Chapter 1

Introduction

The introduction chapter of our graduation project presents an innovative mobile application designed to enhance road safety by alerting drivers to upcoming cracks on their route. In an era where road maintenance often lags behind the increasing traffic load, this application aims to mitigate the risk of accidents caused by road damage. Utilizing advanced mapping technology, the application provides drivers with timely notifications about cracks and potholes, allowing them to take precautionary measures. This chapter outlines the motivation behind the project, highlighting the growing concern over road conditions and their impact on vehicular safety.

1.1 Overview

The Advanced Road Analysis System is a groundbreaking initiative Application designed to develop next-generation infrastructure for precise road crack detection and accurate road sign interpretation.
The implementation of state-of-the-art technologies offers an immersive and secure user experience that surpasses the capabilities of conventional navigation systems.

One of the key features of this system is its ability to detect road surface cracks and alert users as they approach them, thereby preventing potential damage to vehicles and infrastructure. The system employs advanced algorithms to identify and relay information regarding nearby road signs, enhance situational awareness, and aid in avoiding road violations.
The Application also give the user the experience to report cracks that is not located on the map .This makes the Application up to date to the new cracks created in the roads
All these features and more are encapsulated in an android application handy for users for enhanced driving experience.

To sum up, an Advanced Road Analysis System is a remarkable example of the convergence of advanced technology and user-centric design. It addresses critical issues related to road safety and infrastructure preservation and sets a new standard for intelligent and comprehensive road analysis systems.

1.2 Problem statement

1-Road infrastructure cracks problems

Road infrastructure forms the backbone of a nation's transportation system, facilitating the movement of people and goods. However, these vital networks are constantly under siege by a multitude of challenges that can significantly impact their functionality and safety. These are examples that cause cracks of damages to road infrastructure :

1. Deterioration and Decay: Over time, even the most meticulously constructed roads succumb to wear and tear. Factors such as heavy traffic volume, extreme weather conditions, and inadequate maintenance lead to the degradation of road surfaces, manifesting as potholes, cracks, and unevenness.

3. Climate Change: The intensifying effects of climate change pose a new and significant threat to road infrastructure. Extreme weather events such as floods, heatwaves, and storms can damage roads, disrupt transportation networks, and necessitate costly repairs.

Road cracks are a pervasive issue that pose significant challenges to both infrastructure and safety. These cracks, which can arise from a variety of factors such as weather conditions, heavy traffic, and substandard construction materials, lead to the deterioration of road surfaces over time. The presence of cracks not only accelerates the wear and tear of vehicles but also increases the likelihood of accidents, as drivers may lose control when navigating uneven surfaces. Additionally, road cracks can exacerbate traffic congestion as drivers slow down to avoid damaged sections of the road, leading to longer commute times and reduced overall efficiency. In severe cases, these imperfections can evolve into larger potholes, further compromising the safety and usability of the road. Addressing the problem of road cracks requires timely maintenance

and repairs, as well as innovative solutions like real-time monitoring systems to prevent minor issues from becoming major hazards.

2-Road signs Awareness problem

Another problem to focus here is the problems related the ignorance of the importance of road signs either from the user or from the governments .

A World Without Traffic Signs Awareness: A Recipe for Chaos

Imagine a world devoid of traffic signs. Intersections would transform into scenes of anarchy, with drivers locked in a perpetual game of chicken, unsure of who has the right-of-way. Speed limits would become a forgotten concept, with reckless drivers posing a constant threat to pedestrians and other vehicles.

The consequences would be dire:

Increased Traffic Accidents: The absence of clear guidelines would lead to a surge in accidents, resulting in fatalities, injuries, and substantial property damage.

Gridlock and Inefficiency: Without designated lanes or traffic flow directives, congestion would skyrocket, significantly increasing travel times and hindering the movement of goods and people.
Erosion of Trust and Cooperation: The lack of a universal language on the road would breed distrust and hostility among road users, further exacerbating the chaotic environment.

1.3 Scope and objectives :

Our Application has yet identified specific limitations that shape its scope and objective.
This Application provides an new driving experience which you can drive safely far away and from road cracks yet there is no navigation and routing for user to use this Application as a map to reach specific destination route.

The Map crack points are updating internally using technologies and programming techniques not Automatically from the crack reporting feature yet the data gathered from this feature important to rely on .

It also detecting the crucial types of cracks that cause serious like traffic congestion of car damage not for all types of cracks as our scope and interest here is the user only this Application can not be used for organizations to repair any road damages.

The sign detection here is only for the alarming purpose its not working on damaged road signs or signs that are too hard to be interpreted .

Application objectives :

- Alarm system for the user to avoid cracks.
- Crack survey feature to report new cracks that is not on the map
- Save and secure account for the user holding his data.
- Notifications for the road signs located in the chosen route.

1.4 Report organization (Structure)

Chapter 1 :
We have introduced our project overview gives us functionalities of the application , the problems statements that we are working on and our project scope and a list of our main objectives .

Chapter 2 :
We will present the studies behind this project and all the analysis per-work we did to to publish this project.

Chapter 3 :
This chapter shows the proposed solution we pointed out in chapter one at the problem statements, map analysis and design , the application requirements both functional and non-functional , machine learning models and dataset analysis work and finally the application analysis and design diagrams.

Chapter 4 :
This chapters shows all the implementation details for the application, mapping and machine learning models, the experimental training results , model evaluations.

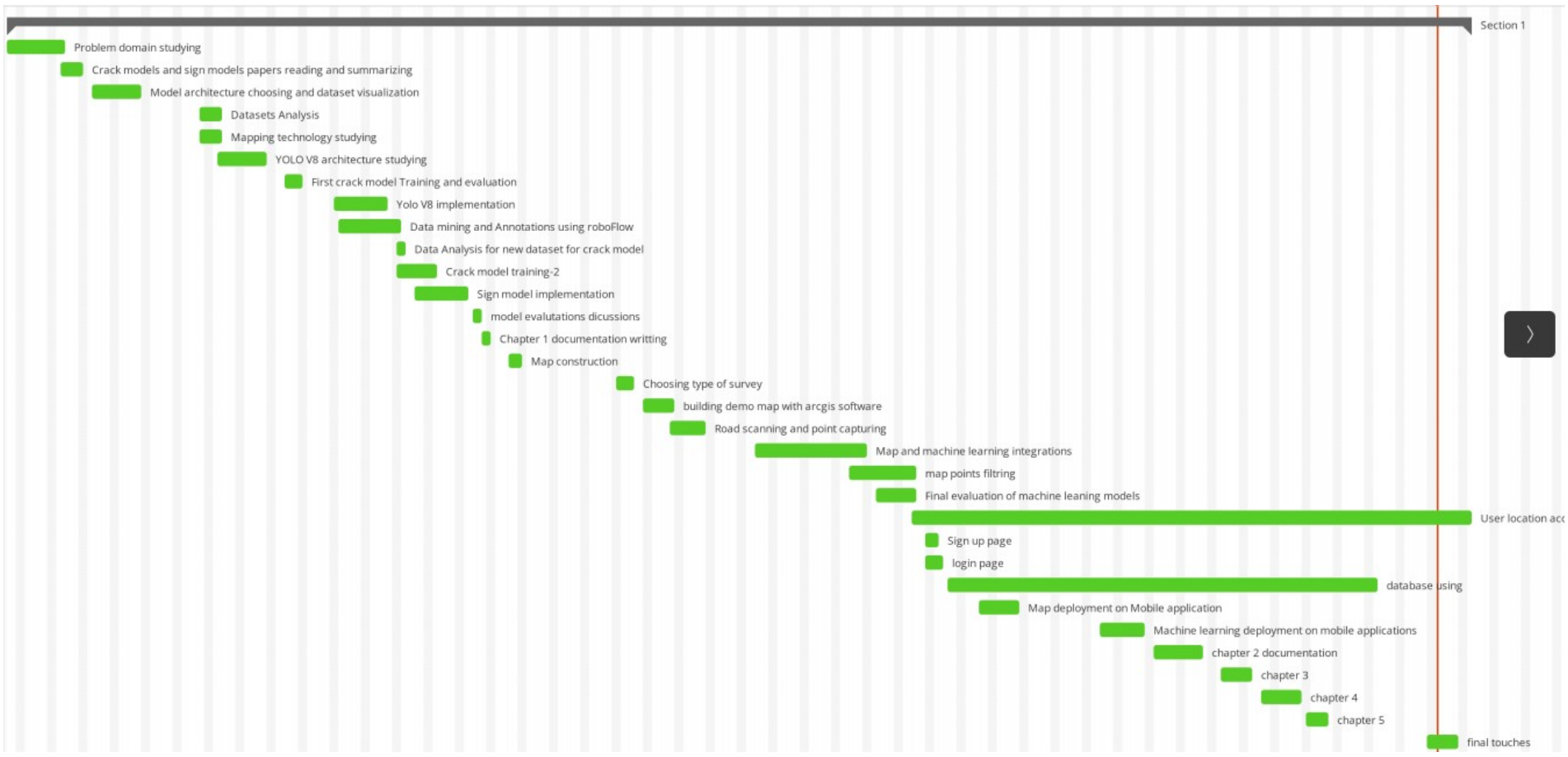
Chapter 5 :
This chapter shows the conclusion part , the summary of the documentation and our future work plan for his project.

1.5 The work methodology

Here we are trying to implement an Application that uses Deep learning models and mapping technologies to enable users to be able to take action before cracks occurs in their roads through an alarm.

1.6 Gantt chart

Shows the workflow of the project implementation



1	Problem domain studying	Unassigned	–	20/Jul	01/Aug	Approved	100%
2	Crack models and sign mod...	Unassigned	–	01/Aug	05/Aug	Approved	100%
3	Model architecture choosin...	Unassigned	–	08/Aug	18/Aug	Approved	100%
4	Datasets Analysis	Unassigned	–	01/Sep	05/Sep	Approved	100%
5	Mapping technology studying	Unassigned	–	01/Sep	05/Sep	Approved	100%
6	YOLO V8 architecture studyi...	Unassigned	–	05/Sep	15/Sep	Approved	100%
7	First crack model Training a...	Unassigned	–	20/Sep	23/Sep	Approved	100%
8	Yolo V8 implementation	Unassigned	–	01/Oct	12/Oct	Approved	100%
9	Data mining and Annotation...	Unassigned	–	02/Oct	15/Oct	Approved	100%
10	Data Analysis for new datas...	Unassigned	–	15/Oct	16/Oct	Approved	100%
11	Crack model training-2	Unassigned	–	15/Oct	23/Oct	Approved	100%
12	Sign model implementation	Unassigned	–	19/Oct	30/Oct	Approved	100%
13	model evalutations dicussio...	Unassigned	–	01/Nov	02/Nov	Approved	100%
14	Chapter 1 documentation w...	Unassigned	–	03/Nov	04/Nov	Approved	100%
15	Map construction	Unassigned	–	09/Nov	11/Nov	Approved	100%
16	Choosing type of survey	Unassigned	–	03/Dec	06/Dec	Approved	100%
17	building demo map with arc...	Unassigned	–	09/Dec	15/Dec	Approved	100%
18	Road scanning and point ca...	Unassigned	–	15/Dec	22/Dec	Approved	100%
19	Map and machine learning i...	Unassigned	–	03/Jan	27/Jan	Approved	100%
20	map points filtring	Unassigned	–	24/Jan	07/Feb	Approved	100%
21	Final evaluation of machine ...	Unassigned	–	30/Jan	07/Feb	Approved	100%
22	User location access	Unassigned	–	07/Feb	10/Jun	Approved	100%
23	Sign up page	Unassigned	–	10/Feb	12/Feb	Approved	100%
24	login page	Unassigned	–	10/Feb	13/Feb	Approved	100%
25	database using	Unassigned	–	15/Feb	20/May	Approved	100%
26	Map deployment on Mobile ...	Unassigned	–	22/Feb	01/Mar	Approved	100%
27	Machine learning deployme...	Unassigned	–	20/Mar	29/Mar	Approved	100%
28	chapter 2 documentation	Unassigned	–	01/Apr	11/Apr	Approved	100%
29	chapter 3	Unassigned	–	16/Apr	22/Apr	Approved	100%
30	chapter 4	Unassigned	–	25/Apr	03/May	Approved	100%
31	chapter 5	Unassigned	–	05/May	09/May	Approved	100%
32	final touches	Unassigned	–	01/Jun	07/Jun	Approved	100%

Chapter 2

Related work

In this chapter we will dig deep more in the background study about our project and similar apps in addition to how this application would benefit its users.
We also will discuss the technologies we used to come up with this project.

2.1 background

Computer vision contribution in solving cracks problems

This project is trying to alarm people about road infrastructure defectives with the help of deep learning models .So we made a research about how Ai and computer vision contribute in this field.

The ever-growing demands placed on road infrastructure necessitate innovative solutions to ensure safety, efficiency, and sustainability. In this arena, computer vision (CV) emerges as a powerful tool, offering a plethora of applications that contribute significantly to solving road infrastructure problems. This article explores the burgeoning field of CV in road infrastructure management, showcasing its potential through specific examples and highlighting its impact on various aspects of the transportation network.

1. Automated Road Inspection: Traditional methods of road inspection rely on manual visual assessments, which can be time-consuming, subjective, and prone to human error. CV algorithms, however, can be trained to analyze images and videos captured from mobile mapping systems or dedicated inspection vehicles. These algorithms can automatically detect and classify road surface defects such as cracks, potholes, and unevenness with high accuracy . This automation empowers transportation authorities to conduct frequent, objective inspections, leading to the early detection of problems and facilitating timely repairs, ultimately extending the lifespan of road infrastructure.

3. Infrastructure Asset Management: Road infrastructure encompasses a vast network of bridges, tunnels, and signage. CV offers a non-destructive and efficient method for monitoring the health of these assets. By analyzing images or video footage of bridges and tunnels, CV algorithms can detect cracks and other structural defects, enabling proactive maintenance and preventing potential catastrophic failures

Our main goal in this project is using the computer vision techniques to analyses road infrastructure and helping in solving road infrastructure problems through recognizing road cracks and road signs benefits both drivers to aware them of the coming cracks and authorities for road problems.

Computer science contribution in Traffic signs problems

Traffic signs are the cornerstone of a safe and efficient transportation system. However, maintaining a vast network of signage presents significant challenges. Traditional methods, often reliant on manual inspections, can be time-consuming, expensive, and susceptible to human error. This paper explores the burgeoning field of computer vision (CV) and its potential to revolutionize how we address traffic sign issues.

Computer Vision: A Powerful New Approach

Computer vision offers a transformative approach to tackling traffic sign issues. By leveraging advanced algorithms and machine learning, CV systems can:

Automate Sign Detection and Recognition: CV algorithms can analyze video footage or digital images to automatically locate traffic signs, classify their type (stop sign, speed limit, etc.), and even extract the information they convey.

Our goal for this part is to be able to notify the user of the upcoming road signs to increase the awareness of this critical and important problem as it affects safety.

2.2 Literature Survey

Our reference academic paper for the Crack detection and recognition model

Automated Road Crack Detection Using Deep Convolutional Neural Networks

The paper proposed an automated pavement distress analysis system based on yolo v2.System is trained on 7240 images acquired from mobile cameras measured by F1 score 87% without including class types and F1 score 79% with including class types.

Proposed methodology : The model target crack detection and classification , model is a cnn based on yolo v2 , NVIDA GTX 1080TI GPU used , Training for yolo v2 takes 18.2 hours.

Yolo v2: You Only Look Once (YOLO) has carved a niche in the realm of object detection with its emphasis on real-time processing speeds. This article delves into YOLOv2, a refined iteration that builds upon the strengths of the original YOLO architecture. We explore the core concepts of YOLOv2, its advantages for object detection tasks, and the advancements it introduced over its predecessor.

Results: For improvement YOLO trained on 40k iteration , learning rate was 0.01 , a box captured over 50% IOU with ground truth box is determined successful match true +ve otherwise false false +ve.

There is miss-classification as some images are not annotated, it could not predict in case of overlapping. It can detect Classes D40 and D20 very well compared to others, while D10 and D11 where the most challenging as the images for them in dataset is very limited

Several Models worked on this problem as yolo v3 , SSD and faster RCNN the got accuracy 5 % less compared it YOLO V2 Removing third conLayed in YOLO V2 architecture increased accuracy by 3 %Varying in batch size and learning rate gives no change Model accuracy increased by 4 % per every 5l iterations yet after 60 k accuracy was insignificant.

Our reference academic paper for the sign detection and recognition model

Improved YOLOv5 network for real-time multi-scale traffic sign detection

This paper uses TT100K dataset , the model used is yolo v5 withreplacing the feature extractor pyramid with AF-FPN.Traffic sign used by CNN and SSD but has high computations and largenumber of parameters so it will not fit in mobile device memory ,so hereThey used yolo v5 as it has low computation and fast recognition speed.

Main contributions: Multiscale target recognition , auto augmentation by Auto Augment and improved yolo v5 to be deployed on mobile

Experimentation and analysis:TT100K has 42.5% small objects , 50.1% medium objects and 7.4% large objects .The input dimension for model 608 *608 , Training images 9156 and test images 1121 images , initial learning rate was 0.01 , batch 32 and 500 epochs.

2.3 Analysis work

Given the survey we made earlier in the previous chapter, we found out that there are many papers written on our problems (crack detection and road sign detection) with varying in the accuracy percentage.

We made use of every paper we read as it seems that object detection task achieved promising results than segmentation , and according to models YOLO is the superior of all models that worked on the two problems.

Yet , There are more enhancements to do , as YOLO v2 model or YOLO v5 model used in our reference papers requires high computational power and need two much time for training so we can replace it with one that much more better as YOLO v8.

Our proposed methodology

We implemented our models using YOLO v8 model. As this is the new stat of art of object detection.

YOLOv8, developed by Ultralytics, builds upon the success of prior YOLO versions by introducing advancements in object detection, segmentation, and other computer vision tasks. It boasts state-of-the-art performance through a streamlined design that facilitates adaptation across various hardware platforms. One key distinction of YOLOv8 is its versatility. In contrast to previous YOLO versions primarily focused on detection, YOLOv8 offers a comprehensive suite of capabilities including detection, segmentation, pose estimation, tracking, and classification.

This allows researchers to leverage a single model for a wider range of computer vision applications within their academic pursuits. Additionally, YOLOv8 maintains the focus on speed and accuracy that characterized prior YOLO models, making it well-suited for real-time applications in academic settings.

CNN-based Object Detectors are primarily applicable for recommendation systems. YOLO (You Only Look Once) models are used for Object detection with high performance.
YOLO divides an image into a grid system, and each grid detects objects within itself. They can be used for real-time object detection based on the data streams. They require very few computational resources.
YOLOv8 is the latest version of the YOLO algorithm, which outperforms previous versions by introducing various modifications such as spatial attention, feature fusion, and context aggregation modules.

Multiple features are to be focused on in YOLOv8. Here are some key features of YOLOv8:

Improved Accuracy: YOLOv8 improves object detection accuracy compared to its predecessors by incorporating new techniques and optimizations.

Enhanced Speed: YOLOv8 achieves faster inference speeds than other object detection models while maintaining high accuracy.

Multiple Backbones: YOLOv8 supports various backbones, such as EfficientNet, ResNet, and CSPDarknet, giving users the flexibility to choose the best model for their specific use case.

Adaptive Training: YOLOv8 uses adaptive training to optimize the learning rate and balance the loss function during training, leading to better model performance.

Advanced-Data Augmentation: YOLOv8 employs advanced data augmentation techniques such as MixUp and CutMix to improve the robustness and generalization of the model.

Customizable Architecture: YOLOv8's architecture is highly customizable, allowing users to easily modify the model's structure and parameters to suit their needs.

Pre-Trained Models: YOLOv8 provides pre-trained models for easy use and transfer learning on various datasets.

Our proposed solution for mapping (Arcgis Mapping technology)

A crucial part of our application is that we worked on a real-world problem that provides people safe driving by detecting cracks and Traffic signs in roads using computer vision and making our detection on real road infrastructure using ArcGIS mapping technology.

ArcGIS Pro is a powerful, modern geographic information system (GIS) application developed by Esri. It is designed to provide GIS capabilities to help users visualize, analyze, and share data through maps and spatial analysis. ArcGIS Pro is particularly well-suited for tasks such as mapping, visualization, editing, analysis, and sharing of geographic information. Here are some of the key features and capabilities that make ArcGIS Pro useful for tasks like marking cracks and signs on maps, especially after detection.

1. Advanced Mapping and Visualization: ArcGIS Pro allows users to create and manage a wide range of 2D and 3D map types. This is crucial for accurately depicting geographical features, such as cracks or signs, on different scales and perspectives.

2. Spatial Analysis and Data Management: The software offers robust tools for spatial analysis, enabling users to understand and interpret data related to spatial patterns, relationships, and trends. For example, after detecting cracks or signs, you can analyze their spatial distribution and potentially correlate them with other geographic data.

3. Editing and Input Features: ArcGIS Pro provides sophisticated editing tools that allow for precise input and modification of data. Users can easily add points, lines, and polygons to represent real-world objects like cracks or signs on a map. These features can be manually added or imported from detection systems.

4. Integration with Sensors and Real-Time Data: ArcGIS Pro can integrate with various data sources, including real-time data feeds and sensors. If cracks and signs are detected via sensors or during field inspections, this data can be automatically incorporated into the GIS system, allowing for real-time updates and notifications.

5. Sharing and Collaboration: Maps and projects created in ArcGIS Pro can be easily shared with others through ArcGIS Online or ArcGIS Enterprise. This facilitates collaboration among multiple users or departments, who might need access

to the latest information regarding infrastructure issues like cracks and signs.

6. Automation and Scripting: With support for Python scripting and the ArcPy module, repetitive tasks and workflows, such as updating the locations of detected cracks and signs, can be automated. This is particularly useful for maintaining up-to-date GIS layers and responding quickly to new data.

