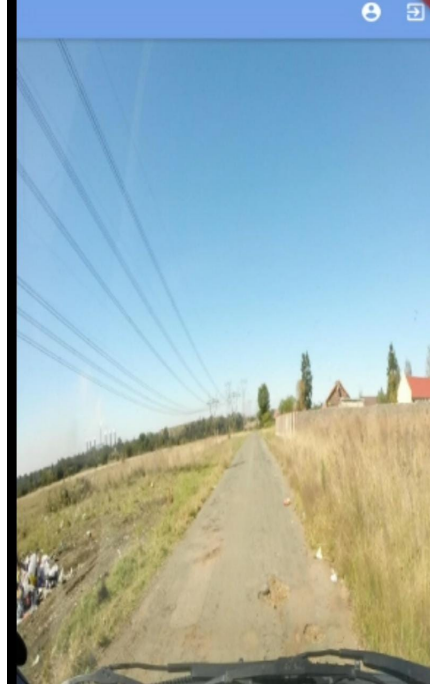


Figure 4.12: 7. Select Image From Gallery

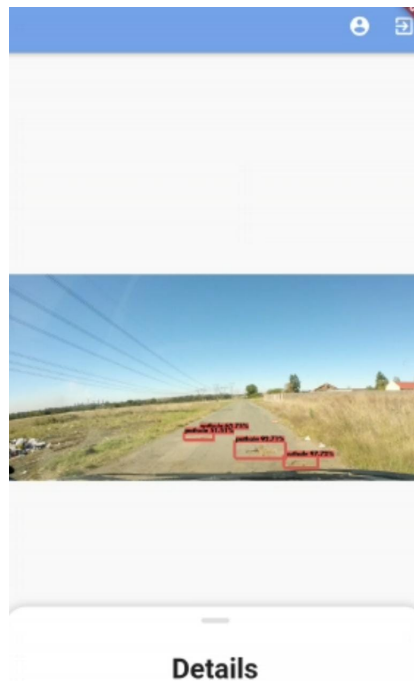


Figure 4.13: 8. Selected Image Results From Server

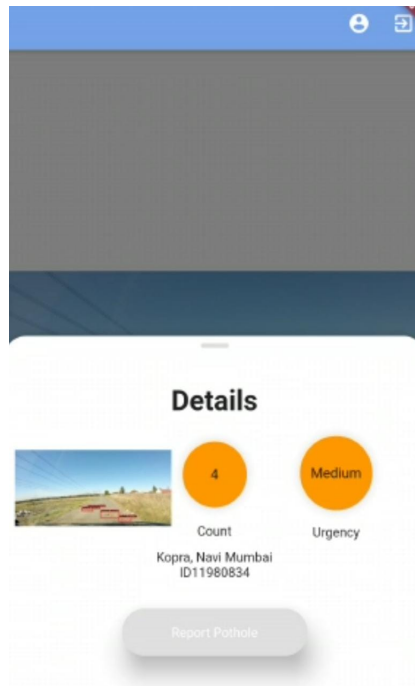


Figure 4.14: 9. Selected Image Details

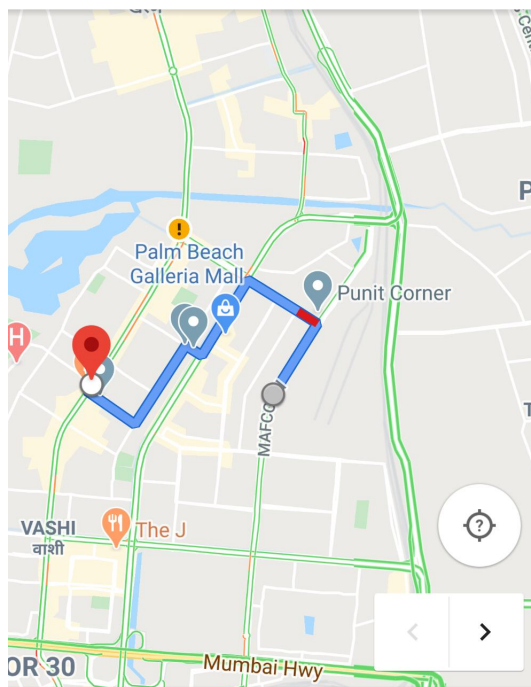


Figure 4.15: 10. View Map (Vashi Area)

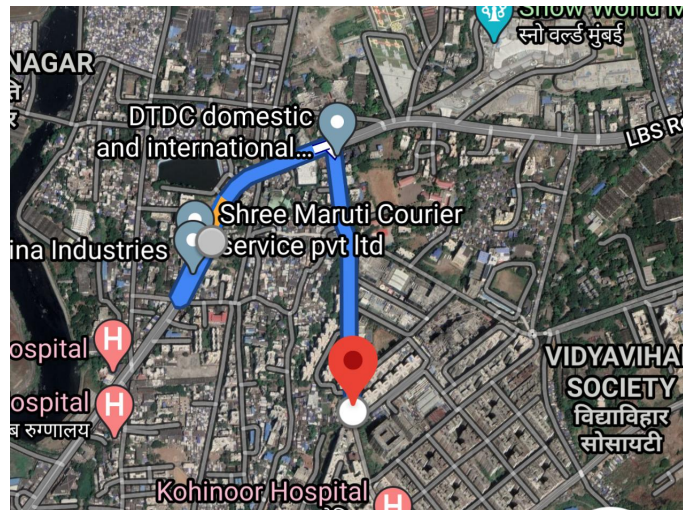


Figure 4.16: 11. View Map (Vidyavihar Area)

Chapter 5

Result and Discussion

5.1 Observations

Using the AlexyAB darknet[12] which is an implementation of yolo version 3 object detection algorithm, we created a pothole detector. Training required various pre-processing techniques on the dataset as well as tweaking the repository towards training the model. Due to the lack of dedicated GPU, Google Colab was where training was done. It took us about 2 to 3 days to create the detector without a dedicated good enough GPU but we estimate that this process can be finished in even lesser time given a dedicated GPU with good compute capability. Before using a GPU, CUDA should be installed and darknet should be installed with the GPU and CUDNN options set to 1. Online tutorials are available on how to do this.

5.1.1 Collection of Data

Dataset collection is the first step, we need a dataset that has enough images to get a good detector. Furthermore, the dataset should have good variety for the classes that we want to build the detector for. Variety means images in which the classes are having different sizes, orientations, colors, etc. The more variety and size of the dataset, the more robust our detector is. There should also be some negative images which do not contain the classes. For our process, we took the dataset from the COCO dataset along with some images scraped from the web. But this won't suffice towards detecting the potholes on Indian roads efficiently. Due to this reason we require images from roads from our locality itself. Thus the next step towards creating the dataset to train is to collect these images which was done using a mobile device and finding potholes around places like Kurla, Panvel and Kharghar. Once a pothole is identified, images were clicked from different angles. This will ensure the model is trained to identify potholes of different types sizes etc.

5.1.2 Annotation

Here are 2 categories in the dataset namely positive and negative. Positive denotes the fact that the road contains one of multiple potholes whereas negative will denote that the image doesn't contain any kind of potholes in it. But training a model in YOLO does not require partition of images in the dataset and hence a 70-30 ratio of images from the positive and negative images were taken from the files into the final dataset[13]. Object Detection using YOLOv3 and darknet on Dataset: With multiple images being captured, it becomes essential that the object detection model de-

detects potholes in the least amount of time. YOLOv3 (You Only Look Once ver.3) improves operation speed which meets real time requirements for detection. Darknet is a framework for YOLO implemented using C/CUDA. To make the object detection model mobile friendly, a server has been set up with the help of Flask. It hosts the object detection model and generates APIs. These APIs are directly used in the mobile device. It will make the model run efficiently in the mobile devices.