7. High-Quality Output: The software supports the creation of high-quality map outputs and layouts for reports, presentations, or online sharing. This ensures that the information about detected cracks and signs is communicated clearly and effectively.

The power of that model has to be encapsulated with Application to make it handy for everyone to use it , instead of being only in the experimental and academic papers , so with the help of mapping technology this enables us integrate both to get high usability .

Chapter 3

Proposed solution

In this chapter we will discuss the our proposed solution to implement this project .we will start by introducing the main components of the project, Application functional and non-functional requirements and last thing but not least the Analysis and Design descriptions and diagrams.

3.1 Solution Methodology

The proposed methodology to accomplish this project was a simple road map as follows :

Goal : Our goal for this project is to implement an application that could be reliable , secure and user friendly at the same time.

As we are implementing a mobile application our goal is to make it easy for user to understand it , with high performance and accuracy in alarming cracks for him .

In addition to useful experience for both user and us to collect as more cracks as we can to always up date our work.

Project steps:

1. We studies road infrastructures condition in Egypt to define our problem correctly and was useful in choosing good dataset and defining our models goals.
2. We worked in parallel on both the Map implementation and training our models.
3. Scanning roads to get data for prediction and recording their point locations on map.
4. When both models and Map were ready we integrated both together to benefit from the model prediction and map representation of cracks and signs points.
5. Application implementation and deployment of Deep learning models and Map.

Justifications : This workflow steps followed as each step is an input for the next as we could not implement crack and sign detection models without exploring the environment where they will use , also locating points on map require the detection of the model.

Tools used : *Models are implemented using YOLO V8 architecture , map is implemented using two GIS and mapping technologies which are ARCGIS and QGIS and the application is implemented using Kotlin language and firebase.*

3.2 Functional / Non-functional Requirements

System non-Functional requirements :

1-Performance:

- **Responsiveness:** Receiving machine learning detection results should appear for user in mean time. This ensures a smooth user experience without lags or delays.
- **Efficiency:** The app should utilize device resources efficiently to minimize battery drain and overheating. This is especially important for features like GPS navigation and continuous machine learning detection.

2-Accuracy:

- **Machine Learning Detection:** The machine learning model should provide accurate detections with minimal false positives or negatives. This is crucial for features relying on real-time object identification.

3-Availability:

- **Machine Learning Model Updates:** The app should have a mechanism to download and update the machine learning model periodically to maintain accuracy and incorporate new detection capabilities.

4-Usability:

- **Intuitive Interface:** The map, navigation controls, and machine learning detection features should be easy to understand and navigate for users of all technical backgrounds.

5-Security:

- **User Data Privacy:** The app should collect and store user data (e.g., location history) securely and adhere to relevant data privacy regulations.
- **Model Security:** The machine learning model should be protected against adversarial attacks that could manipulate its detection results.

System Functional requirements :

1-Warning alarm for the nearest cracks : The system should provide the user with a warning system before cracks appear with sufficient distance so the driver can avoid it.

2-Warning alarm for the nearest speed limit or stop sign : The system should provide the user with a notification with the type of the speed limit sign or if its stop sign before ahead to prepare himself for the speed changing.

3-crack and sign survey : The system should provide a service to provide the user to capture cracks and signs images and the system will detect its verification and then will takes its location and needed information to save this points for further map points allocations.

3.3 Analysis and design

1-Deep learning models :

Crack Model dataset Analysis and visualization

1-Business analysis and understanding

Our goal in the project is to implement a crack detection model that has specific requirements related to the major problems in Egypt Streets locally and similar countries globally. So, Our work here is related to specific types of cracks, scope, and conditions to match our needs to Monitor Egypt's road infrastructure. Our crack detection model has these specific requirements to work properly:

1-Crack types :

- Longitudinal crack [D00]
- Traverse crack [D10]
- Alligator crack [D20]
- Pothole [D40]
- Some other types but with a very small samples

**Our main goal is to detect the D40 and D20 cracks properly as they are the most commonly located cracks in Egypt streets and also common the most to cause problems as they could damage vehicles and cause car accidents .*

2-Model scope :

The model is likely to be trained on local roads , state highways and national highways covering metropolitan as well as non-metropolitan cracks .

3-Model goal :

Model goal is to detect pothole and alligator with high accuracy compared to the other cracks types to help in monitoring road cracks to decease the cause of car accidents.

The dataset reference used :

RDD2018: A multi-national image dataset for automatic Road Damage Detection

This is the dataset that was used in the reference paper we worked on as a base for our model (*Automated Road Crack Detection Using Deep Convolutional Neural Networks*).

The dataset consists of 9053 damage images containing 15,435 damages annotated, with bounding boxes of 8 types of road cracks.Images are extracted from an image set cracked by capturing pictures of large-scale roads using vehicle-mounted smartphones.

The image here contains a wide variety of weather conditions and illuminations.Using a smartphone was the best option as a camera mounted to the car is illegal in many countries.

The image capturing rate was one image per second with a speed of 40 km/hr, and the image was captured in Japanese streets.

The annotation method was Pascal VOC format, Image resolution is 600*600 pixels.

Results: D01 and D44 have high recall and precision and D11 and D40 have low related to the number of samples intraining examples.

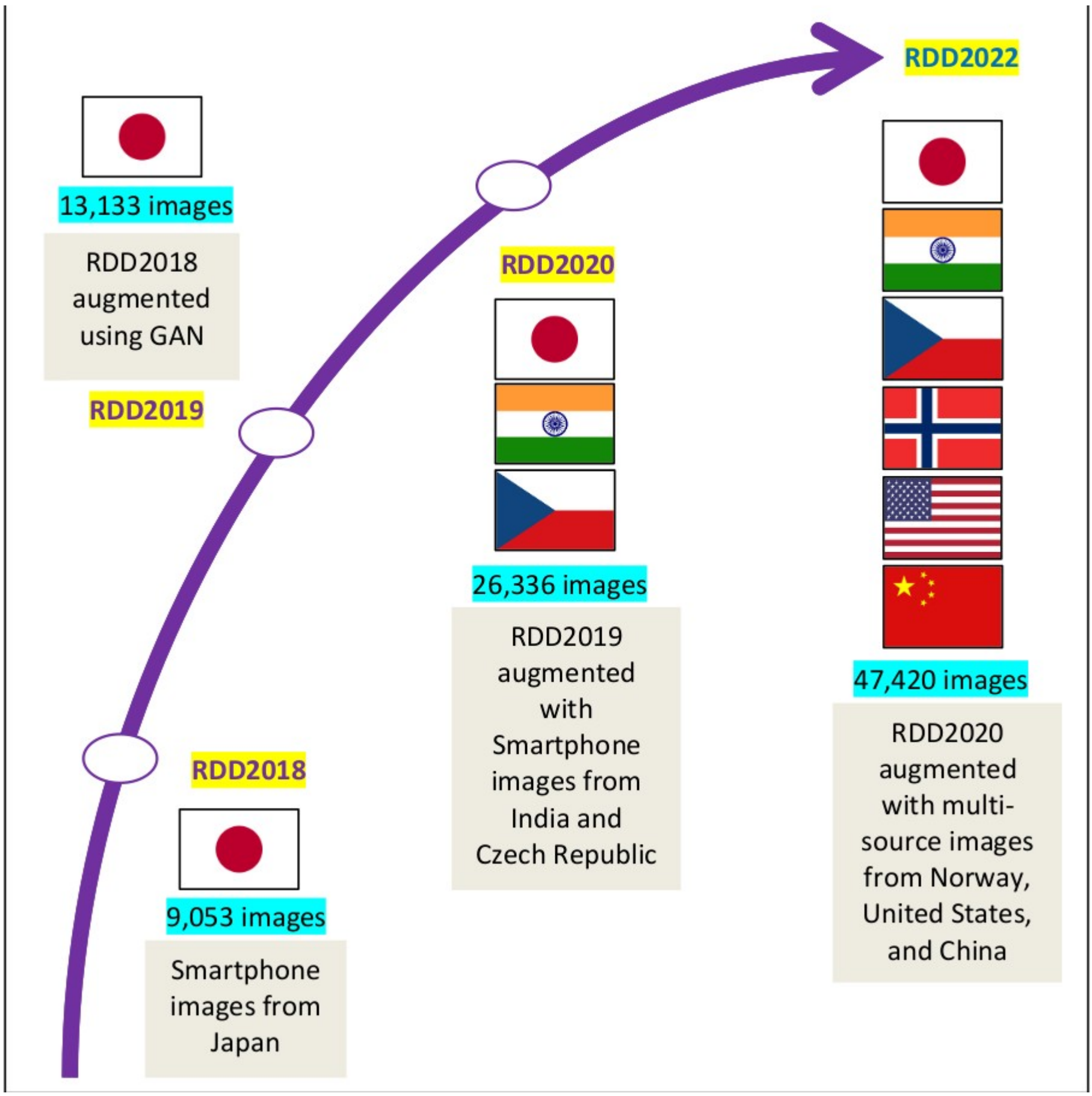
These were the results that came from analyzing and reviewing the paper on this dataset For these reasons this dataset was not suitable for our main project goal.

The Actual Dataset used :

Our problems with the previous dataset was that our main focus crack type D40 was very little in the training sample and this is a crucial part that we can not skip.

So we have searched for alternative but also have the benefits with the previous dataset and we have found

RDD2022: A multi-national image dataset for automatic Road Damage Detection



This dataset is an updated version of RDD2018 that matches our needs , We chose RDD2022 as it has the largest D40 samples.

Dataset contains 47,420 road images from 6 countries , images annotated with more than 55,00 instances of road images.

Dataset has focus on the main 4 types of cracks we focus on [Longitudinal crack-Traverse crack - Alligator - Pothole].

The dataset images are augmented which make it more suitable for training,adding countries rather than japan in previous dataset improves the performance (achieved generalization) .

The Dataset has 2 folders : 1-Trained annotated 2-Test not annotated

Each folder in the dataset related to a specific country , Japan has the highest train dataset and is annotated by pascal VOC format.

We choose the dataset folders related to Japan and India as they has the largest number of images and also the highest D40 samples .

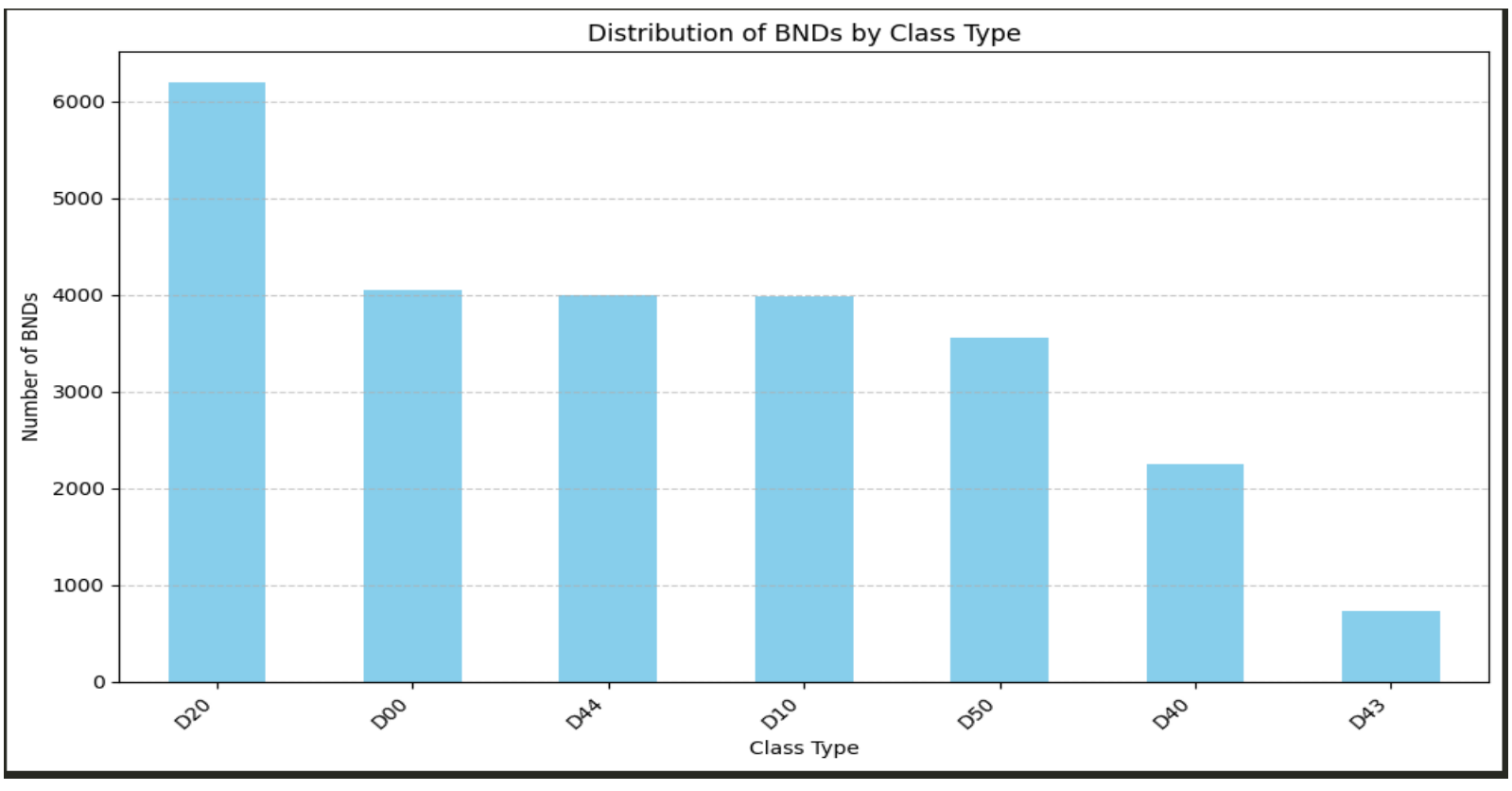
We applied Exploratory Data Analysis methods to understand and analyze these both files .

1) Exploratory Data Analysis For the Japan cracks dataset :

This dataset is annotated as pascal VOC format [XML Annotations] , image format is only JPG with resolution size 600*600.

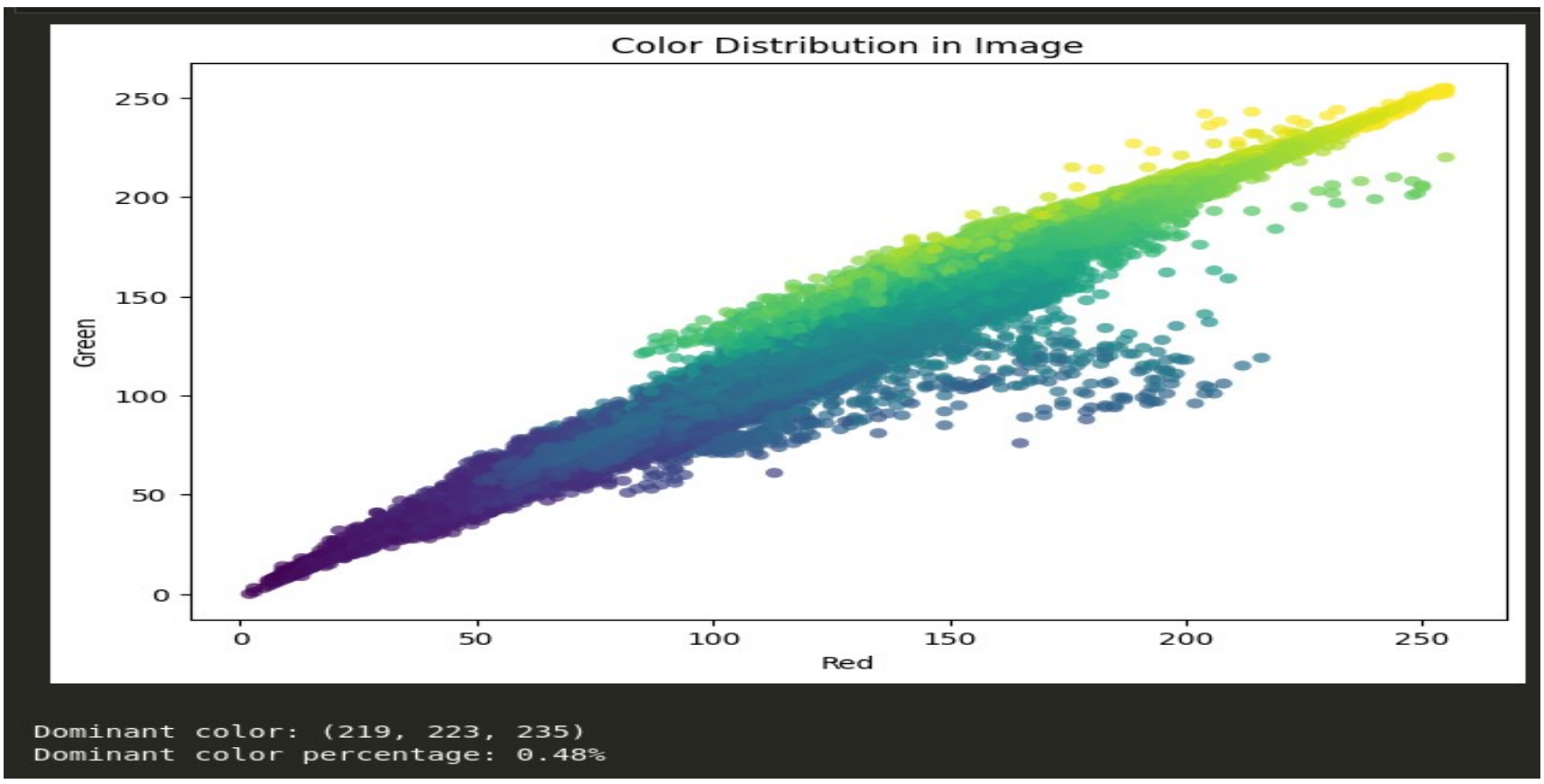
The Dataset structure :

- Train Folder :
  - Has 10506 images
  - Matches 10506 XML annotated file
  - Number of bounding boxes = 24754
  - Number of classes = 6
  - Image that has no cracks = 794

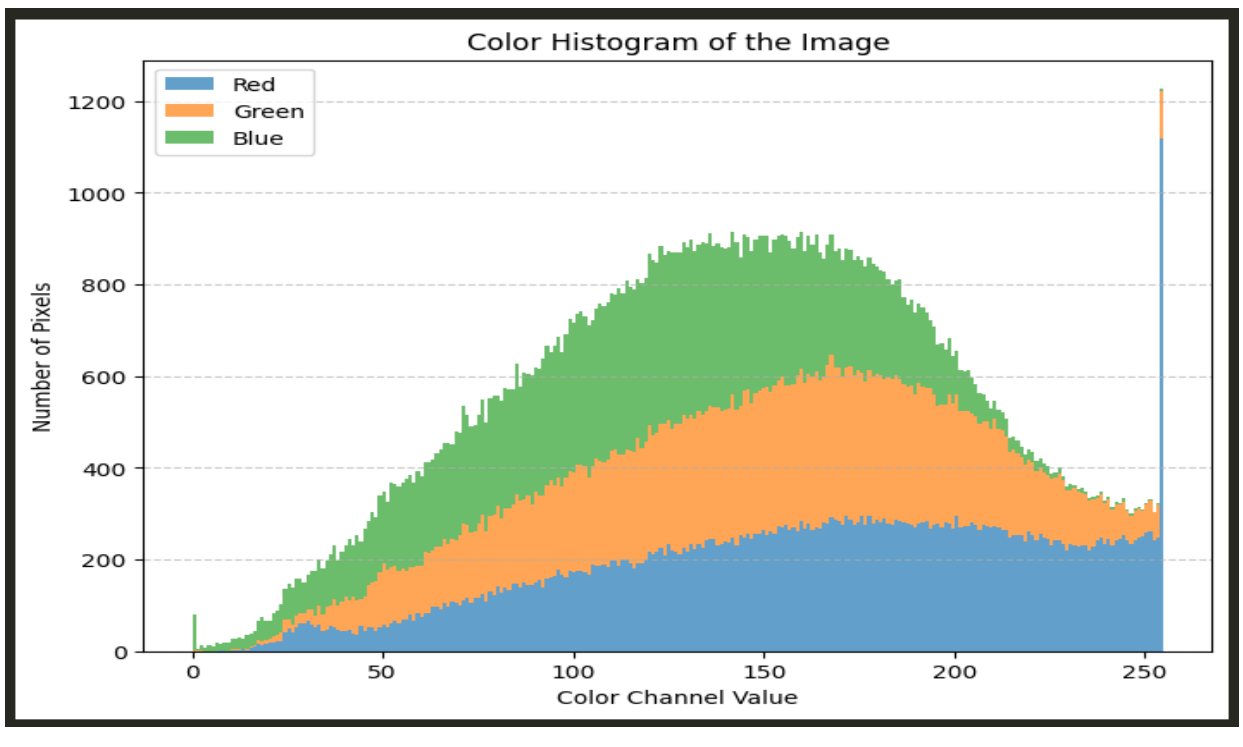


Other analysis types such as colour analysis are applied as Colour features can be used in conjunction with other image features like shape and texture to improve object recognition algorithms.

Color Distribution and dominant color (here is Grey of course).



Color histogram shows the range of colors present (from dark to bright values) and how spread out they are.
A narrow range with peaks concentrated at specific values indicates limited color variation, while a wider distribution suggests a more diverse color palette.