Table 5.1: Dataset information for the detector

Total no. of images used	1600
Training images	1120
Testing images	480
No. of classes	1

5.1.3 Training

The main part of building a detector is training. Training the neural network requires a lot of computations which are usually done using good NVIDIA GPUs. These are expensive and require additional packages or softwares installed for their use which require a proper environment. Instead of doing all this manually, we decided to use the GPUs that Google provides on their colab platform. The GPUs provided are powerful and all the necessary installations are easy to carry out. The downside is that the runtime is refreshed every 12 hours meaning that a lot of redundant work needs to be done. On top of that sometimes, colab causes the browser to crash and gets disconnected if the browser is inactive for more than 20-30 minutes. Anyhow, for lack of a better option, we used colab to train. We used the initial darknet53.conv.74 to start training using YOLO. The training statistics are provided in the following table.

Darknet gives us a graph that visualizes the training process and helps us give some idea about what stage the training is at. If we are satisfied, then we can stop the training or we can continue. It gives us the average loss which is the error (the less is better) and the mAP. Now, the AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1. mAP (mean average precision) is the average of AP. In some context, we compute the AP for each class and average them. But in some context, they mean the same thing. For example, under the COCO context, there is no difference between AP and mAP.

Once the model is trained, it is ready to be tested on real world images. Now the model is stored as .weights file during the training. and to use this convolutional neural network model easily in other places it is essential to convert the .weights file to a tensorflow object detection model. Once we convert it using in built functions provided by tensorflow platform. Once convert to the required tensorflow model it can easily be used in any api or device if required.

Table 5.2: Training statistics

Parameters	YOLO	tiny-YOLO
No. of iterations	12000	6000
Time taken (hours)	20	15
Average Loss	0.6	0.86
mAP(Mean Average Precision)	54%	48.4%

5.1.4 Mobile App Development

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Windows, Mac, Linux, Google Fuchsia and the web. Thus using a single code base we will be able to cater all the operating systems making it the best choice for us to develop this application. Developing the application requires a division and creation of two separate user base. That will be the Civic body who is responsible for scanning the roads and citizen who can click images of destructed roads.

For the civic body, the application concentrates towards creating an environment where the person on the field scanning has to do minimal amount of work. This requires the need for navigation, camera from the mobile device, and active internet connectivity to send and receive data from API's and database. Flutter constitute various libraries such as geolocator library, camera dart and the google maps API to help solve this problem. In,case of the application for the citizen to use, the same libraries can be applicable to get the geolocation of the citizen who has initiated a request to fix a road. Using the device camera the user and help to plot the same on the map for the user to check. With the help of the object detection model deployed throught the API locally we can detect with the image sent by the citizen can classify the severity of the road by analyzing the area of the pothole covered. Later the data is sent to the database which will then be used to plot it on the map itself.

5.1.5 Application Program Interface (API)

An application programming interface (API) is a computing interface to a software component or a system, that defines how other components or systems can use it. It defines the kinds calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. It can also provide extension mechanisms so that users can extend existing functionality in various ways and to varying degrees. An API can be entirely custom, specific to a component, or it can be designed based on an industry standard to ensure interoperability.

API's essential for building this application is part of the Google Maps API collection. Specifically, the roads API which has the feature of snapping any geolocation to the nearest road. This is helpful because of the jitters caused by GPS in small mobile devices. Also all conditions that this application test are on roads itself. This makes the Roads API most essential API in this project. The municipal personnel part of the application clicks images for evaluation based on the distance. Thus, the Distance Matrix API comes into play which calculates distance based on the two sets of positions given in the form of latitude and longitude. Once indicated with a certain set of distance, we proceed to click images from the device. For processing the images, either the device can itself be used to run the machine learning model. But accuracy is greatly affected with this method. Therefore, improvement in accuracy is essential to keep the data more reliable for the citizens and the responsible authorities. Creating an API using Python Flask and tensor flow allows us to use the model to its full potential without the need to create a mobile friendly version which will sacrifice the accuracy greatly. Also, the geocoding API from google helps to get the names of the roads for better searching functions if any citizens needs to find the conditions of roads based on their area.