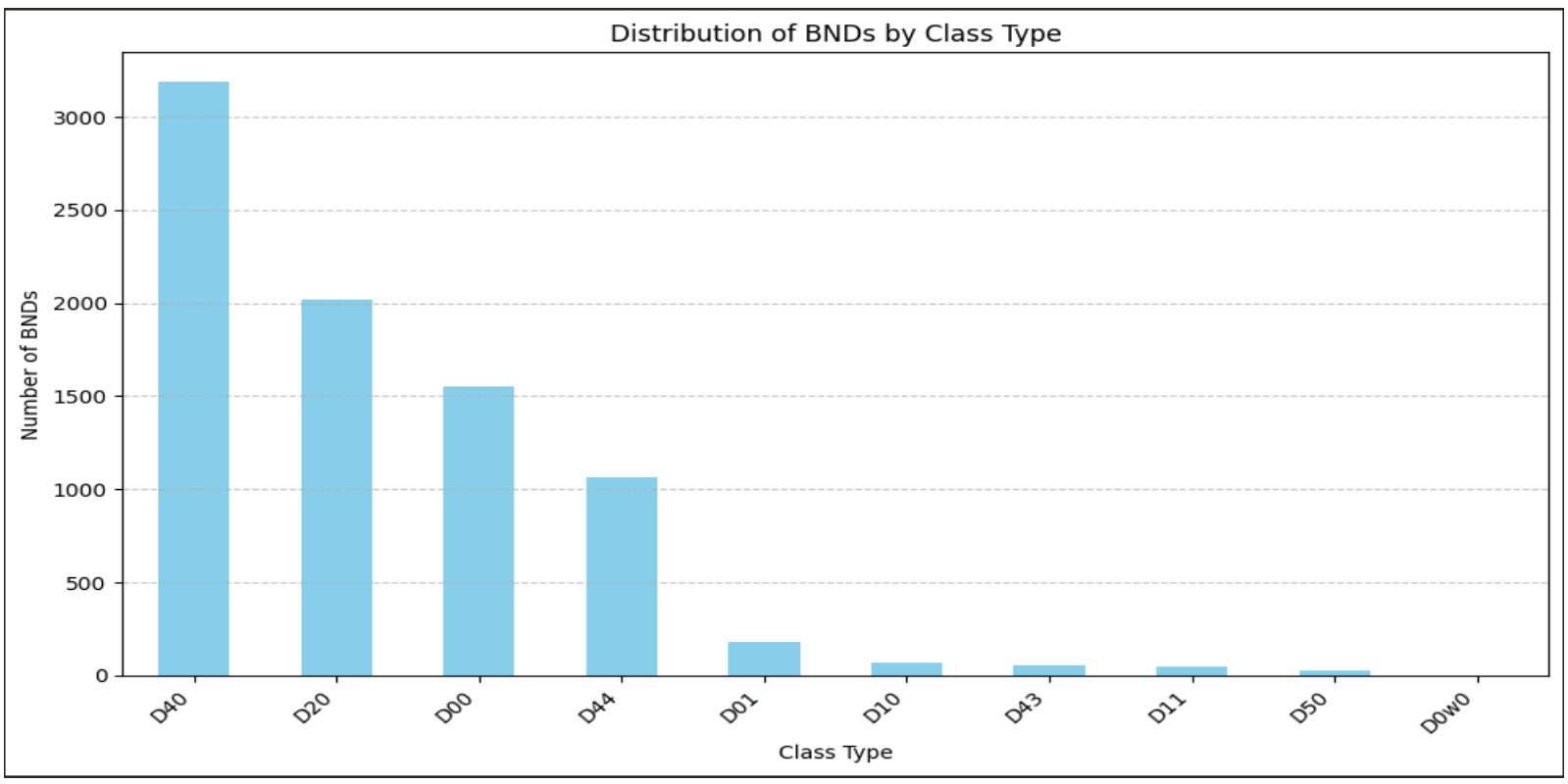


2) Exploratory Data Analysis For the India cracks dataset :

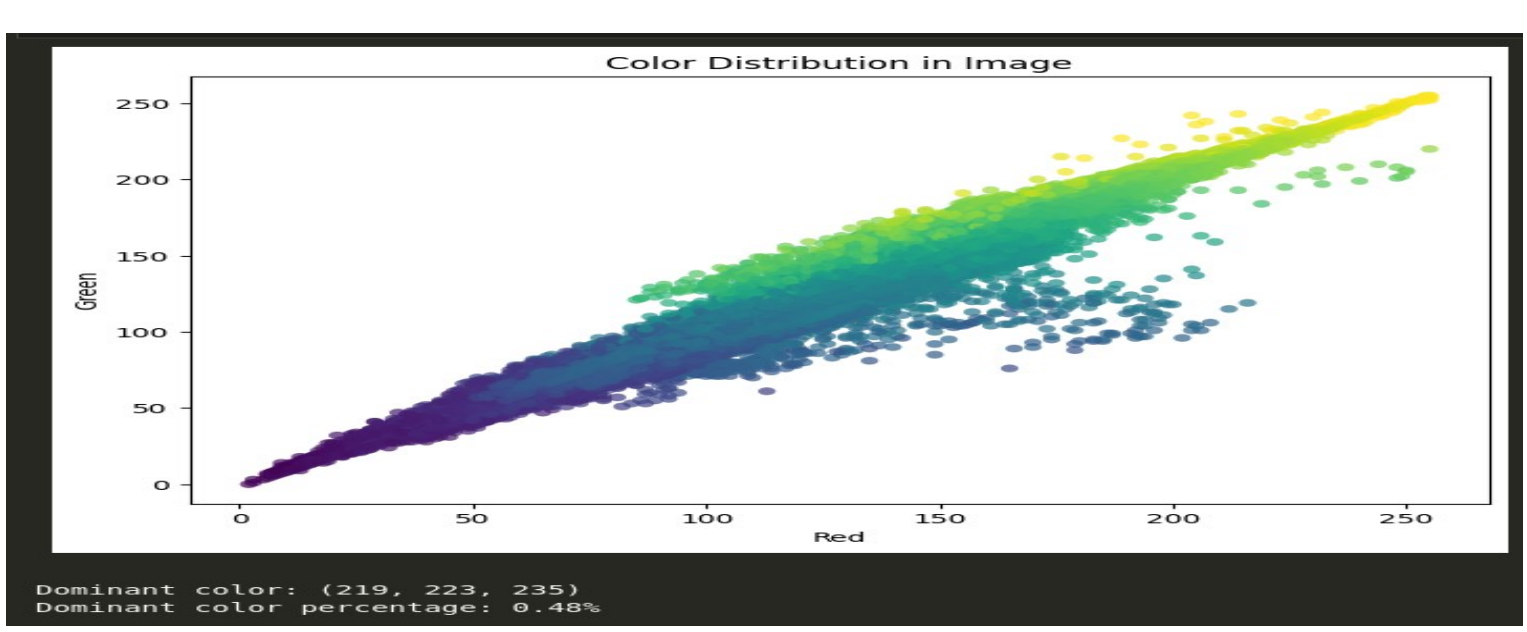
This dataset is also annotated by pascal VOC format with xml annotation files ,the resolution size 720*720 and all images are JPG format.

The dataset structure :

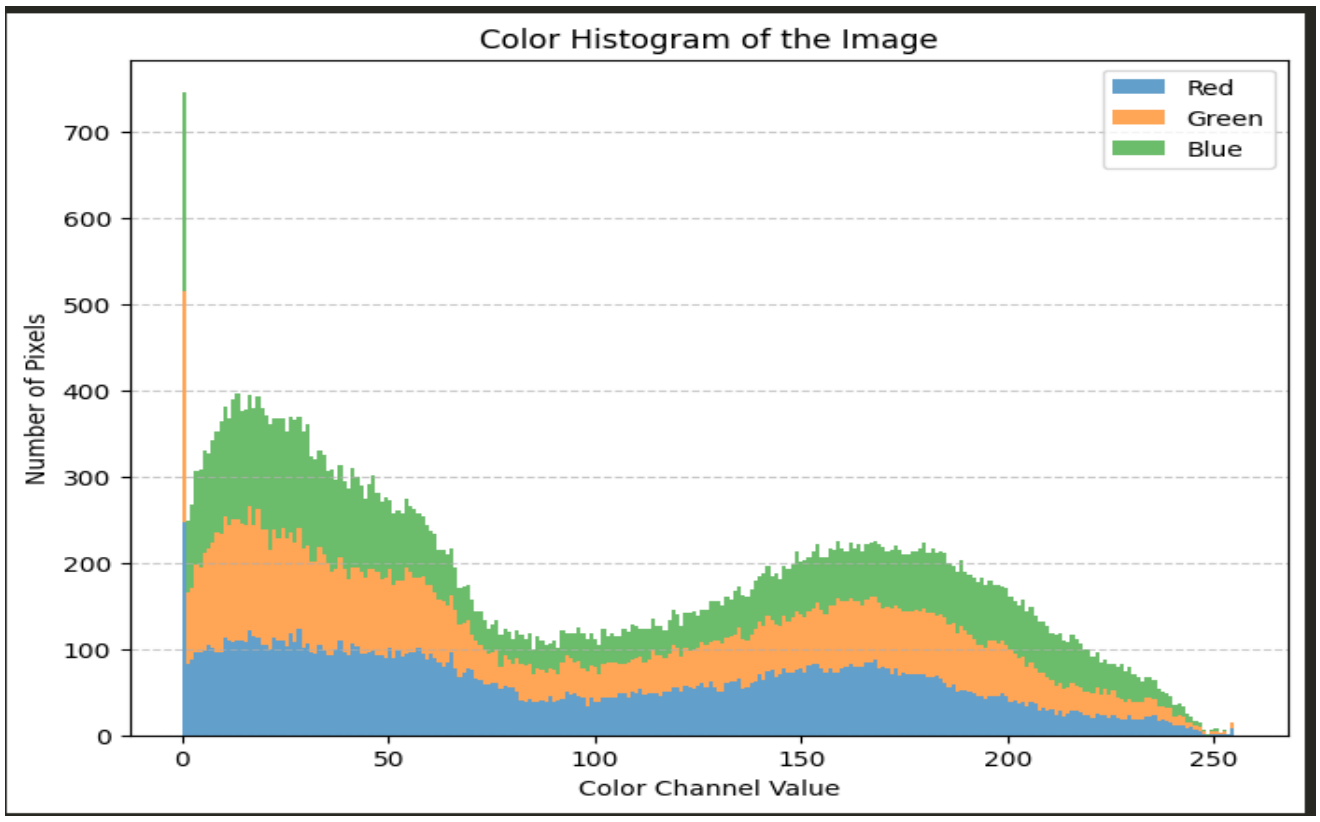
- Train folder :
  - Has total images 7706
  - Matched 7706 XML annotated file
  - Images with no bounding boxes = 3921
  - Number of classes 10
  - Number of bounding boxes = 8203



Again the color analysis for India dataset. The color distribution and color dominant .



Here is the color histogram for the India crack dataset.



From this analysis on two different road cracks dataset we found some keys that will help us during our model work as the distribution of class types in each dataset and also the number of training images will influence the training process.

Sign detection Model

1-*Business understanding and analysis for The Crack detection model:*

The second model we are implementing here can be used to notify the user about the upcoming Traffic signs in the road.
This would enable users to follow traffic rules and regulations,analyzing every sign whether it’s about speed limit or stop and following traffic lights ,This could achieve by implementing a model that could detect and recognize road signs carefully.

So, Our work here is also related to specific types of signs, scope, and conditions to match our needs to solve Egypt's road infrastructure.

1- Road sign Types :

- Green Light
- Red Light
- Stop sign
- Speed Limit signs (We have 12 speed limit to work with)

2-Model scope:

This model will be trained on this types of road signs that captured from highway roa as these roads are the most to have road signs that local roads.

3-Dataset conditions :

Also images captured in the morning are more predicted to be detected at high accuracy as the training dataset captured in the morning

4-Model goal :

Model goal is to be able to detect these types of signs in a reliable way to make me able to rely on it for alarming system .

The dataset reference used :

Traffic-Sign Detection and Classification in the Wild

Based on our model reference paper(*Improved YOLOv5 network for real-time multi-scale traffic sign detection*).The dataset used was (TT100k).

Its a huge dataset that contains many different signs captured from china streets and covers every possible option of road sign. It provides 100000 images containing 30000 traffic-sign instances. These images cover large variations in illuminance and weather conditions. Each traffic-sign in the benchmark is annotated with a class label, its bounding box and pixel mask.

This dataset annotated as Pascal VOC format, it contains many types of road signs but we extracted only the speed limits , stop and traffic light as this is our main concern here in the model.

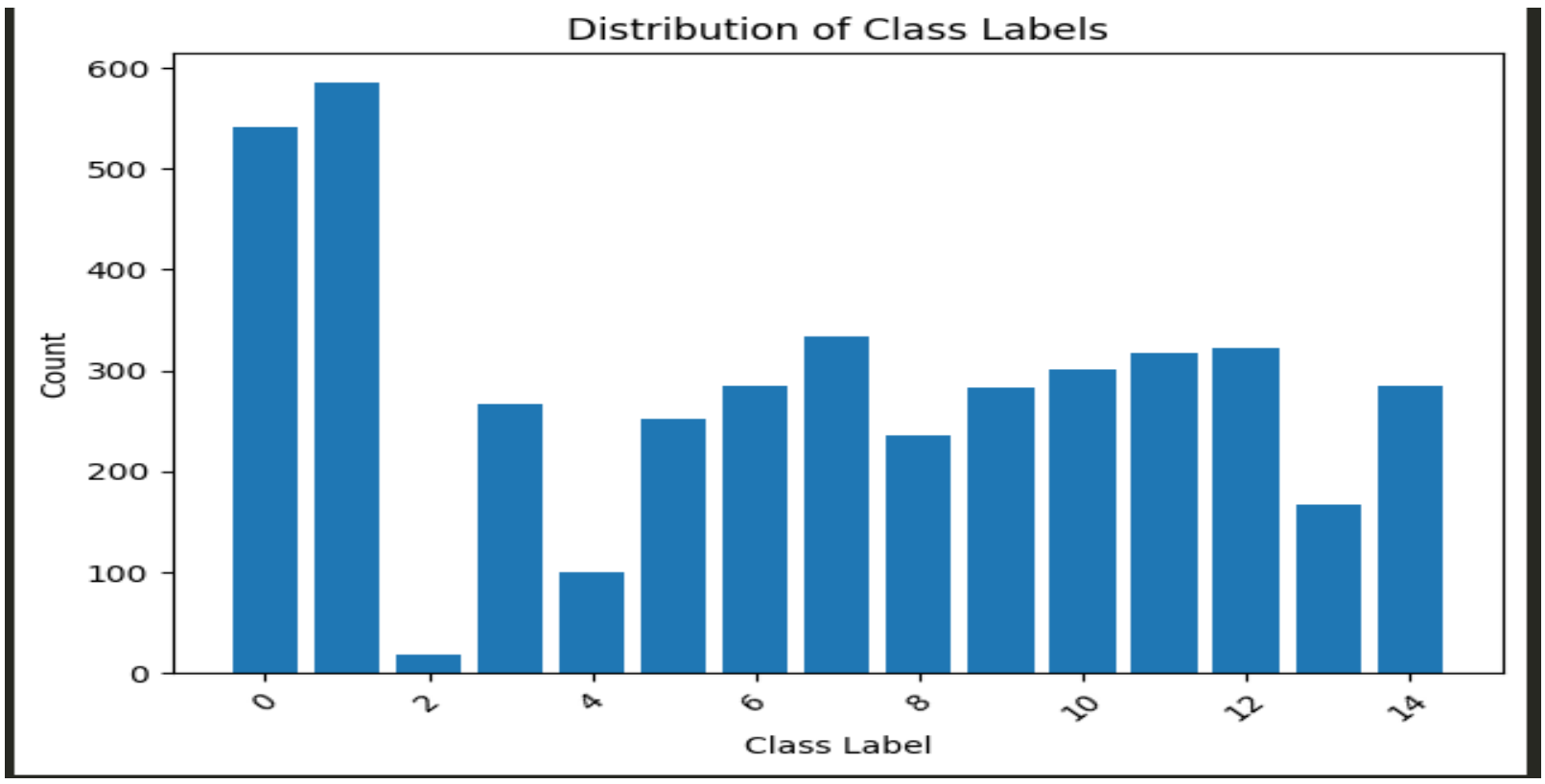
The Actual Dataset used Analysis :

1) Exploratory Data Analysis For the road signs dataset :

This dataset is annotated as pascal VOC format [XM Annotations] , image format is only PNG with resolution size 416*416.

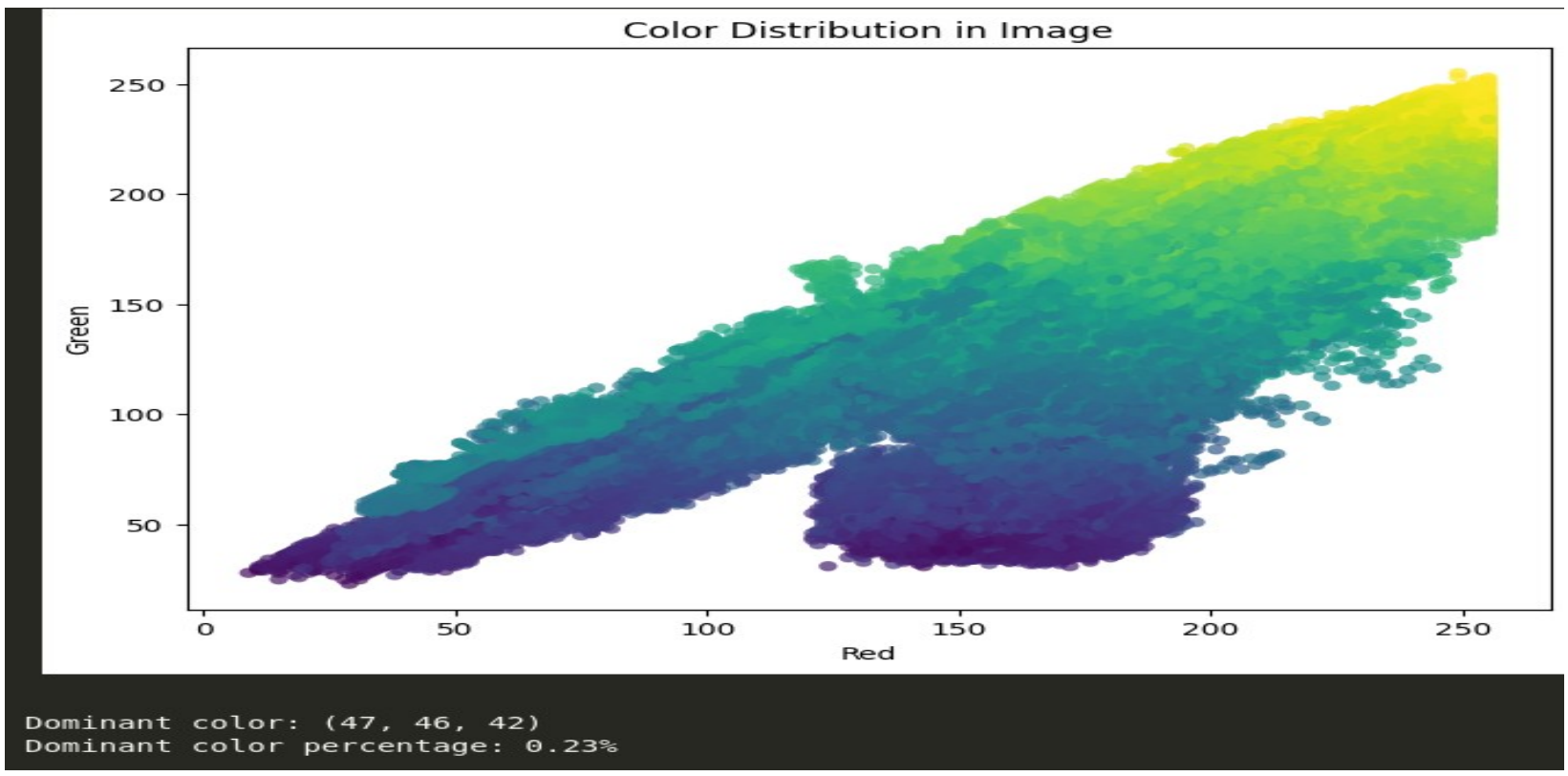
The Dataset structure :

- Train Folder :
  - Has 4969 images
  - Matches 4969 XML annotated file
  - Number of bounding boxes = 6012
  - Number of classes = 15
  - Image that has no signs =0

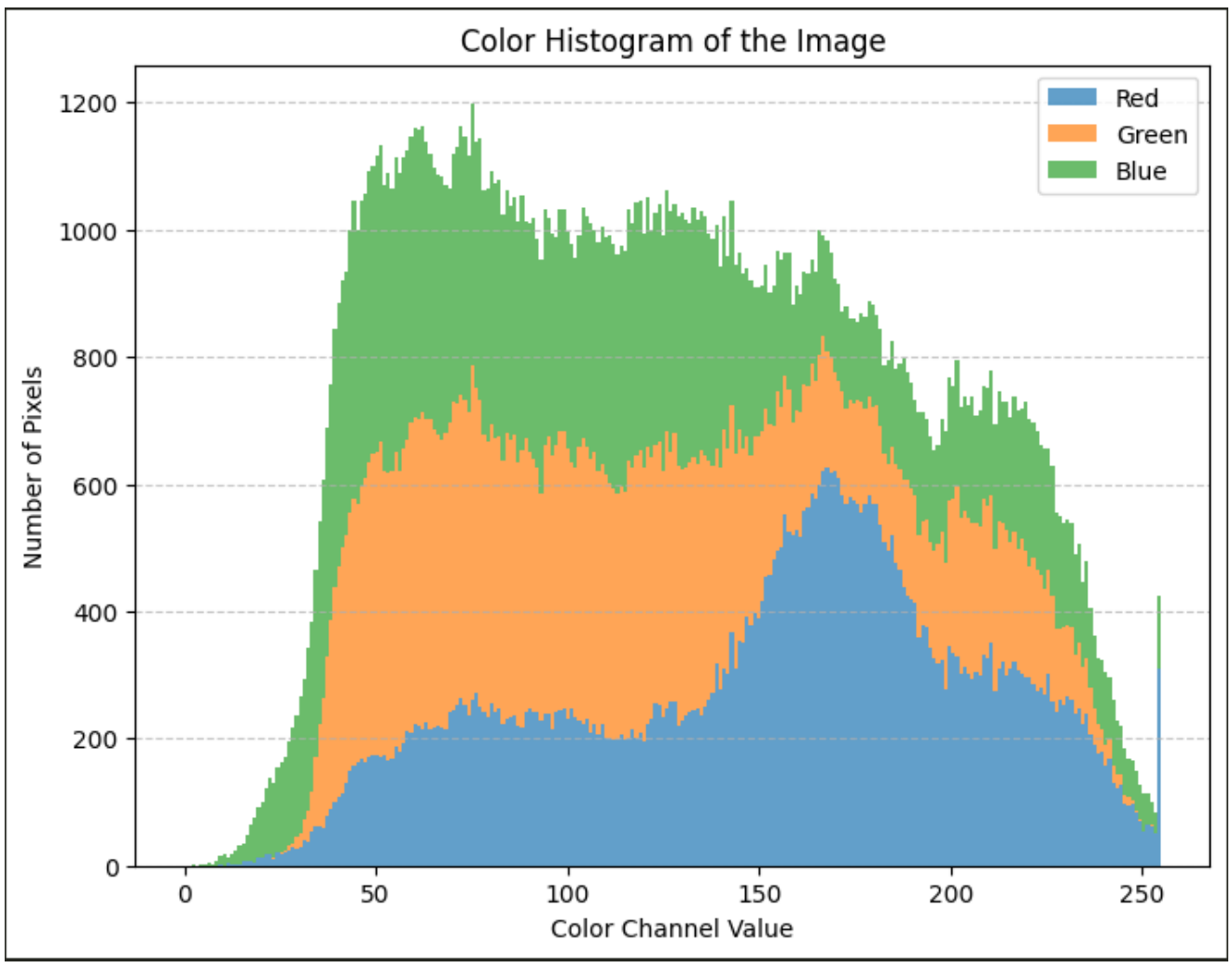


We also applied color analysis such as Color features to be able to distinguish our images according to their colors as here Traffic light images can be detected by getting the dominant color in the image.