5.1.6 Database (Firestore NoSQL)

Firebase is a Backend-as-a-Service — BaaS — that started as a YC11 startup and grew up into a next-generation app-development platform on Google Cloud Platform. Firebase frees developers to focus crafting fantastic user experiences. You don't need to manage servers. You don't need to write API's. Firebase is your server, your API and your data store, all written so generically that you can modify it to suit most needs. Yeah, you'll occasionally need to use other bits of the Google Cloud for your advanced applications. Firebase can't be everything to everybody. But it

gets pretty close. In case of this particular application Firestore with is a late beta phase of firebase is used and it is a Nosql type of database. All the different forms of data can be collected from the application mainly from citizens and municipal authorities. Using Firebase is important because it makes the integration with flutter very much simpler and also it makes the data have no particular structure. Firebase also help towards the authentication of different users in the system be it a citizen who will be rewarded for their help or even an authority worker scanning the road. Thus Firebase base acts as an all round backend solution towards creating this application.

5.2 Important Code Snippets

5.2.1 API that returns JSON with classes found in images

```
@app.route('/detections', methods=['POST'])
def get_detections():
    raw_images = []
    images = request.files.getlist("images")
    image_names = []
    for image in images:
        image_name = image.filename
        image_names.append(image_name)
        image.save(os.path.join(os.getcwd(),
                                image_name))
        img_raw = tf.image.decode_image(
            open(image_name, 'rb').read(), channels
            =3)
        raw_images.append(img_raw)

    num = 0

    # create list for final response
    response = []

    for j in range(len(raw_images)):
        # create list of responses for current image
        responses = []
        raw_img = raw_images[j]
        num+=1
```

```

img = tf.expand_dims(raw_img, 0)
img = transform_images(img, size)

t1 = time.time()
boxes, scores, classes, nums = yolo(img)
t2 = time.time()
print('time: {}'.format(t2 - t1))
for i in range(nums[0]):
    tfData = np.array(boxes[0][i])
    box = tfData.tolist()

    responses.append({
        "class": class_names[int(classes[0][i])],
        "confidence": float("{0:.2f}".format(
            np.array(scores[0][i])*100)),
        "box": box
    })

with counter.get_lock():
    counter.value += 1
    count = counter.value

img = cv2.cvtColor(raw_img.numpy(), cv2.
    COLOR_RGB2BGR)
img = draw_outputs(img, (boxes, scores,
    classes, nums), class_names)
cv2.imwrite('detectedImages/' + 'detection' +
    str(count) + '.jpg', img)
path = 'detection' + str(count) + '.jpg'
print(path)
print('output saved to: {}'.format('
    detectedImages/' + path))
count += 1
response.append({
    "image": image_names[j],
    "detections": responses

})

```

```

#remove temporary images
for name in image_names:
    os.remove(name)
try:
    return jsonify({"response":response}), 200
except FileNotFoundError:
    abort(404)

```

5.2.2 Function to add Marker objects into a List and used for generating Markers

```

void addMarkersOnMap() {
  for (int i = 0; i < marker.length; i++) {
    String id = "markerID" + i.toString();
    setState(() {
      allMarkers.add(
        Marker(
          markerId: MarkerId(id),
          draggable: false,
          onTap: () {
            print("Tapped");
          },
          position: marker[i],
          icon: pinLocationIcon,
        ),
      );
    });
  }
  print(allMarkers);
}

```

5.2.3 Distance Matrix API

```

import 'package:http/http.dart' as http;
import 'dart:convert';

```

```

class DistanceMatrixAPI{

```

```

  /* distance matrix API takes 2 sets of latitudes and
    longitudes and returns the distance between the two
    coordinates if the returned distance is in meters
    else it just returns a random number */

```

```

static Future<int> distanceMatrixAPI(lat1 , long1 ,
    lat2 , long2) async{
    print("\n\n");
    print("distance Matrix api called");
    int distance;
    var url = "https://maps.googleapis.com/maps/api/
        distancematrix/json?units=metric&origins="+lat1
        .toString()+"", "+long1.toString()+"&
        destinations="
            +lat2.toString()+"", "+long2.toString
            ()+"&key=$api_KEY";
    var response = await http.get(url);
    var jsonResponse = json.decode(response.body);
    String distanceInString = jsonResponse['rows
        '][0]['elements'][0]['distance']['text'];
    distance = jsonResponse['rows'][0]['elements
        '][0]['distance']['value'];
    print(lat1.toString()+"", "+long1.toString()+"", "+
        lat2.toString()+"", "+long2.toString());
    print(distance.toString()+" matrix distance");
    print(distanceInString);
    if (distanceInString.split(' ').contains('km')){
        print("contains km");
        return -1010;
    }
    else {
        return distance;
    }
}
}