Color Distribution and dominant color.



Also color histogram to show the range of colors present (from dark to bright values) and how spread out they are.



Model Analysis :

Our two models that we are trying to implement here are object detection models , so our goal in this section is to make a comparison between models that have been used in object detection tasks to choose one to apply the two dataset on and use it for our Application.

Object detection is a key task in computer vision, requiring models to accurately identify and locate objects within images. This paper compares several prominent object detection models: YOLOv8, Faster R-CNN, SSD, RetinaNet, and EfficientDet. We examine these models based on their architecture, performance metrics, and real-world applications, including numerical data to illustrate their comparative strengths and weaknesses.

1-YOLOv8

YOLOv8, the latest in the YOLO series, emphasizes real-time detection with a single-stage approach. It is known for its speed and efficiency.

- **Architecture**: Single-stage, unified model for detection and classification.
- **mAP**: 52-55% (mAP@0.5), 40-42% (mAP@0.5:0.95)
- **Speed**: 40-50 FPS on NVIDIA RTX 3080
- **Inference Time**: 20-25 ms per image

2-Faster R-CNN

Faster R-CNN is a two-stage detector that uses a Region Proposal Network (RPN) to generate candidate object regions.

- **Architecture**: Two-stage, with separate region proposal and classification stages.
- **mAP**: 41-44% (mAP@0.5), 35-37% (mAP@0.5:0.95)
- **Speed**: 7-10 FPS on NVIDIA RTX 3080
- **Inference Time**: 100-143 ms per image

3-SSD (Single Shot MultiBox Detector)

SSD is a single-stage detector that predicts bounding boxes and class scores without a separate region proposal step.

- **Architecture**: Single-stage, using multi-scale feature maps.
- **mAP**: 43-46% (mAP@0.5), 35-38% (mAP@0.5:0.95)
- **Speed**: 20-25 FPS on NVIDIA RTX 3080

- **Inference Time**: 40-50 ms per image

4-RetinaNet

RetinaNet is a single-stage detector known for its Focal Loss, which addresses the class imbalance issue.

- **Architecture**: Single-stage, using Focal Loss to improve performance.
- **mAP**: 49-51% (mAP@0.5), 39-41% (mAP@0.5:0.95)
- **Speed**: 12-15 FPS on NVIDIA RTX 3080
- **Inference Time**: 67-83 ms per image

5-EfficientDet

EfficientDet is a family of models optimized for both speed and accuracy, leveraging EfficientNet as its backbone and a novel BiFPN for feature fusion.

- **Architecture**: Single-stage, using EfficientNet backbone and BiFPN.
- **mAP**: 50-52% (mAP@0.5), 40-43% (mAP@0.5:0.95)
- **Speed**: 18-22 FPS on NVIDIA RTX 3080
- **Inference Time**: 45-55 ms per image

Model	mAP@0.5	mAP@0.5:0.95	FPS	Inference Time (ms)
YOLOv8	52-55%	40-42%	40-50	20-25
Faster R-CNN	41-44%	35-37%	7-10	100-143
SSD	43-46%	35-38%	20-25	40-50
RetinaNet	49-51%	39-41%	12-15	67-83
EfficientDet	50-52%	40-43%	18-22	45-55

So according to the previous given analysis we need a model to have high speed with high accuracy and this could be achieved by using YOLO v8.

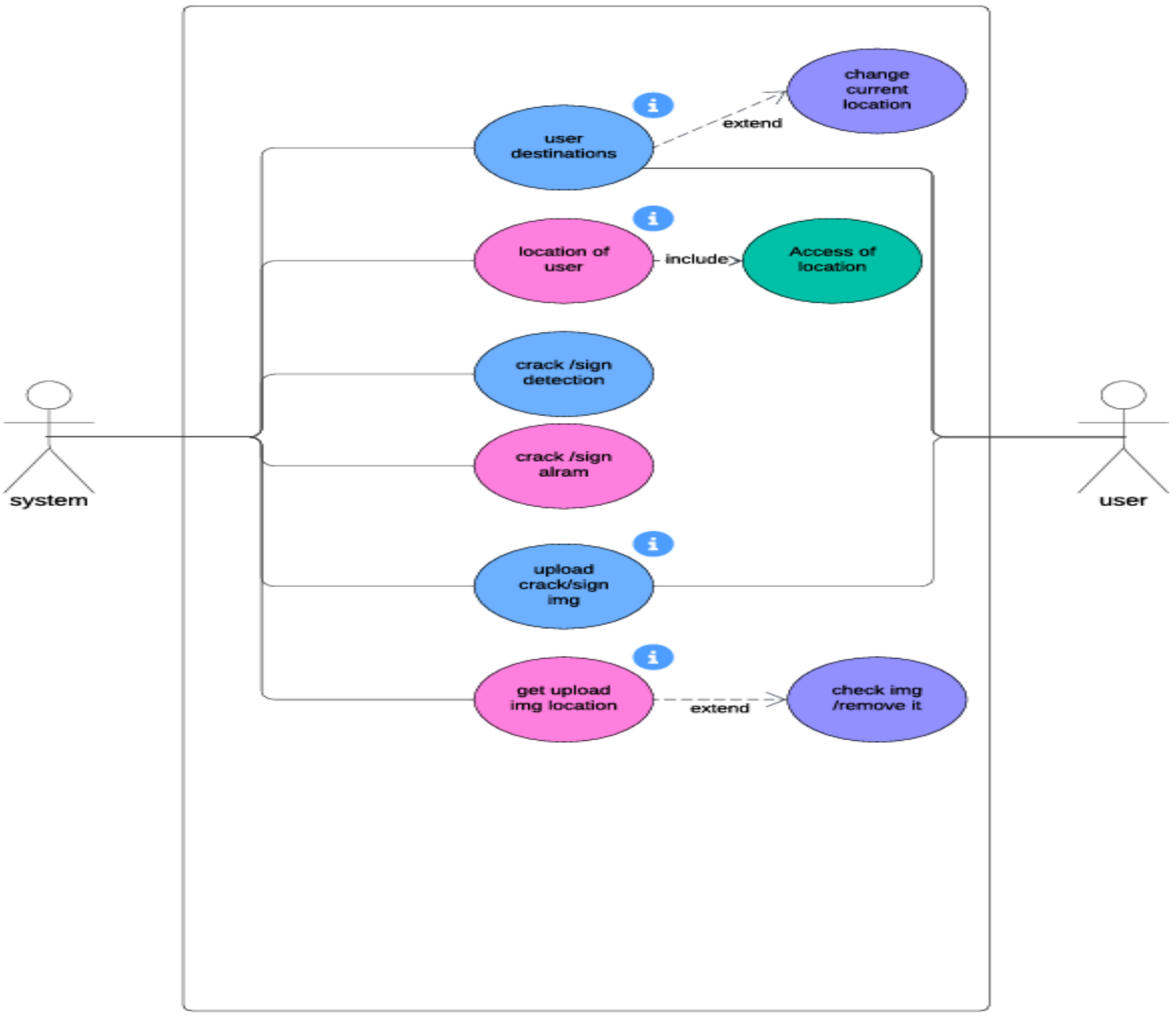
2-Application Analysis and design Diagrams

1. Use cases.

A use case is a description of a specific way in which a system can be used to achieve a particular goal or objective. It is a technique used in software engineering and other fields to capture the functional requirements of a system or product from the perspective of stakeholder. In a use case, the focus is on the interactions between the user or actor and the system. It typically includes a description of the preconditions, steps, and post conditions that are necessary for the user to achieve their goal using the system or product. Use cases can be represented in various ways, such as in natural language, diagrams, or formal notation.

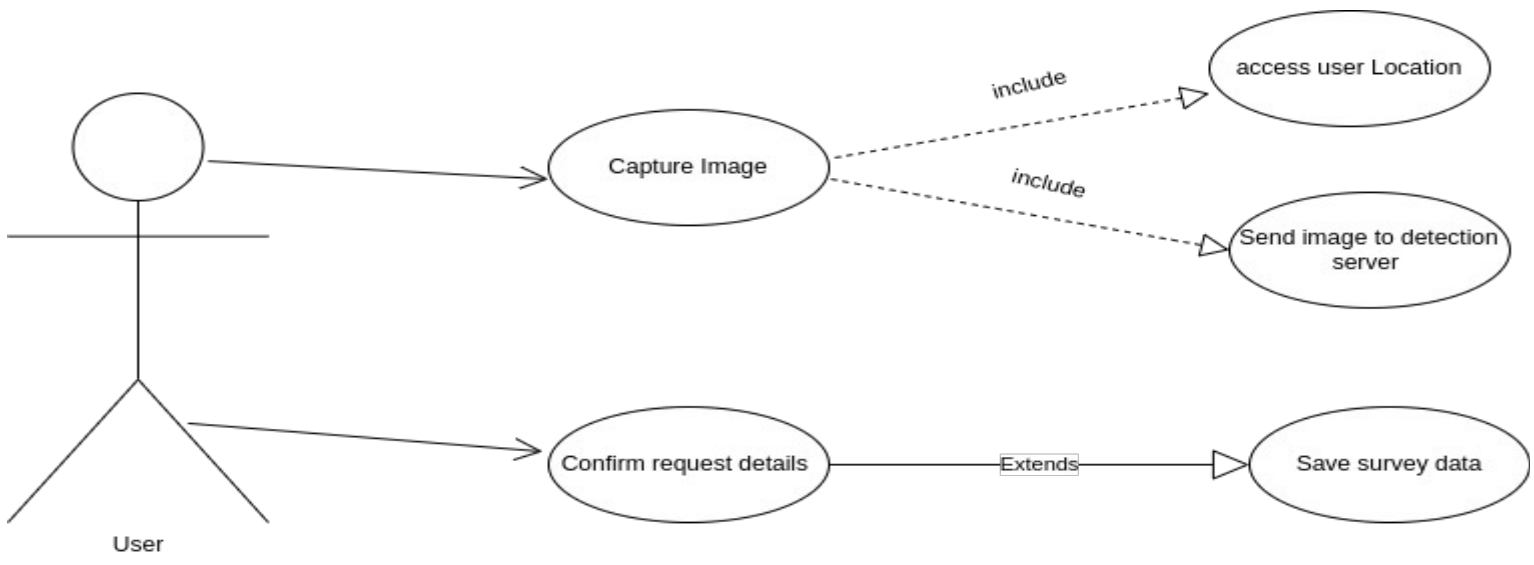
Actor	*System , User*
Description	*The system can detect the crack location and send alarm*
Precondition	*Must been sign up*
Post condition	*The user location will be added*
Main successful scenario	*Found a crack and send alarm*
unsuccessful scenario	*Didn’t find a crack*

System use case Diagram



1- Crack survey use case

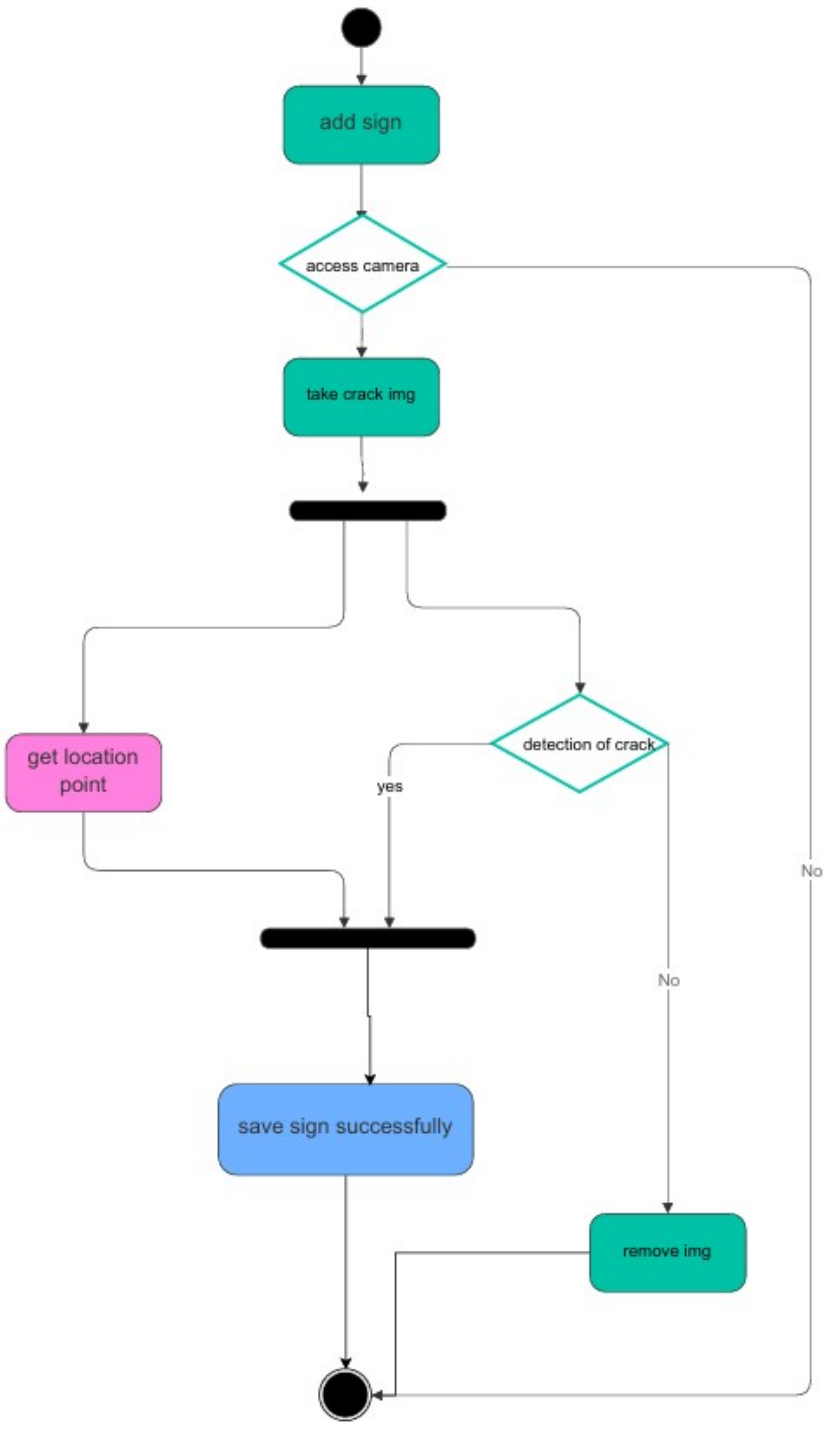
Shows how users interact with the feature and want the needed process in it.



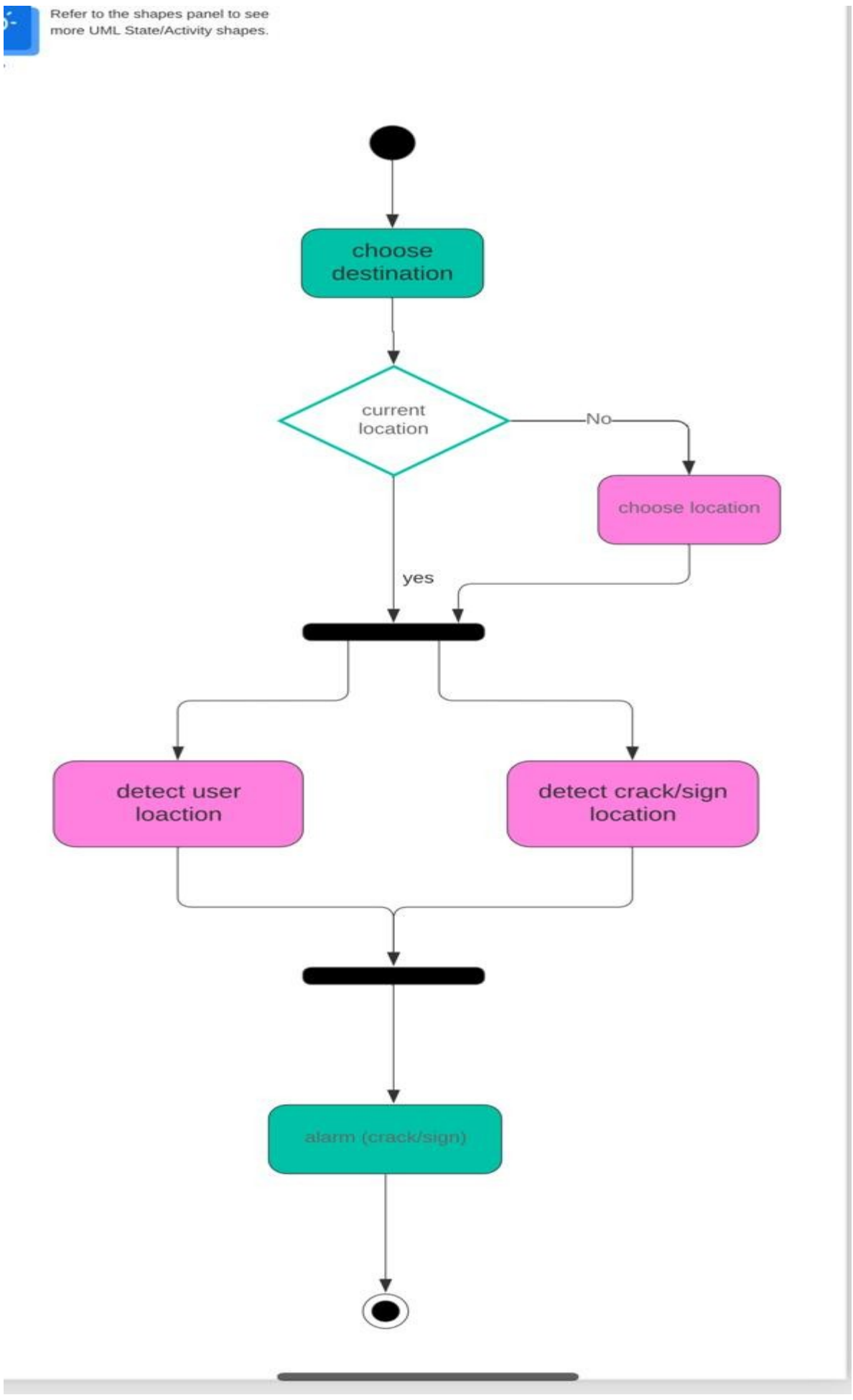
2-Activity diagram.

An activity diagram is a type of behavior diagram in the Unified Modeling Language (UM L) that shows the flow of activities or actions in a system, process, or workflow. It is used to model business processes, software applications, and other complex systems where there are multiple activities or actions that need to be coordinated. We use it to show steps or activities to user to do it to do specific function , we also describe the steps to achieve any functions in use case diagram . We also use the successful scenario in use case.

1- Crack Reporting : enable user to report the cracks he found and not locateted in the map .