```

5.2.4 Roads API

```

import 'package:http/http.dart' as http;
import 'package:geolocator/geolocator.dart';
import 'dart:convert';

class RoadsAPI{

    /* takes input as position(geolocator) and returns a

```



```
list containing latitude and longitude respectively
. */
static Future<List> roadAPI(Position position) async
{
    print("roadsAPI called");
    var lat = position.latitude.toString();
    var long = position.longitude.toString();
    var url = "https://roads.googleapis.com/v1/
        nearestRoads?points="+lat+','+long+'&key=
        $api_KEY';
    var response = await http.get(url);
    var jsonResponse = json.decode(response.body);
    var latApi = jsonResponse['snappedPoints'][0]['
        location']['latitude'];
    var longApi = jsonResponse['snappedPoints'][0]['
        location']['longitude'];
    return [latApi, longApi];
}
}
```

5.3 Testing

Table 5.3: Testing Report

Functionality	Test Cases Executed	Test Cases Passed	Remarks
Citizen Authentication	10	7	Pass
Citizen Google Authentication	10	7	Pass
Municipal Authentication	10	7	Pass
Anonymous Authentication	10	8	Pass
View Maps	15	12	Pass
Sending the image to Server	25	21	Pass
Fetching a response from the server	25	21	Pass
Opening Camera from the mobile	20	16	Pass
Opening Gallery from the Mobile	20	16	Pass
Displaying the pothole detected	25	20	Pass
Displaying Markers on the Map	30	23	Pass
Displaying polylines on maps	30	24	Pass
Logging Out for Citizen	10	8	Pass
Logging Out for Municipal Authority	10	8	Pass
Converting Citizen email to Gmail Authentication	10	10	Pass

Chapter 6

Conclusion & Future Scope

The municipal authorities labour manually for inspecting potholes and require a pair of eyes or more to visually detect them and condition the roads. The proposed idea automates the inspection with the help of a mobile application, vigil citizens and the municipal authorities to work together in bringing in notice the potholes and making it easier to fix them by the authorities. A combination of Computer Vision and Machine Learning to detect the potholes which supports the mobile application to capture images and send them to a server for identifying potholes is used. Firebase, which is a cloud storage solution, is used for storing the data inputs by both the citizens and authorities.

The end result of the hard work is shown on a map with the help of markers which denote the potholes and colored roads which depict the severity of the condition of the road. It is a user friendly system which has a flexible and easy to use interface. We understand that user of this system may not have technical expertise. Therefore the system is designed in such a way that the user is not burdened with the technical jargon and still be used with ease.

The future work can be extended by integrating our system to the users travelling by road. Car Manufacturers like Tesla have inbuilt cameras for their auto pilot system, which can be integrated with the pothole detection system developed by us. This can help in identifying potholes more easily which reduces the burden for the municipal bodies. A notification system can be developed to aid the drivers with the information of any particular roads having potholes.

References

- [1] H.-W. Wang, C.-H. Chen, D.-Y. Cheng, C.-H. Lin, and C.-C. Lo, “A real-time pothole detection approach for intelligent transportation system,” *Mathematical Problems in Engineering*, vol. 2015, 2015.
- [2] S.-K. Ryu, T. Kim, and Y.-R. Kim, “Image-based pothole detection system for its service and road management system,” *Mathematical Problems in Engineering*, vol. 2015, 2015.
- [3] Y. Jo and S. Ryu, “Pothole detection system using a black-box camera,” *Sensors*, vol. 15, no. 11, pp. 29316–29331, 2015.
- [4] V. Rishiwal and H. Khan, “Automatic pothole and speed breaker detection using android system,” in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1270–1273, IEEE, 2016.
- [5] K. Vigneshwar and B. H. Kumar, “Detection and counting of pothole using image processing techniques,” in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–4, IEEE, 2016.
- [6] R. Madli, S. Hebbar, P. Pattar, and V. Golla, “Automatic detection and notification of potholes and humps on roads to aid drivers,” *IEEE sensors journal*, vol. 15, no. 8, pp. 4313–4318, 2015.
- [7] M. Jiang, T. Hai, Z. Pan, H. Wang, Y. Jia, and C. Deng, “Multi-agent deep reinforcement learning for multi-object tracker,” *IEEE Access*, vol. 7, pp. 32400–32407, 2019.
- [8] P. Tumas and A. Serackis, “Automated image annotation based on yolov3,” in *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, pp. 1–3, IEEE, 2018.
- [9] “Caglar gulcehre.” <http://deeplearning.net/>.