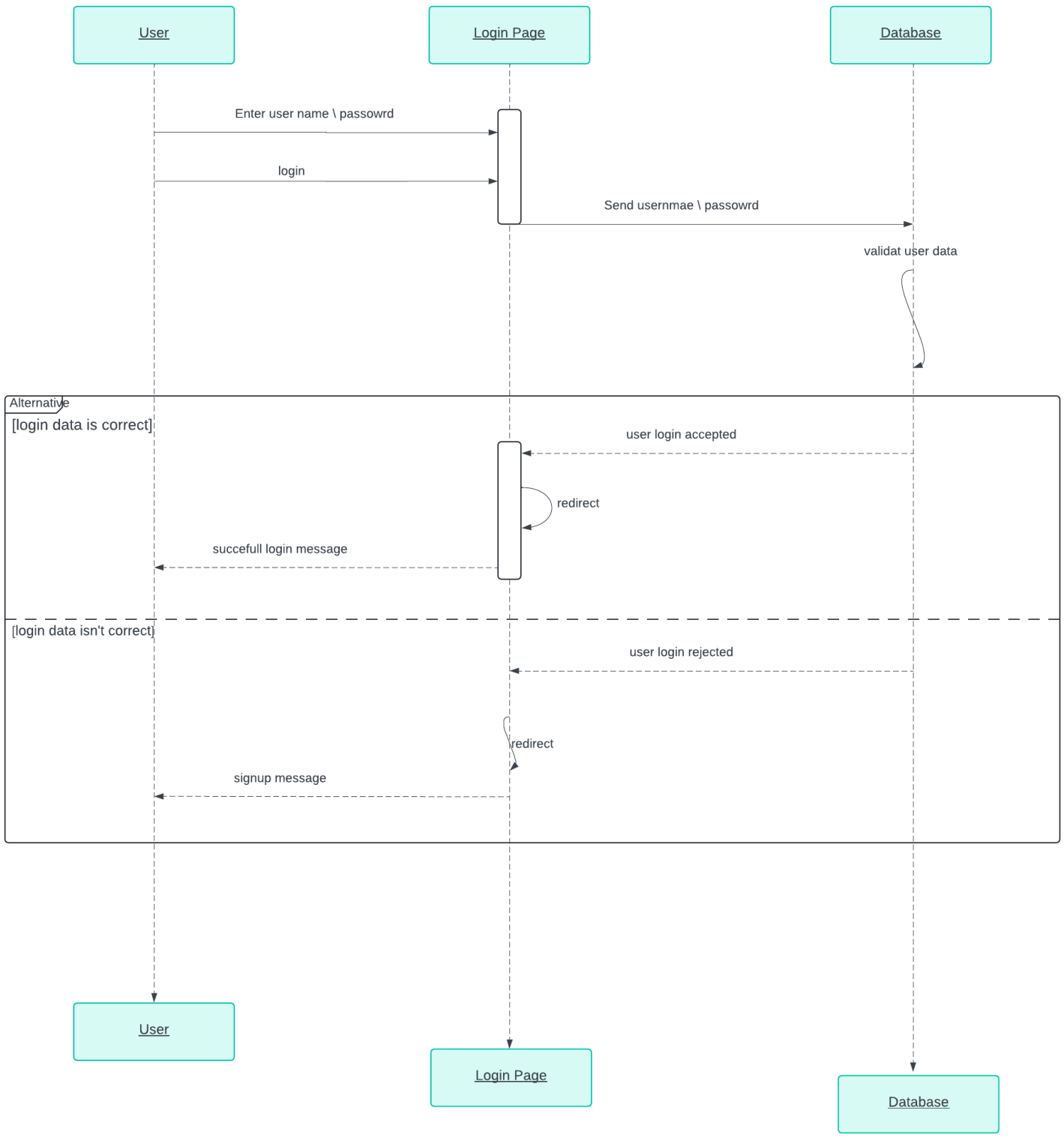
2- Alarm system : Alarm for the user for upcoming cracks



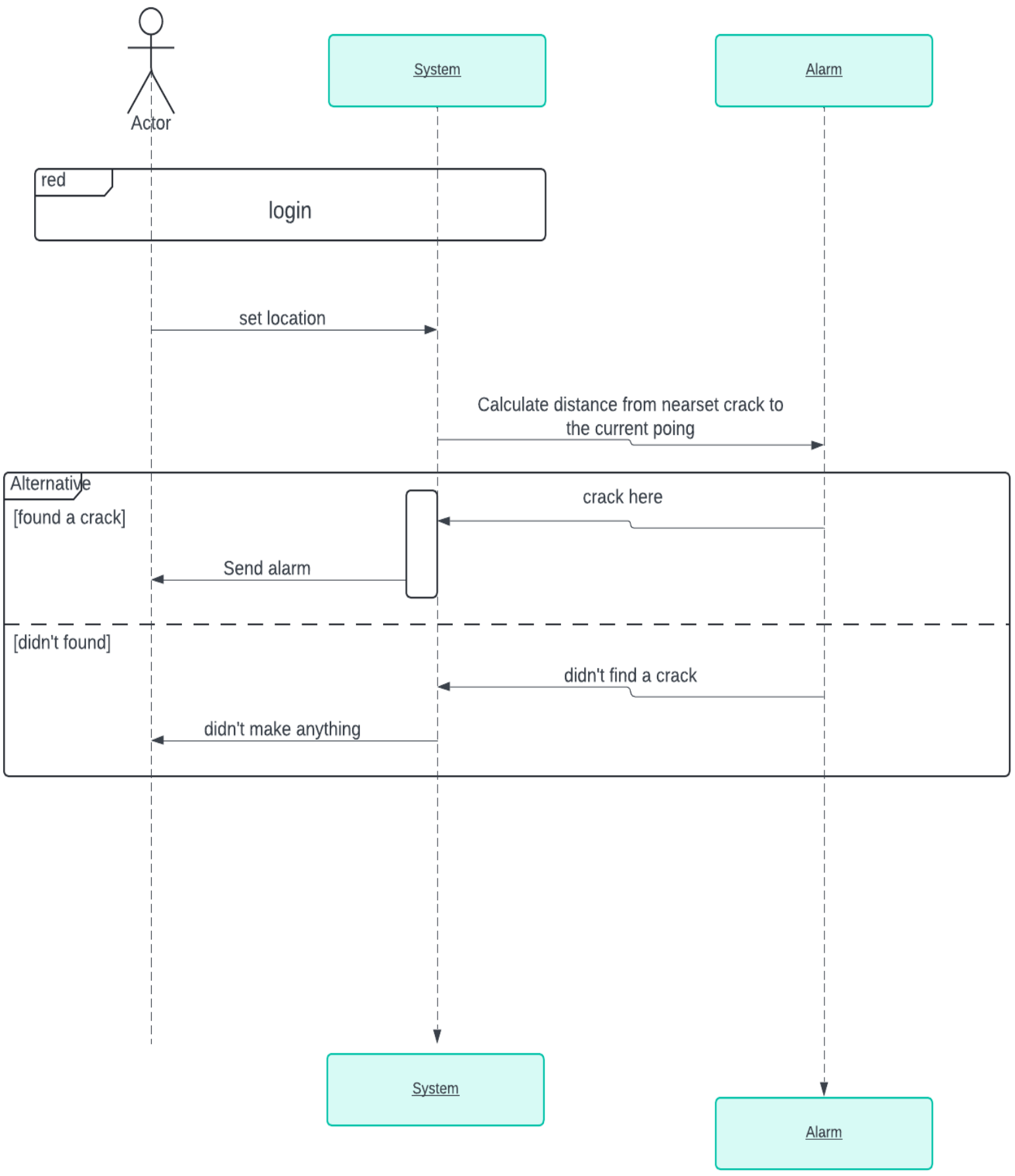
3-sequence diagram.

Sequence diagrams illustrate the interactions between objects or components in a system in a time-ordered manner. They are used to show the flow of messages and method calls between objects, along with the order in which they occur. Sequence diagrams are often used to model use cases or scenarios and can be used to identify potential issues or errors in the interaction between system components.

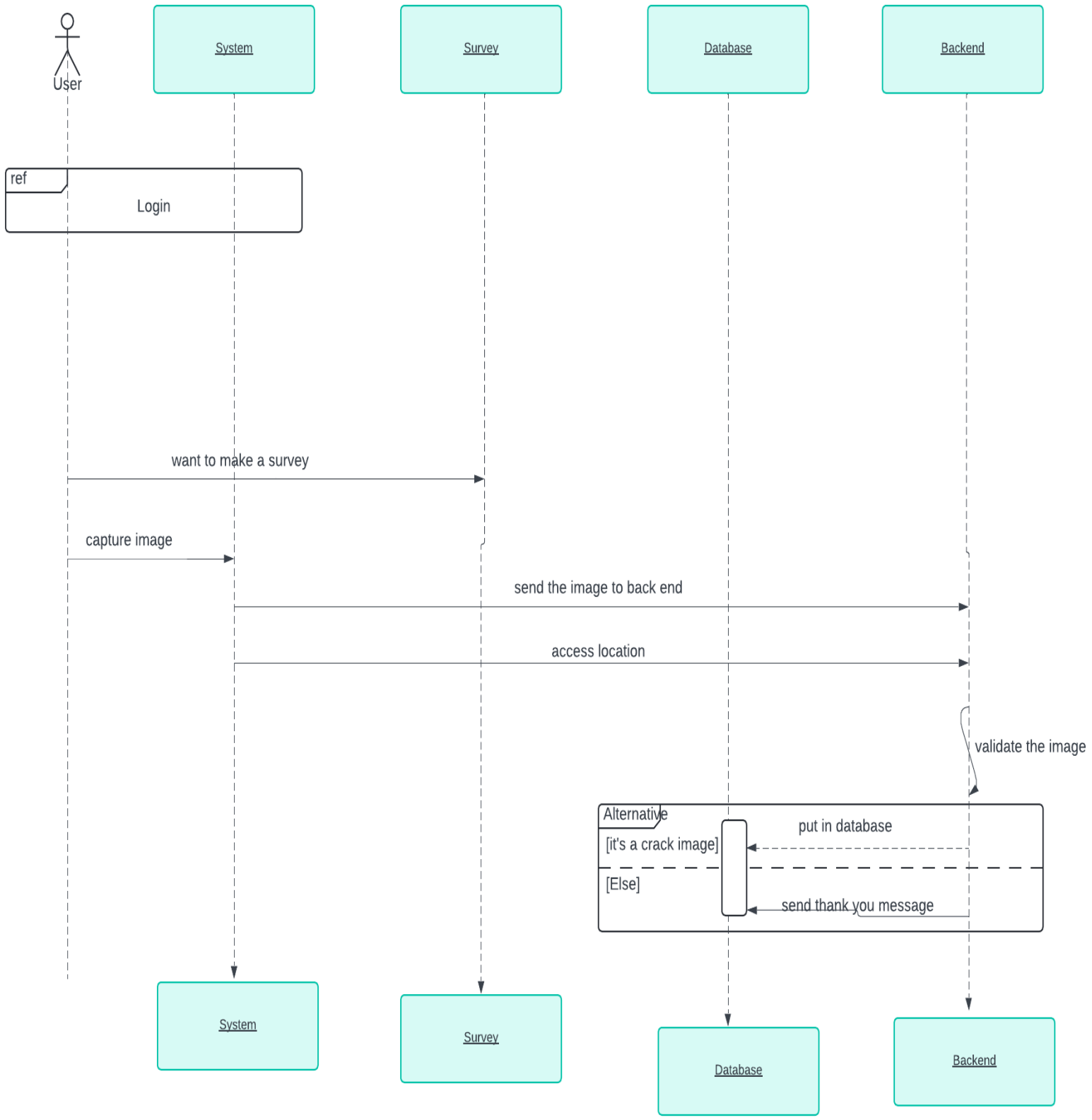
1-Login



2- Alarm System



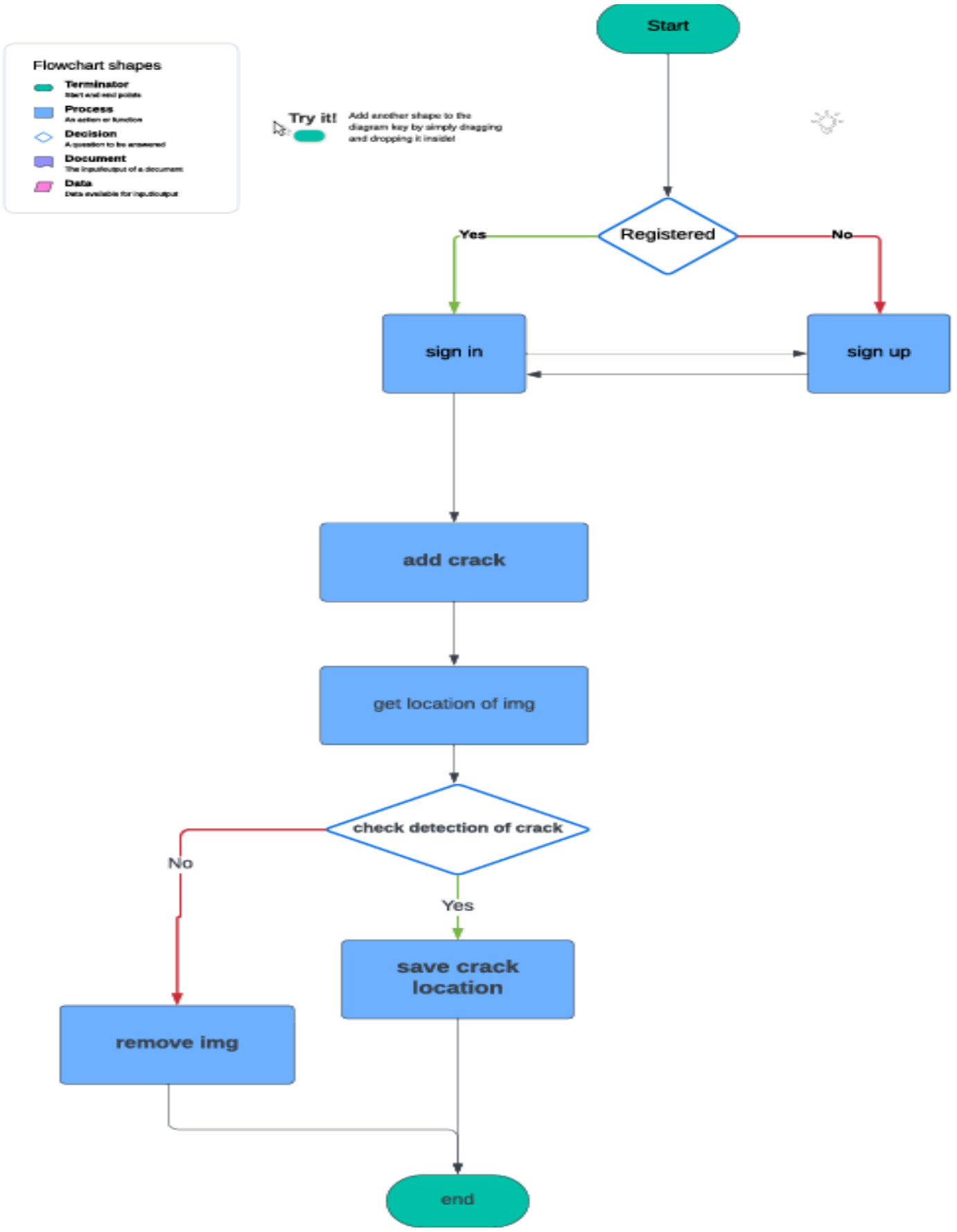
3-Crack survey



4- Flowchart diagram.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

1.App Flowchart.



3-Mapping Analysis

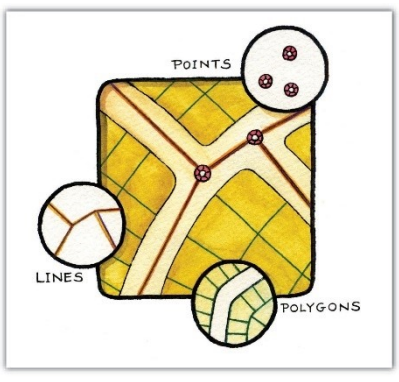
1-Visualizing Road Cracks and Road Signs: Representing the detected cracks and signs in points on a map Or Utilize imagery techniques such as high-resolution cameras or drones to capture detailed images of road cracks and signs.

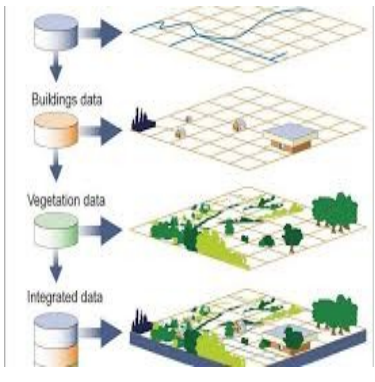
There are two main approaches to visualizing road cracks and signs:

- **Point-based Representation:** This is simpler and faster. It involves collecting location data for cracks and signs and displaying them as points on a map.
- **Imagery Techniques:** This provides a more detailed view. It involves capturing high-resolution images, using computer vision to automatically detect cracks and signs, and then extracting information about them (size, severity, type for cracks; content for signs).

The best approach depends on your needs. If you just need to know where cracks and signs are located, a point-based map is sufficient. If you need more detail about the cracks and signs themselves, then imagery techniques are a better option.

Feature	**Point-based Representation**	**Imagery Techniques**
Data representation	Points on a map	Images
Level of detail	Low	High
Information captured	Location only	Location, size, severity, type (cracks), content (signs)
Cost	Lower	Higher (equipment)
Processing complexity	Lower	Higher





2-Defining Details of the Map: Discover and gather the needed geospatial data. Ensure the map includes all necessary geographical details and layers for effective analysis.

Tailor Your Map:

- **Goal:** What story are you trying to tell (traffic patterns, population density)?
- **Audience:** Who will use the map (experts or general public)?

Set the Canvas (Optional):

- **Area:** City block, region, or the entire world?
- **Base Map:** Political map, satellite imagery, or something else?

Gather the Pieces:

- **Data Layers:** Roads, rivers, or your specific theme's data (crime rates, pollution).
- **Data Sources:** Government websites, commercial providers, or non-profits.

Data Check:

- **Quality:** Is the data accurate and recent?
- **Compatibility:** Can your mapping software use the data format?
- **Permissions:** Do you have the right to use the data?

Prepare the Data:

- **Download:** Get the data from the chosen sources.
- **Clean Up:** Fix any errors or inconsistencies.
- **Format:** Ensure all data uses the same system for alignment.

Visualize

- **Symbols & Labels:** Choose clear ways to represent data layers.
- **Map Elements:** Include scale, legend, north arrow, and title.
- **Integration:** Combine data layers with the base map to create your final map.

3-Defining the Schema for the Geodatabase: Define fields for attributes like location coordinates, crack dimensions, sign types, and conditions.

Data Entities: Decide what information to store (individual cracks or signs).

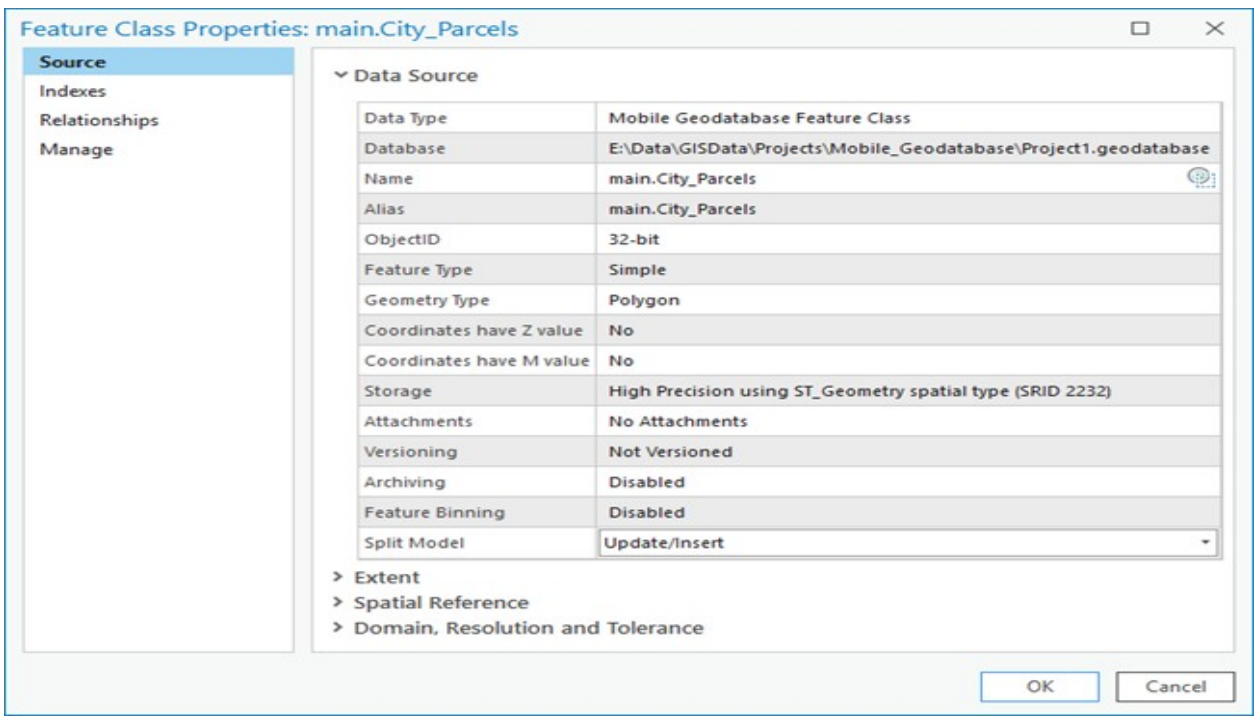
Location: Choose how to represent location (point, line, or area).

Attribute Design:

- Include location coordinates.
- For cracks: track dimensions, type, and severity (if applicable).
- For signs: record type, content (if extractable), and condition.
- Define data types for each attribute (text, number, date, etc.).

Data Validation: Set rules to ensure data accuracy (e.g., valid ranges for crack dimensions).

Document Everything: Clearly explain what each data point means.



4-DefiningFeature Class: Set appropriate data types for each attribute and ensure spatial reference is accurately defined.

- **Choose Feature Class Name:**

- Select a descriptive name that clearly indicates the type of features it will contain (e.g., "RoadCracks" or "TrafficSigns").

- **Define Feature Geometry:**

- Refer to step 2 of defining the schema (see previous explanation). Choose the appropriate geometry type to represent the location of each crack or sign (point, line, or polygon).

- **Set Attribute Data Types:**

- Assign a suitable data type for each attribute defined in the schema. Here's a breakdown of common data types and examples:
    1. **Text:** For short descriptive names or alphanumeric data (e.g., crack type, sign content).
    2. **Double:** For numerical data with decimals (e.g., crack dimensions).
    3. **Integer:** For whole numbers (e.g., severity rating).
    4. **Date:** For recording the date of data collection or inspection (optional).
    5. **Boolean:** For true/false values (e.g., indicating a sign is damaged).

- **Define Spatial Reference:**

- This is crucial for ensuring accurate geographic positioning of your features.
- Choose a spatial reference system (SRS) that is compatible with the area your data covers.
- Common options include latitude/longitude coordinate systems with a specific projection (e.g., UTM, WGS84).

- **Domain and Validation Rules (Optional):**

- **Domains:** You can define a limited set of valid values for specific attributes (e.g., restricting crack type options).
- **Validation Rules:** Set rules to check data upon entry and prevent invalid values (e.g., ensuring crack width is positive).

- **Field Aliases (Optional):**

- Assign user-friendly aliases to attribute names for easier visualization and understanding within the software interface.

5-Constructing Methodology to Add Features to the Feature Layers: Develop a systematic approach for adding new features to the layers. This includes field data collection, data entry procedures, and quality assurance checks.

1. Field Data Collection:

- Choose methods: visual inspection, mobile apps, or remote sensing (optional).
- Design forms (paper or digital) to capture all needed data (location, dimensions, type, etc.).

2. Data Entry Procedures:

- Establish a workflow to transfer data from forms/apps to the geodatabase.
- Define data entry standards for consistency and accuracy (format, validation rules).

3. Quality Assurance (QA) Checks:

- Clean data to identify and fix errors (missing values, outliers, etc.).
- Validate data to ensure it meets defined standards and rules (data types, pre-defined checks).
- Verify data by comparing it to existing data/imagery or conducting field audits.

4. Documentation:

- Document the entire methodology for adding features:
  - Data collection methods
  - Data entry procedures and standards
  - Quality assurance checks
  - Roles and responsibilities

6-Choosing Survey Type to Match Limited Resources: Select a cost-effective and efficient survey method, such as mobile-based surveys or UAV surveys, considering available resources and the scope of the project.

Ground Survey Methodologies:

Encompass various techniques for gathering field data.

Essential for validating remote sensing data and informing decision-making.

Types of Ground Surveys:

Traditional Manual Surveys:

- Conducted by field teams equipped with GPS devices and measuring instruments.
- Direct measurements of ground features like elevation, vegetation coverage, and infrastructure details.

- **LiDAR Surveys (Light Detection and Ranging)**:
  - Utilize laser technology to capture highly accurate 3D terrain data.
  - Provide detailed information on terrain morphology and surface characteristics.

- **Drone-Based Surveys**:
  - Offer flexible and cost-effective solutions for capturing aerial imagery.
  - Useful for conducting localized inspections and monitoring inaccessible areas.

Selection Considerations:

- Factors include project scope, budget, terrain complexity, and data accuracy requirements.
- Each survey type has advantages and limitations.

Benefits:

- Comprehensive and reliable geospatial data for urban planning, environmental monitoring, infrastructure development, and natural resource manageme

By following these steps and considering your specific needs, you can effectively visualize road cracks and signs, manage your geospatial data, and gain valuable insights for infrastructure maintenance and road safety improvements.

4-Mapping Designing

- **Visualizing Road Cracks and Road Signs**
  To visualize road cracks and road signs, utilize advanced image processing techniques alongside GIS tools. This approach enables the detailed visualization of these road features, providing a clear and precise representation on a map. Techniques such as those available in OpenCV can process images to detect and highlight cracks and signs, which are then imported into GIS software like ArcGIS or QGIS for spatial representation and analysis.

1. Define Goals:

- Determine the information to display (location, size, severity of cracks; type, content of signs).
- Consider the target audience's needs (engineers, public works, citizens).

2. GIS Data Preparation:

- Integrate existing GIS data:
  - Road network data for spatial context.
  - Traffic data (optional) for further analysis.

3. Image Processing Integration:

- Import processed data from image processing tools:
  - Location and attribute data for detected cracks and signs.
  - Crack dimensions, severity classification (if applicable).
  - Sign type and extracted text content (if available).

4. Spatial Representation:

- Choose appropriate GIS representations for cracks:
  - Points for small, localized cracks.
  - Lines for elongated cracks.
  - Polygons for complex crack shapes.
- Represent signs using point symbols with variations based on type and color.

5. Symbology and User Interaction (Optional):

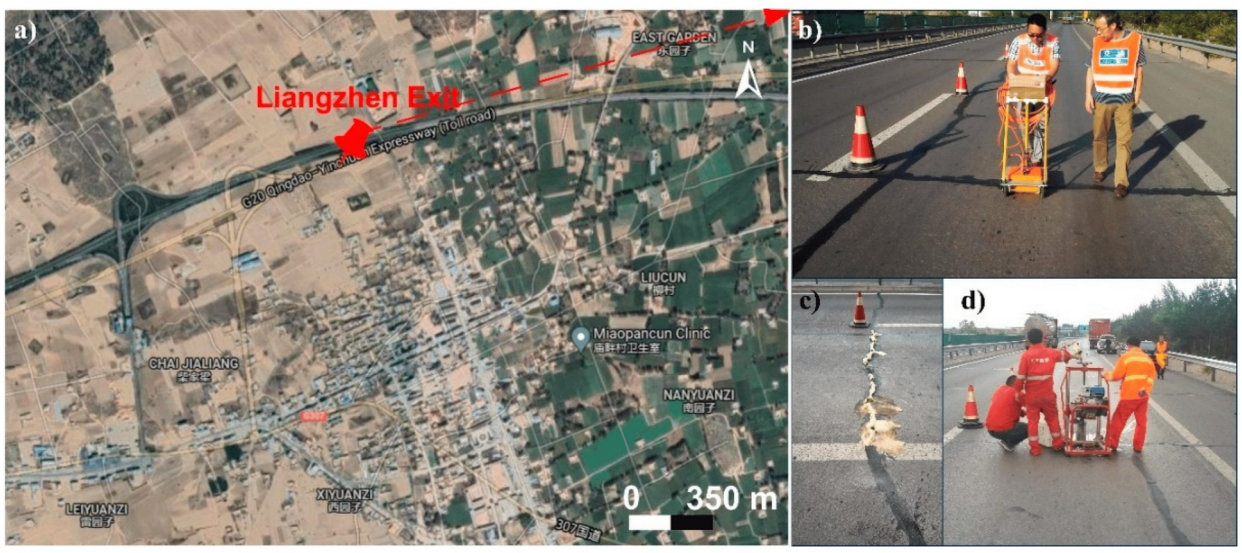
- Design clear symbology for cracks and signs based on severity, type, and content.

- Implement interactive features within the GIS software (e.g., click on a crack for details).
- Develop dashboards or reports summarizing crack & sign distribution.

6. Sharing and Analysis:

- Export the final map in a suitable format (image, web map) for analysis or sharing.
- Leverage the GIS platform to perform spatial analysis on crack & sign data.

This approach allows for a detailed and informative visualization of road cracks and signs within a GIS environment, enabling further analysis and decision-making for infrastructure management purposes.



- **Map Construction**
  The map construction process involves using GIS software to design and implement a map schema, including base layers like road network, terrain, and administrative boundaries. The map should integrate data sources and provide a comprehensive spatial framework for analysis and feature integration.

1. **Define Map Purpose and Audience:**

- Determine the overall goal of the map (infrastructure assessment, public awareness).
- Consider the needs of the target audience (engineers, public works, citizens).

2. **Base Layer Selection:**

- Choose a base map that provides context and aids visualization:

- Detailed street map for precise location reference.
- Satellite imagery for a broader view of the area.
- Topographic maps for terrain analysis (if relevant).

3. Data Integration:

- Import existing geospatial data relevant to your analysis:
  - Road network data (essential for crack and sign placement).
  - Administrative boundaries (city limits, districts).
  - Additional thematic data layers (e.g., traffic data, land use) for a more comprehensive view (optional).

4. Symbology and Labeling:

- Design clear and informative symbology for base map elements (roads, boundaries).
- Establish a legend to explain the meaning of symbols and colors used.

5. Map Layout and Design:

- Arrange map elements (title, legend, north arrow, scale bar) for optimal readability.
- Consider incorporating visual hierarchy to emphasize key information.

6. Map Projection:

- Choose a suitable map projection that accurately represents the spatial relationships of your study area.

7. Scalability and Future Use:

- Design the map with future updates and data integration in mind.

By following these steps, you can create a well-designed and informative base map within your GIS software, providing a solid foundation for integrating road crack and sign data for further analysis and visualization.

- **Geodatabase Schema**
  Defining the schema for the geodatabase is crucial to store geospatial data effectively. The schema should include tables and attributes tailored for road elements. For instance, a table for road cracks might include fields such as ID, Location, Severity, and Date_Detected, while a table for road signs could include fields like ID, Location, Sign_Type, Condition, and Date_Detected. This structured approach ensures organized and efficient data storage.

1. **Data Entity Definition:**

   - Identify the primary data entities you want to represent (e.g., "RoadCrack" and "RoadSign" as separate feature classes).

2. **Spatial Representation:**

   - Determine how the location of each crack and sign will be stored:
     - Points for individual locations.
     - Lines for elongated cracks (optional).
     - Polygons for complex crack shapes (optional).

3. **Attribute Design:**

   - Define relevant attributes for each data entity:
     - Include a unique identifier field (ID).
     - Store location data using geographic coordinates (latitude, longitude).
     - For cracks: add attributes like "Severity" (text or coded values) and "Date_Detected" (date).
     - For signs: include attributes like "Sign_Type" (text or coded values) and "Condition" (text or coded values).

4. **Data Type Selection:**

- Assign appropriate data types for each attribute:
  - Text for descriptive names and short text entries.
  - Numbers for measurements and rankings (severity).
  - Date for recording detection time.

5. **Data Validation (Optional):**

- Implement rules to ensure data accuracy upon entry:
  - Define valid ranges for values (e.g., severity levels).
  - Create domain lists to restrict options for specific attributes (e.g., pre-defined crack types).

6. **Metadata Documentation:**

- Clearly document the meaning and purpose of each attribute within the geodatabase schema.

By following these steps, you can design a well-structured geodatabase schema in your GIS software, specifically tailored for storing and managing road crack and sign data efficiently. This will facilitate further analysis, visualization, and data management tasks.

- **Feature Class Creation**
  Creating feature classes for different road elements within the GIS software is the next step. Feature classes such as Road_Cracks and Road_Signs are established, each with its own set of attributes as defined in the schema. These feature classes serve as containers for spatial data, enabling detailed and specific analysis of each road element.

1. **Define Feature Classes:**

- Based on your schema, create separate feature classes in your GIS software:
  - "Road_Cracks" to store data on individual cracks.
  - "Road_Signs" to store data on individual signs.

2. **Link to Schema Attributes:**

- Associate each attribute defined in the schema with its corresponding feature class. This ensures data is stored and managed according to the defined structure.

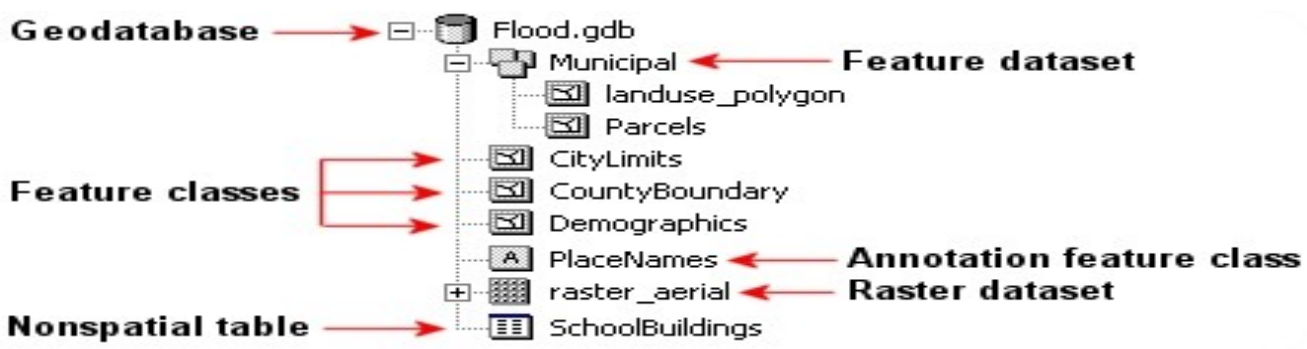
3. Spatial Reference:

- Assign a suitable spatial reference system (SRS) to the feature classes. This ensures accurate geographic positioning of your data and compatibility with other spatial data layers within your GIS project.

4. Metadata (Optional):

- Provide metadata for each feature class, documenting its purpose and the type of data it contains.

By following these steps, you can create well-defined feature classes within your GIS software, specifically designed to store and manage road crack and sign data based on your schema. This enables you to perform detailed spatial analysis and create informative visualizations of these road elements.



- **Feature Layers Creation**
  Develop individual feature layers for road cracks and road signs, ensuring each layer includes the necessary attributes. This step involves populating the feature classes with data points representing the actual road conditions. Each feature layer is then customized with symbology and labeling to enhance visualization and understanding.

1.Populating Feature Classes:

- Import data points representing road cracks and signs into their respective feature classes ("Road_Cracks" and "Road_Signs").
- This data may come from various sources like field surveys, image processing results, or existing databases.

2. Symbology and Labeling:

- Design clear and informative symbology for each feature layer:
  - **Road Cracks:** Use different point symbols, colors, or sizes to represent severity levels (e.g., red for critical, yellow for moderate).
  - **Road Signs:** Utilize point symbols with variations in color and shape to represent different sign types (e.g., red octagon for stop sign, yellow diamond for yield sign).
- Implement clear labeling to display relevant attribute information when hovering over a feature (e.g., crack severity, sign type, date detected).

3. Advanced Customization (Optional):

- Apply graduated colors or proportional symbols to feature layers to visually represent additional attributes (e.g., crack width for cracks, size for signs).
- Create pop-up windows displaying detailed attribute information for each feature upon clicking.

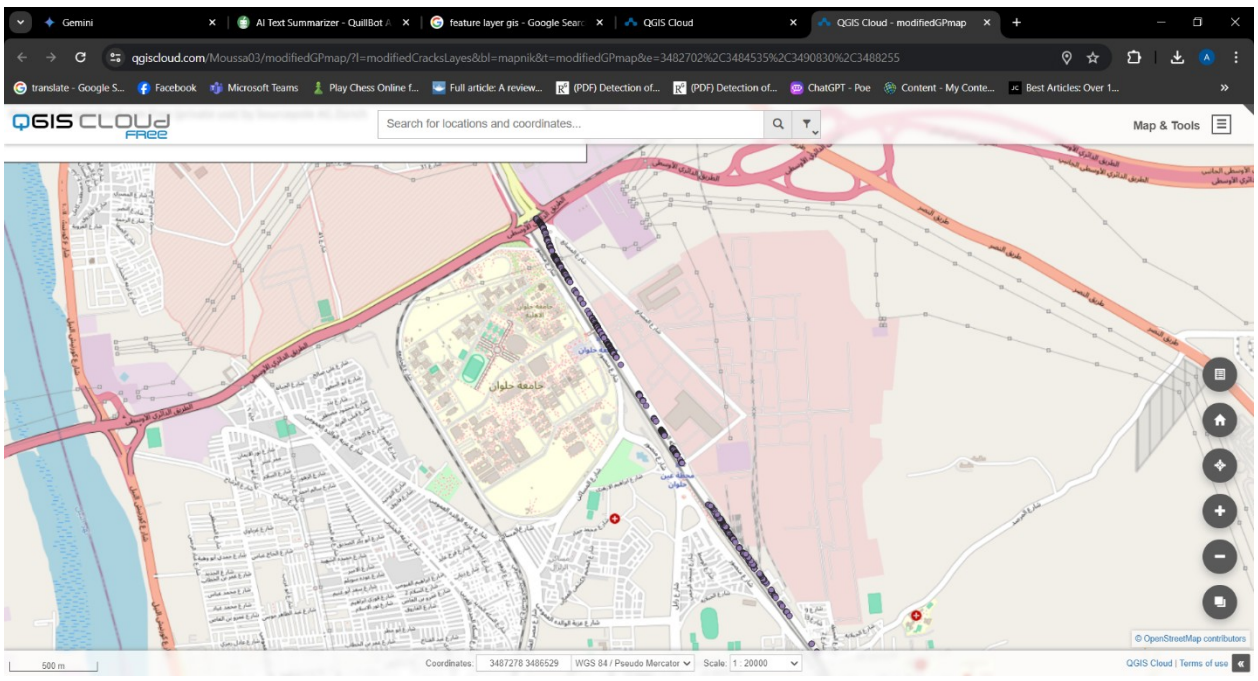
4. Quality Control:

- Perform visual inspection and data validation to ensure accurate representation of road cracks and signs within the feature layers.

5. Layer Management:

- Organize feature layers within the GIS project for clear hierarchy and efficient visualization.

By following these steps, you can create informative and visually appealing feature layers for road cracks and signs within your GIS software. This allows for clear communication of road conditions, facilitating analysis and informed decision-making for infrastructure maintenance and safety improvements.



- **Feature Addition Methodology**
  Establish a methodology for adding features to the layers, detailing protocols for data collection, entry, and updates. This process involves using tools like Esri'sQuickCapture and Field Maps to facilitate real-time data recording and ensure that new features are accurately and efficiently incorporated into the GIS database.

1. Data Collection Methods:

- Choose appropriate methods for collecting data on new road cracks and signs:
  - **Field Surveys:** Utilize GPS devices and data collection apps (e.g., Esri's Collector, QuickCapture) for real-time data recording.
  - **Image Processing Integration:** If using image processing techniques, ensure seamless integration with the GIS for data import and attribute population.

2. Data Entry Procedures:

- Define clear procedures for entering collected data into the GIS:
  - Establish data entry forms (paper or digital) capturing location, attributes, and photos (optional).
  - Set data entry standards for consistency and accuracy (format, validation rules).

3. Quality Assurance (QA) Checks:

- Implement a QA process to ensure data integrity:
  - Clean data to identify and fix errors (missing values, outliers).
  - Validate data against pre-defined rules and attribute domains.
  - Conduct field audits or utilize image re-analysis to verify data accuracy.

4. Version Control (Optional):

- Consider implementing version control within the GIS to track changes made to the feature layers over time.

5. Documentation:

- Clearly document the entire methodology for adding features:
  - Data collection methods used
  - Data entry procedures and standards
  - Quality assurance checks
  - Roles and responsibilities for data collection and updates

By following these steps, you can establish a robust and efficient methodology for adding new road cracks and signs to your GIS database. Utilizing mobile data collection tools and implementing quality assurance measures ensures accurate and up-to-date information for ongoing analysis and visualization of road conditions within your GIS environment.

By following these steps and considering your project's specific needs, you can create a robust GIS workflow for visualizing road cracks and signs. This will facilitate effective analysis, decision-making, and ultimately, improved infrastructure management.

Chapter 4

Implementation

In this chapter we will discuss and in details the implementation methods and details about our application including the three main components Deep learning models implementations , Map implementation and Android App implementation.

4.1-Implementation Details

1-Model Implementation :

YOLOv8, developed by Ultralytics, builds upon the success of prior YOLO versions by introducing advancements in object detection, segmentation, and other computer vision tasks. It boasts state-of-the-art performance through a streamlined design that facilitates adaptation across various hardware platforms.

One key distinction of YOLOv8 is its versatility. In contrast to previous YOLO versions primarily focused on detection, YOLOv8 offers a comprehensive suite of capabilities including detection, segmentation, pose estimation, tracking, and classification. This allows researchers to leverage a single model for a wider range of computer vision applications within their academic pursuits. Additionally, YOLOv8 maintains the focus on speed and accuracy that characterized prior YOLO models, making it well-suited for real-time applications in academic settings.

CNN-based Object Detectors are primarily applicable for recommendation systems. YOLO (You Only Look Once) models are used for Object detection with high performance. YOLO divides an image into a grid system, and each grid detects objects within itself. They can be used for real-time object detection based on the data streams. They require very few computational resources.

YOLOv8 is the latest version of the YOLO algorithm, which outperforms previous versions by introducing various modifications such as spatial attention, feature fusion, and context aggregation modules.

Multiple features are to be focused on in YOLOv8. Here are some key features of YOLOv8:

Improved Accuracy: YOLOv8 improves object detection accuracy compared to its predecessors by incorporating new techniques and optimizations.

Enhanced Speed: YOLOv8 achieves faster inference speeds than other object detection models while maintaining high accuracy.

Multiple Backbones: YOLOv8 supports various backbones, such as EfficientNet, ResNet, and CSPDarknet, giving users the flexibility to choose the best model for their specific use case.

Adaptive Training: YOLOv8 uses adaptive training to optimize the learning rate and balance the loss function during training, leading to better model performance.

Advanced-Data Augmentation: YOLOv8 employs advanced data augmentation techniques such as MixUp and CutMix to improve the robustness and generalization of the model.

Customizable Architecture: YOLOv8's architecture is highly customizable, allowing users to easily modify the model's structure and parameters to suit their needs.

Pre-Trained Models: YOLOv8 provides pre-trained models for easy use and transfer learning on various datasets.

YOLOv8 seems to be a promising advancement with the potential for faster and potentially more accurate object detection compared to both YOLOv2 and YOLOv5. However, for established benchmarks and confirmed performance, YOLOv5 remains a solid choice. The final decision depends on the specific needs of your project. If real-time performance and potential for accuracy gains are crucial, YOLOv8 might be worth exploring once it's officially released. If established performance benchmarks and ease of use are priorities, YOLOv5 is a strong option.