- [10] A. M. Algorry, A. G. García, and A. G. Wofmann, “Real-time object detection and classification of small and similar figures in image processing,” in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 516–519, IEEE, 2017.
- [11] P. Lahudkar, S. Sawale, V. Deshmane, and K. Bharambe, “Nosql database-google’s firebase: A review,” *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 7, no. 3, 2018.
- [12] “Online.” <https://github.com/AlexeyAB/darknet>.
- [13] “Labelimg.” <https://awesomeopensource.com/project/tzutalin/labelImg>.

Acknowledgement

Success of a project like this involving high technical expertise, patience and massive support of guides, is possible when team members work together. We take this opportunity to express our gratitude to those who have been instrumental in the successful completion of this project. We would like to show our appreciation to **Mr. Amroz K. Siddiqui** for their tremendous support and help, without them this project would have reached nowhere. We would also like to thank our project coordinator **Mrs. Rakhi Kalantri** for providing us with regular inputs about documentation and project timeline. A big thanks to our HOD **Dr. Lata Ragha** for all the encouragement given to our team. We would also like to thank our principal, **Dr. S. M. Khot**, and our college, **Fr. C. Rodrigues Institute of Technology, Vashi**, for giving us the opportunity and the environment to learn and grow.

Project Group Members:

1. Joel Cherian, 101608

2. Joseph Blessingh, 101621

3. Harun Joe, 101660

Appendix A : Timeline Chart

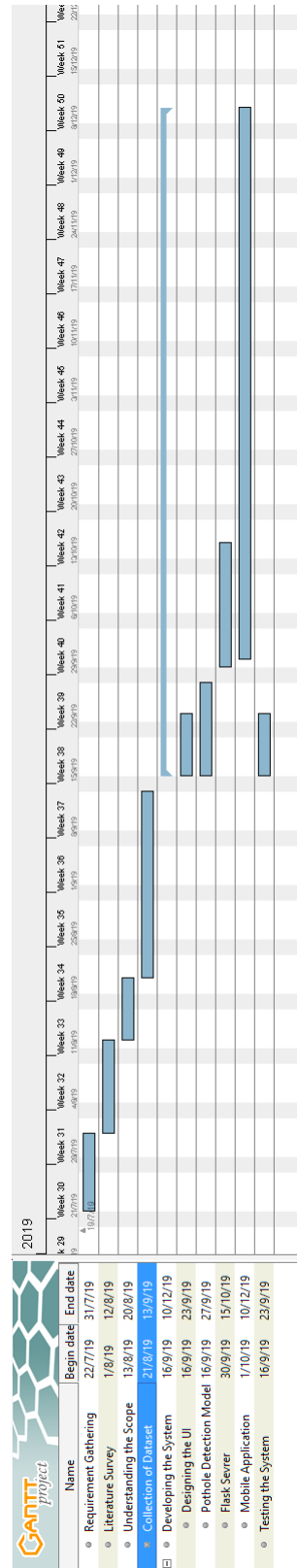


Figure 6.1: Timeline Chart For Project

Appendix B : Publication Details

An Intelligent Pothole Detection System using Deep Learning

International Research Journal of Engineering and Technology (IRJET),
Volume 7, Issue 2, February 2020

<https://irjet.net/archives/V7/i2/IRJET-V7I2334.pdf>