1-YOLO V8 architecture :

Yolo v8 architecture consists of three blocks,The first one is Backbone Block which acts as a feature extractor for the model.
The backbone block consists of

- Conv blocks : Each one is a normal convolutional block that contain conv2 layer , batchNorm layer and SLIU layer.
- C2F blocks : it takes input(Feature map) from conv Block and split it to Bottleneck block and then concatenates outputs , it enhances feature extraction and regulates dimensions.
- SPPF block : it has the MaxPooling layers , each resulting feature map is concatenated before the end , it also represents a fixed representation of object sizes without resizing.

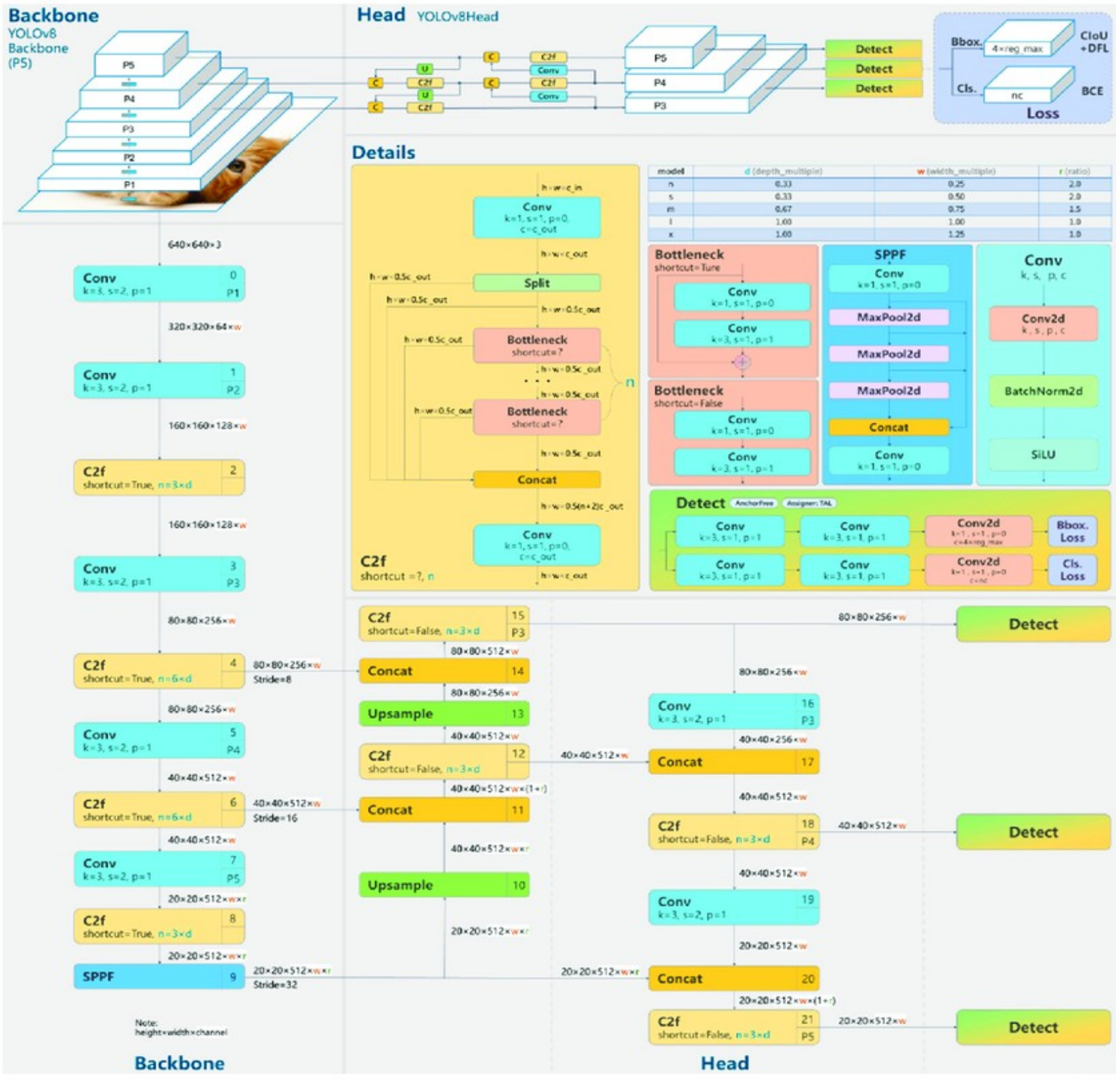
The second part is neck block wich used to enhance the feature extraction on the backbone block and some neglect it totally.

The neck Block consists of

- C2F block
- Concate layer
- Upsampling layer : to increase the feature map resolution to match the actual image resolution.

The last block is the head block and yolo v8 has 3 Detect Block one for small objects , one for medium objects and the last for large objects and also isolates the bounding box detection from class detection in each head.

This is the full architecture of the Yolo V8 model with the details of each block and sub blocks illustrated as layers .



2-YOLO V8 Algorithm :

Preprocessing:

- Read the input image.
- Resize the image to the model's input size.
- Normalise the pixel values (often done by subtracting the mean and dividing by the standard deviation).

Backbone Feature Extraction:

- Pass the preprocessed image through the backbone network (e.g., modified CSPDarknet53).
- The backbone extracts features from the image at different resolutions.

Neck (Path Aggregation):

- Combine feature maps from various stages of the backbone.
- (Possible) Apply Feature Aggregation modules that leverage spatial attention or context aggregation for improved feature representation.

Head (Prediction):

- Pass the processed features from the neck through a series of convolutional layers.
- Use prediction layers to estimate:
    - Bounding boxes for potential objects.
    - Confidence scores indicating the likelihood of a bounding box containing an object.
    - Class probabilities for the detected objects.

Post-processing (Non-Max Suppression):

- Apply Non-Max Suppression (NMS) to remove redundant bounding boxes with high overlap and low confidence scores.

Attached A high level pseudocode for the Algorithm.

```
function YOLOv8_ObjectDetection(image):
  # Preprocessing
  preprocessed_image = preprocess(image)

  # Backbone Feature Extraction
  features = backbone(preprocessed_image)

  # Neck (Path Aggregation)
  combined_features = path_aggregation(features)

  # Head (Prediction)
  predictions = head(combined_features)

  # Post-processing (Non-Max Suppression)
  final_detections = non_max_suppression(predictions)

  return final_detections
```

3-YOLO V8 Loss function :

In Yolov8 the loss consists of two parts:

1. classification branch, which utilises Binary Cross Entropy (BCE) Loss

2. regression branch, responsible for bounding box prediction, which uses a combination of two independent losses: Distribution Focal Loss (DFL) and Complete Intersection over Union (CioU) loss. DFL is intended to address the class imbalance problem in object detection tasks.

In Yolov8 it is also used to improve bounding box regression, especially for objects with blurry or unclear boundaries which are difficult to predict. Instead of predicting a fixed bounding box, DFL estimates the probability distribution of bounding boxes coordinates, and addresses uncertainties in their positions. CIoU loss, on the other hand, considers the aspect ratio differences between the predicted and ground truth boxes in addition to the overlap between them.

Combined, these two losses improve the regression task performance, especially when dealing with smaller objects.

The objectiveness loss, present in Yolov5, was dropped in YOLOv8. It determines how confident the model is about the presence of an object in the bounding box by contrasting the model's probability of an object being present with the ground truth value. In the new version 8 the model does not need a separate loss component as the objectiveness confidence is integrated into the IoU- Aware Classification Score.

The VarifocalLoss (VFL) is an alternative to DFL which was also designed to handle datasets with imbalanced class labels. Although its code is present in Yolov8's ultralytics/ultralytics/yolo/utils/loss.py module, VFL is not implemented and no longer maintained since it did not show any improvements over DFL.

In summary, Yolov8 uses BCE for classification and DFL and CIoU losses for bounding box regression. The final loss is a weighted sum of these three individual losses. The weights control the relative importance or contribution of each loss in the final combined loss that gets optimized during training.
These weights are equivalent to the following 3 hyper parameters in ultralytics/ultralytics/yolo/cfg/default.yaml:

4-YOLO V8 Activation function :

YOLOv8 also uses a new activation function, Mish, which provides improved accuracy and faster convergence compared to traditional activation functions like ReLU.

Definition:

The Mish function is a smooth, non-monotonic activation function defined as:

$$f(x) = x * \tanh(\ln(1 + \exp(x)))$$

Properties:

- **Smoothness:** Unlike ReLU (Rectified Linear Unit), which has a sharp kink at zero, Mish has a smooth curve, making it less prone to vanishing gradients during training.
- **Non-Monotonic:** Similar to Swish, Mish is non-monotonic, meaning its output can increase or decrease depending on the input value. This allows the network to learn more complex relationships in the data.
- **Bounded Output:** The output of Mish is always between -1 and 1, similar to the tanh function.

Benefits:

- **Improved Performance:** Studies have shown that Mish can lead to better performance compared to ReLU and Swish activation functions in various neural network architectures and tasks like image classification.
- **Faster Training:** The smoothness of Mish can potentially accelerate the training process of neural networks.
- **Self-Regularization:** Some researchers believe that the properties of Mish might contribute to a form of self-regularization, reducing the need for additional techniques to prevent over fitting.

2-Data pre-prossessing

1-Sign detection model dataset

As we mentioned in chapter 3 , The dataset we are using here contains many road sign types yet we chose only the Speed limits , stop and traffic lights.

This part that we have taken passes through pre-processing phases before training with YOLO v8.

1-image resizing :

The TT100K dataset does not have a specific size for image as it has images captured from different perspective and has signs that are zoomed in and others that are captured from a long distance and all the images differ in size so we resize all images to be to one size which is 704 * 704 which by experimental training and applying it the validation dataset gives better accuracy .

Also YOLO v8 considers input sizes that are multiple of 32 so we started at least image resolution we have then increasing by 32 as there is images that have important features we need to highlight during training until we reach 704.

2- Formatting :

The dataset was annotated according to pascal VOC format which has XML file contains the bounding boxes and classes , although Xml files are human readable and really good , Its parsing time is really long and we want to make it quicker for the training so we changed the format to YOLO format which has Text files instead of XML files and its parsing time much quicker that XML .

```
1 1 0.8383333333333334 0.8925 0.3233333333333333 0.20166666666666666
2 0 0.8391666666666666 0.7041666666666667 0.315 0.165
3 4 0.7925 0.8933333333333333 0.27166666666666667 0.06
4 2 0.8916666666666667 0.83 0.08333333333333333 0.14666666666666667
```

3- Data Ratio :

The ratio of dataset creation is 70 % for training , 15% for Validating and 15% for testing.
This is also confirmed by experimental training with different ratios.
The part we extracted from the original dataset was 4969 image , 3531 for training ,639 for Testing and 802 for validating.

2- Crack detection model dataset pre-processing

As we mentioned in chapter 3 in the Analysis part of the used dataset .This dataset is captured in japan streets and its details mentioned in chapter 3 (Analysis and design).

We also have applied some pre-processing steps to the dataset before training it with the YOLO v8 model.

1- Image resizing :

Important to note , The images in this dataset was all resized to 600*600.This resolution has to be changed as 600 it's not a multiple of 32 and this is a constraint for YOLO V8 input size, We also could not go lower that because detecting cracks is so sensitive task and we have to consider every feature in the image so we went for higher and by experimental training we considered 800*800 as it gives the best results.

2-Data Augmentation :

Dealing with crack types is way too different from signs as all the crack types are similar and the differences are very close.

For that reason , we needed to apply data augmentation to increase the dataset with more features that cover all the possibilities of the pattern of each class so the model could distinguish between crack types more easily.

Types of data Augmentation Applied :

1. Rotation range = 20 degree
2. Width shift range = 0.1
3. Hight shift range = 0.1
4. Zoom range = 0.2
5. Horizontal flip

We have divided the whole dataset into sections then each section one type of data augmentation applied to it then all merged together to made new 4761 images

3- Formatting :

This dataset was also annotated by Pascal VOC format by XML file for each image and for the same reasons we mentioned earlier in the sign model part we changed it also to YOLO format with
Text file instead for each image.

4-Dataset Ratio :

The dataset splitting ratio here is 80 % for training 20 % validating. The training sample is about 7769 images and 1942 validation .
The ratio is also obtained from experimental training.

3- Tools and Frameworks :

1-Deep learning part :

1-YOLO v8 mode :

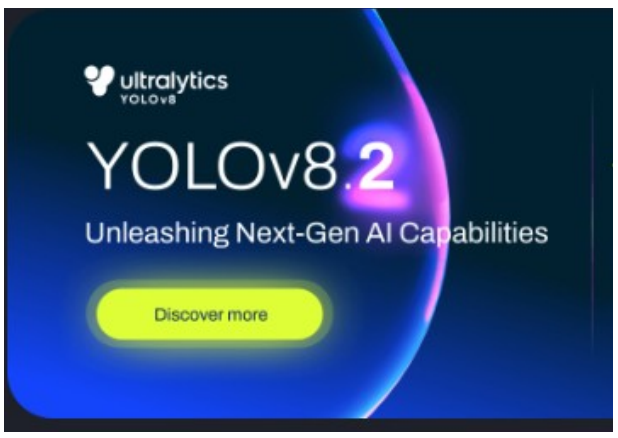
YOLO v8 model here implemented using the ultralytics package containing all the other needed packages.

Here is the documentation to check it out : https://docs.ultralytics.com/

We chose to use the training from scratch method , but there are also many pre-trained models to work with.

2-Dataset pre-processing :

All the dataset preprocessing tasks formating , resizing , data Augmentation, even the analysis part were implemented using python 3.12.0 and many python packages . Especially Tensorflow and Matplotlib for visualisation.



2-Mapping Part :

Map implemented using ArcGIS and QGIS technologies and deployed using on Andriod Application using kotlin native.

3-Application Part :

Back end and Front end **:** Both implemented using Kotlin native on Andriod studio with gradle environment for dependency and implementation.

Database : The database used for the registration part and crack reporting part are both NOSQL database real time Firebase .

4-Deep learning models integration with Map :

We have reached the point where we have our Deep learning models ready (The crack and sign detection models check ch3 and earlier in this chapter for further information about them). We also have our map ready (check section 2 in this chapter for more information about how we implemented it).

We want to make use of our components. What we are about to do is that we want to integrate our models with the map to allocate points where cracks and signs are located actually in actual streets.

We followed crucial steps to achieve model/Map integration.

1) Scanning :

Scanning is all about filming a video for a road or a region that have road cracks using the same method the dataset which the model trained on gathered (mounted vehicle using smart phone method)

We scanned a region (Helwan University region) containing cracks with the types we trained our model on.

The installation setup of the smartphone in the car is shown in the figure.

the result of this step is a video fiming the whole road (preferred to be one shot).

Factors affecting the filming:

1- sun position :

The filming was taken in the morning from 9 Am to 11 Am. In the morning, the sun is lower in the sky, which can create a more pleasing angle of light. In the afternoon, the sun is higher, leading to less flattering lighting angles and potential overexposure.

2-The phone angle position :

The phone has to cover the road ahead as shown in the figure to cover all the road cracks.

3- Speed :

Car speed influences the filming quality. We need car speed to be compatible with filming so I can have clear video frames and input for my model (We used 40 Km/hr).

We used the iPhone 11 for filming. It can capture 60 frames/second. The resulting video was 8.07 minutes resulting in 29220 frames.

Images are captured at a resolution of 828x1792 and later resized to 600x600 to maintain uniformity with the dataset the model trained on.

Besides this video another software tool used called Quick capture its a mobile application that record points while driving (as you go it records points) this points is represented on a map which we will use later on in our app .

The benefit of using this software is to recorded the points I passed through filming as it saves a time stamp to each point which will help me match this points time stamps and the video frames detected.

2) Creating CSV File :

with the help of the resulted time stamp from Quick capture and the video we have now we will create a csv file that match every point with its video frame.

First, The video added to the model to be predicted, Yolo V8 predict video by exporting its frames and detect on each frame if there object here or not and what is that object , so the result from this step is a csv file contains each frame and the detection and detection type for it .

The output of model will be as follow specifies the detection of each frame.

```
video 1/1 (frame 60/11389) /content/drive/MyDrive/VID_20240221_141029.mp4: 640x640 (no detections), 1910.9ms
video 1/1 (frame 61/11389) /content/drive/MyDrive/VID_20240221_141029.mp4: 640x640 1 D00, 1588.7ms
video 1/1 (frame 62/11389) /content/drive/MyDrive/VID_20240221_141029.mp4: 640x640 1 D00, 1612.8ms
video 1/1 (frame 63/11389) /content/drive/MyDrive/VID_20240221_141029.mp4: 640x640 1 D00, 1599.3ms
video 1/1 (frame 64/11389) /content/drive/MyDrive/VID_20240221_141029.mp4: 640x640 1 D20, 1 D00, 1633.2ms
video 1/1 (frame 65/11389) /content/drive/MyDrive/VID_20240221_141029.mp4: 640x640 1 D20, 1601.0ms
```

Second step , we are going to match the first time stamp of a point detected by Quick Capture and first fram to create a csv file with Time stamp , frame number and detection type .

This CSV file that we will use to import in Map.

Important notes :

As we mentioned earlier that the device we used for filming capture 60 frame /second so according to the video length we have 29220 frame but Quick Capture record by second so we need to set for each 1 second time stamp 60 frame.

One important thing to notice that many frames could be with no detection so we can filter some of them to keep balance between map points and frames detected by keeping the important ones .

Using excel we created the CSV file needed by data from two components.

This is a sample of what its look like.

1	Date	Time	Timestamp	Frame number	Detection
2	3/10/2024	11:32:46	3/10/2024 11:32:46	1/29204)	(no detections)
3	3/10/2024	11:32:46	3/10/2024 11:32:46	2/29204)	(no detections)
4	3/10/2024	11:32:46	3/10/2024 11:32:46	3/29204)	1 D20
5	3/10/2024	11:32:46	3/10/2024 11:32:46	4/29204)	(no detections)
6	3/10/2024	11:32:46	3/10/2024 11:32:46	5/29204)	(no detections)
7	3/10/2024	11:32:46	3/10/2024 11:32:46	6/29204)	1 D20
8	3/10/2024	11:32:46	3/10/2024 11:32:46	7/29204)	1 D20
9	3/10/2024	11:32:46	3/10/2024 11:32:46	8/29204)	(no detections)
10	3/10/2024	11:32:46	3/10/2024 11:32:46	9/29204)	(no detections)
11	3/10/2024	11:32:46	3/10/2024 11:32:46	10/29204)	(no detections)
12	3/10/2024	11:32:46	3/10/2024 11:32:46	11/29204)	1 D20
13	3/10/2024	11:32:46	3/10/2024 11:32:46	12/29204)	(no detections)

5-Deep learning crack model deployment

Here we implemented a feature to allow users to report the new cracks they could face and not located on our map. So , This feature allows the user to capture a crack image and image location taken at the same time , then send this image to the server which has the model for prediction.If the prediction result has cracks it will be saved in a database with its type and location .The implementation of this feature is implemented using two important stages .

First : Model deployment server

We implemented a python server using the Flask python Web framework .
This part of code takes the image in the form of an array of bits as an input , reconstructs the image and then passes it to the model for prediction.The response JSON file contains detected classes types , Bounding box coordinates for each class and the confidence for each one.

Second : Android mobile Application :

Front-End part : The front-end part consists of two layouts , One for using mobile camera to capture image and confirm user location and the other to receive the response from the Flask server to and view image with the detected bounding boxes and controller to send this data to the database for saving.

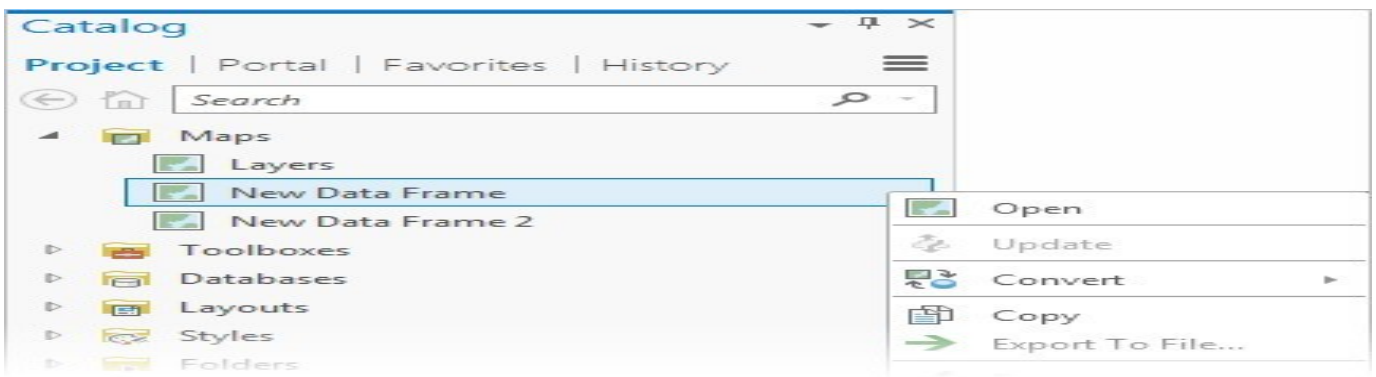
Backend : The back-end serving the process divided between the two layouts , First one access user camera , resize the image captured to reduce request/response time , access user location and send image to the flask server , another part takes the responsibility from the response draw bounding boxes and dealing with database.

The database used here is NOSQL database realTime firebase, the reference contains three values: cracks types , longitude and latitude.

6-Mapping implementation

1. Visualizing Road Cracks and Road Signs

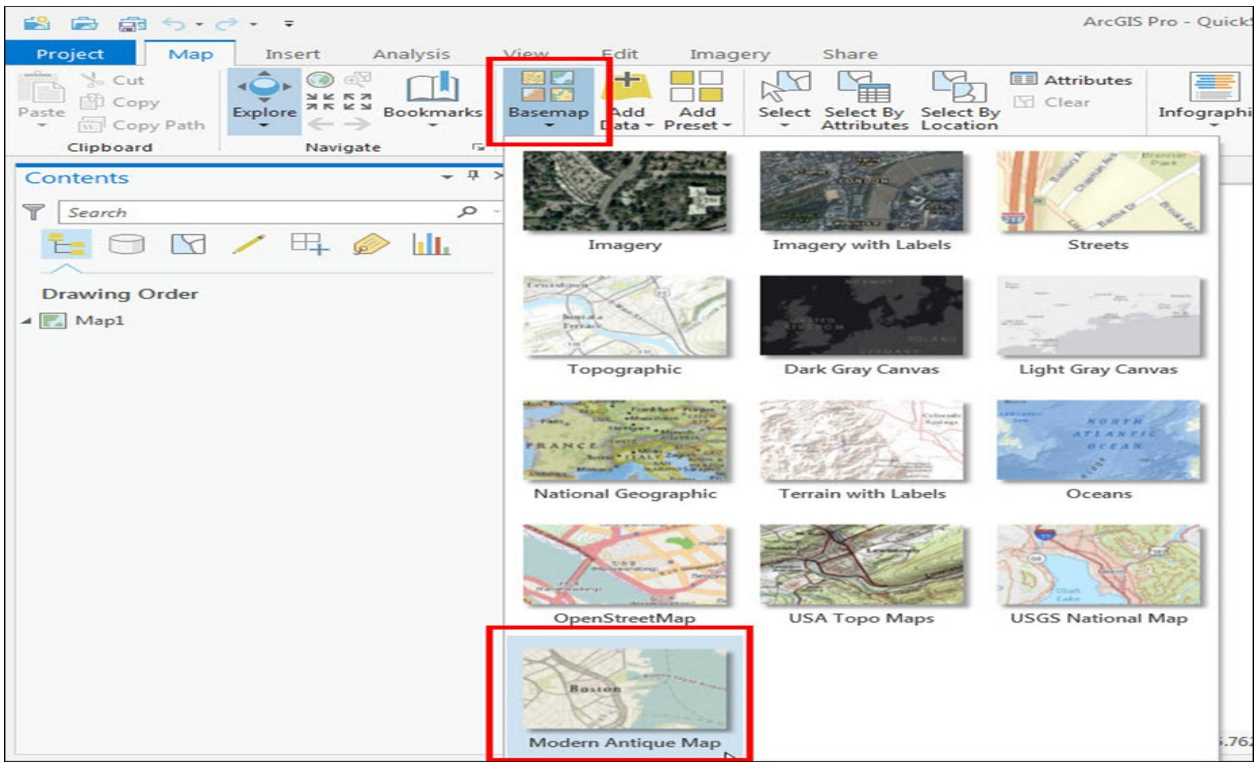
- **Description:**ArcGIS software can be used to visualize road cracks and signs from high-resolution images captured with cameras or drones. After image acquisition, optional marking of features on the images can be done for easier data import. The images are then imported into ArcGIS, and separate feature layers are created for cracks and signs. These layers involve georeferencing (if needed), digitizing crack and sign locations as points, and assigning symbology and labels for clear visualization.
- **Tools:** Use GIS software “ArcGISPro“, search engines and data repositories “Chrome, Kaggle, etc.….”
- **Steps:**
  - Collect high-resolution images of road surfaces using cameras or drones.
  - Mark detected features on the images.
  - Import processed images into GIS software.
  - Create a layer in the GIS software to visualize detected road cracks and signs.



2. Map Construction

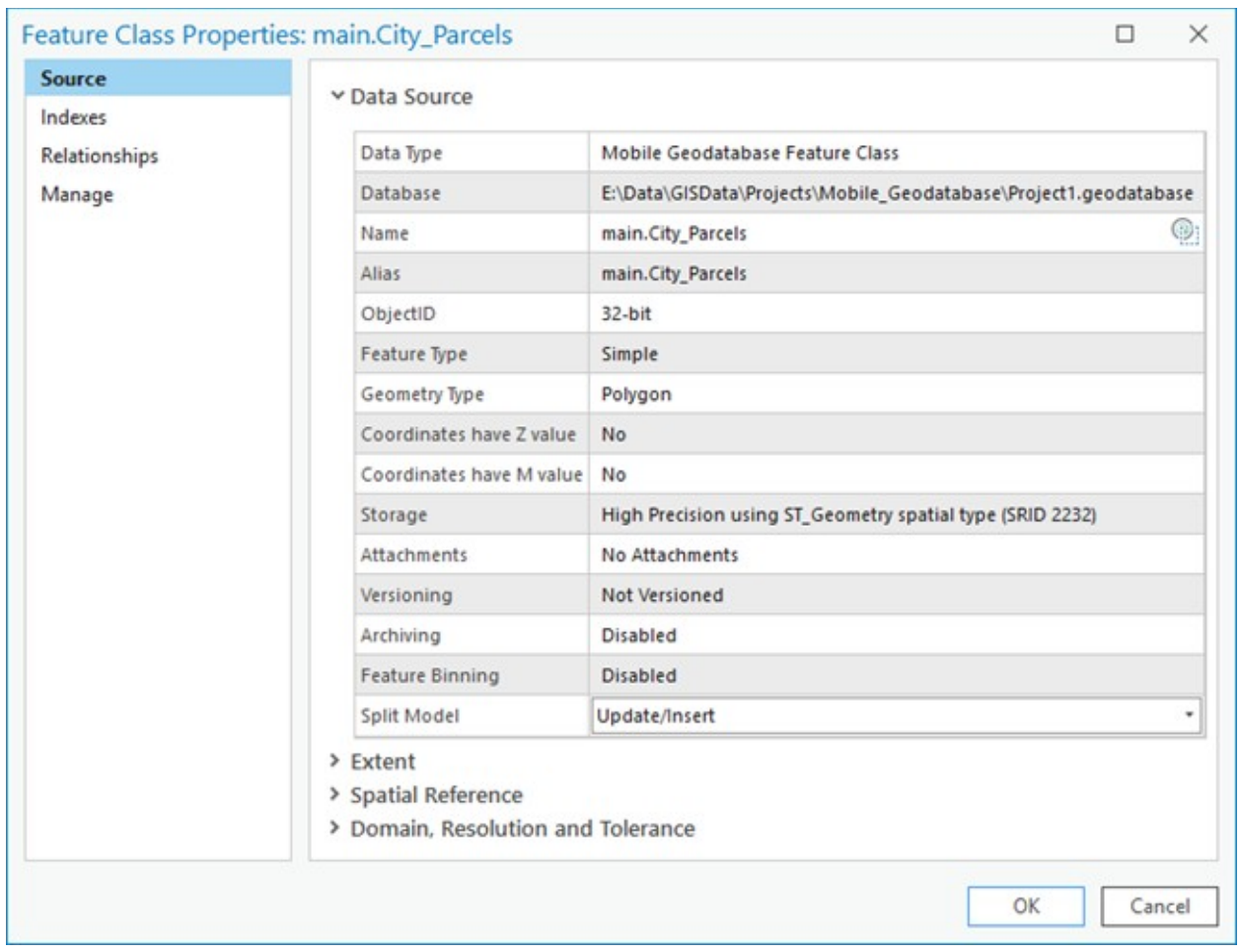
- **Description:** ArcGIS (or other GIS software) is used to construct a base map for visualizing road cracks and signs. The process starts with creating a new project and importing essential layers like satellite imagery, road networks, and administrative boundaries. Then, the map schema is designed, outlining the layers and their attributes. Finally, the map layout and symbology are customized to enhance readability and emphasize key features. This initial map setup is then saved for further development.

- **Tools:** GIS software“ArcGIS Pro”.
- **Steps:**
  - Open GIS software and create a new project.
  - Import base layers such as satellite imagery, road networks, and administrative boundaries.
  - Design the map schema, defining necessary layers and attributes.
  - Customize the map layout and symbology to highlight key features.
  - Save the initial map setup.



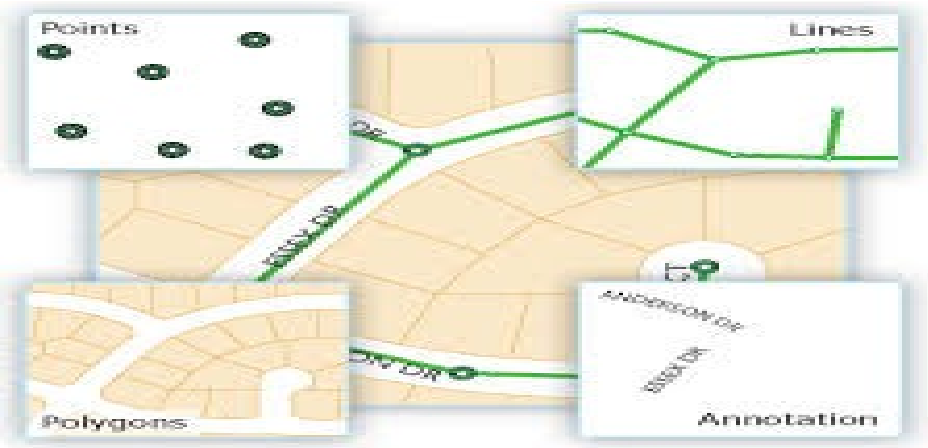
3. Geodatabase Schema

- **Description:**ArcGIS (or other GIS software) is used to design a geodatabase schema for storing road crack and sign data efficiently. This involves defining tables with specific fields tailored for each element. For road cracks, this includes ID, location coordinates, condition, and date detected. Signs would have similar fields but also include sign type and optionally, an image field. Finally, separate feature classes, like "RoadCracks" and "RoadSigns", are created within the geodatabase to store the actual data records based on the defined schema.
- **Tools:** GIS software “ArcGIS Pro”.
- **Steps:**
  - Define tables and fields required for storing geospatial data.
  - For Road Cracks: Fields like ID, Location (coordinates), Condition, Date_Detected.
  - For Road Signs: Fields like ID, Location, Sign_Type, Date_Detected.Sign_image
  - Create standalone feature class for both Cracks and Signs
  - Implement the schema in the GIS software’s geodatabase.



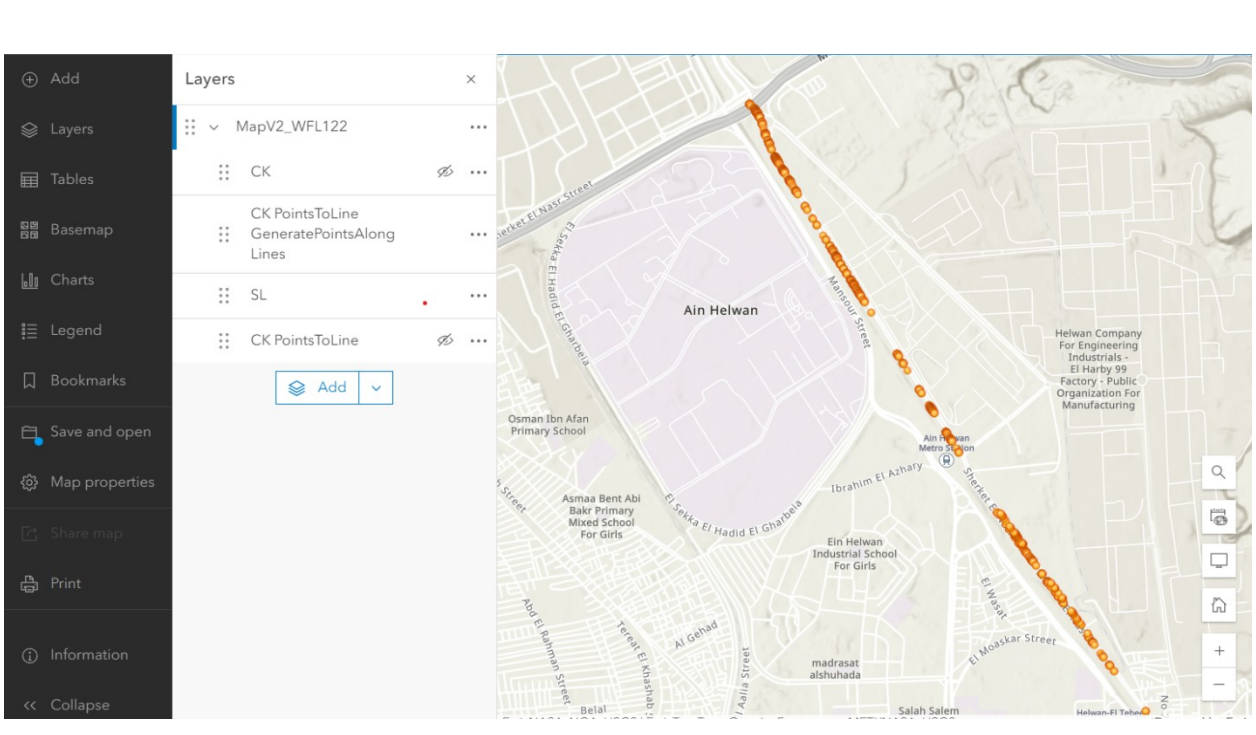
4. Feature Class Creation

- **Description:** ArcGIS Pro (or similar GIS software) is used to create separate feature classes for road cracks and signs within the existing geodatabase. This involves first accessing the geodatabase and then creating a new feature class for each element, "RoadCracks" and "RoadSigns." Each class definition specifies the appropriate geometry type (point, line, or polygon) depending on the crack or sign shape. Finally, attributes defined in the geodatabase schema, such as ID, location, and details specific to cracks or signs, are added to their respective feature classes.
- **Tools:** GIS software.
- **Steps:**
  - Open the geodatabase in the ArcGIS pro.
  - Create a new feature class for road cracks.
  - Define the geometry type (e.g., point, line, polygon) based on the crack feature.
  - Create a new feature class for road signs.
  - Define the geometry type (e.g., point, line, polygon) based on the sign feature.
  - Add attributes as per the schema.
  - Repeat for road signs.



5. Feature Layers Creation and Combination

- **Description:***ArcGIS software* is used to create feature layers from the previously defined feature classes for road cracks and signs. These layers can then be populated with initial data (crack locations, sign types) if available. Each layer's symbology is then customized to visually differentiate between various attributes, such as using different colors for different crack severities. Finally, these informative feature layers are saved within the project for further analysis and visualization
- **Tools:**ArcGIS pro, ArcGIS cloud service and portal.
- **Steps:**
  - Publish the created feature class on the cloud service
  - Populate these layers with initial data if available.
  - Set symbology for each layer to differentiate between various attributes (e.g., crack severity).
  - Save the feature layers.
  - Open the project containing the Road_Cracks and Road_Signs layers.
  - Add both layers to a single map.
  - Ensure proper alignment and overlay of the layers.
  - Configure layer properties for optimal visualization and interaction.
  - Choose the compatible symbology
  - Save the combined map.

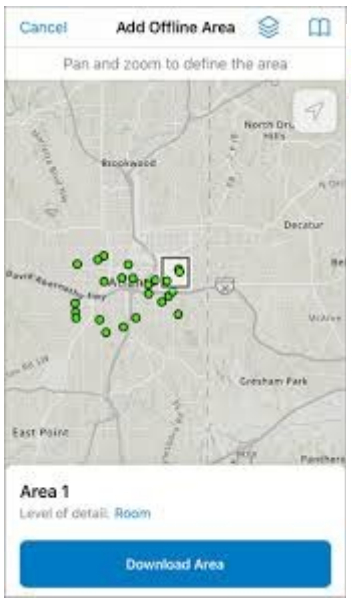


7. Publishing Web Map

- **Tools:**ArcGIS Online, QGIS Cloud.
- **Steps:**
  - Export the combined map from the GIS software.
  - Sign in to the chosen cloud service (ArcGIS Online, QGIS Cloud).
  - Upload the map and associated data layers.
  - Configure sharing settings to make the map accessible online.
  - Publish the map and test its accessibility.

8. Feature Addition Methodology

- **Description:***Keeping Your Data Fresh*To ensure the GIS database stays current, a methodology for adding new road crack and sign data is established. This involves defining data collection protocols (how often, who collects it) and training field crews on using ArcGIS tools like QuickCapture and Field Maps to capture data in real-time. Standardized data entry procedures and quality checks guarantee data accuracy. A workflow is then set up to seamlessly sync this field data with the central database. Regularly reviewing and updating this methodology keeps the data collection and integration process efficient and optimized.
- **Tools:**QuickCapture, Field Maps.
- **Steps:**
  - Define protocols for data collection (e.g., frequency, responsible teams).
  - Train field teams on using QuickCapture and Field Maps.
  - Establish data entry standards and quality checks.
  - Implement a workflow for syncing field data with the central database.
  - Regularly review and update the methodology as needed



9. Survey Typeand QuickCapture

- **Description:**Efficient Data Collection with QuickCapture, Esri's QuickCapture app is adopted for its efficiency in gathering new road crack and sign data. Resource limitations (budget, personnel, equipment) are considered during the planning phase, where survey routes, schedules, and device readiness are determined. Field crews receive comprehensive training on using the app and safety protocols. The QuickCapture project is configured to capture essential attributes like location, type, and condition for each element. Field teams then utilize the app to record real-time geospatial data, syncing it with cloud storage for further processing. Data accuracy is ensured through monitoring and validation measures.
- **Tools:**Esri'sQuickCapture.
- **Steps:**
    1. Evaluate available resources (budget, personnel, equipment).
    2. Select a mobile survey method using QuickCapture for its efficiency.
    3. Plan the survey routes and schedule.
    4. Ensure necessary devices and software are ready and configured.
    5. Brief survey teams on procedures and safety.
    6. Configure QuickCapture project to capture required attributes (location, type, condition).
    7. Distribute devices with QuickCapture installed to field teams.
    8. Field teams use QuickCapture to record geospatial data in real-time.
    9. Sync collected data to cloud storage.
    10. Monitor and validate incoming data for accuracy.

10. Video Recording

- **Description:**Enhancing Data with Video To capture additional details, field teams can utilize mobile devices or dash cameras to record continuous video footage while

conducting road surveys. Timestamping and geotagging the video ensures alignment with the recorded features (cracks, signs) for later analysis. These video files are stored securely for further processing, with video quality maintained to facilitate effective analysis.

- **Tools:** Mobile devices, dash cameras.
- **Steps:**
  - Equip field teams with mobile devices or dash cameras.
  - Record continuous video footage while conducting the survey.
  - Ensure the video is timestamped and geotagged for alignment with recorded features.
  - Store video files securely for processing.
  - Ensure video quality is sufficient for analysis.

11. ML Model Integration

- **Description:**Leveraging Machine Learning For automated anomaly detection, pre-trained or custom-built Machine Learning (ML) models can be integrated into the workflow. Recorded video footage from road surveys is first divided into individual frames. Each frame is then processed by the ML models to identify potential road anomalies (cracks, damaged signs). The data on these detected anomalies, including corresponding frame numbers and geospatial coordinates, is collected. Finally, the model's outputs are validated by comparing them to manually annotated samples to ensure accuracy.
- **Tools:** TensorFlow, PyTorch.
- **Steps:**
  - Develop or use pre-trained ML models for anomaly detection.
  - Extract frames from the recorded videos.
  - Process each frame using ML models to detect road anomalies.
  - Collect detected anomalies data with corresponding frame numbers and coordinates.
  - Validate the model outputs against sample manual annotations

12. CSV File for Anomalies

- **Description:**Anomaly Data in CSV FormatThe Machine Learning (ML) model's output on detected road anomalies is transformed into a structured format for further analysis. This involves compiling data on each anomaly, including its type (crack, damaged sign), location coordinates, corresponding frame number from the video footage, and a timestamp. This compiled data is then exported into a CSV file, a common and user-friendly format. Finally, the CSV file is reviewed to ensure all necessary information is captured accurately and completely.

- **Tools:** ML framework, CSV processing tools.
- **Steps:**
  - Compile detected anomalies into a structured format.
  - Include fields like anomaly type, location (coordinates), frame number, and timestamp.
  - Export this data to a CSV file.
  - Review the CSV file for completeness and accuracy.
  - Store the CSV file for further integration with GIS.

13. Feature Matching

- **Description:**Reconciling Data for Accuracy To ensure precise feature location, features identified by the Machine Learning model in the CSV file (anomaly type, location, frame number) are carefully aligned with the corresponding frames in the recorded video footage using timestamps or frame numbers. This aligned data is then imported into the GIS software. The detected anomalies from the video are then correlated with the features (cracks, signs) previously recorded during the survey. By comparing video evidence, feature positions within the GIS software might be adjusted to achieve the most accurate representation. Finally, the accuracy of these matched features is confirmed to guarantee a reliable dataset for further analysis.
- **Tools:**ArcGIS pro, video processing tools.
- **Steps:**
  - Align features in the CSV file with video frames using timestamps or frame numbers.
  - Import the CSV data into the GIS software.
  - Correlate detected anomalies with the recorded survey features.
  - Adjust feature positions as necessary based on video evidence.
  - Confirm the accuracy of the matched features.

14. Feature Filtering

- **Description:**Refining Feature Data with ArcGIS Pro ArcGIS Pro is used to refine the feature data (road cracks, signs) by comparing them with the anomalies detected by the Machine Learning model. Features that weren't identified by the models are filtered out, resulting in separate lists for detected and undetected features. Discrepancies are then analyzed to pinpoint potential issues or errors in the data or the model itself. Finally, the feature layers within the GIS software are updated based on the filtered results, ensuring a more accurate and comprehensive dataset.
- **Tools:**ArcGIS pro.
- **Steps:**
  - Compare detected anomalies with manually recorded features.

- Filter out features that were not detected by the ML models.
- Create separate lists for detected and undetected features.
- Analyze discrepancies to identify potential issues or errors.
- Update the feature layers based on the filtered results.

15. New Feature Layer Creation

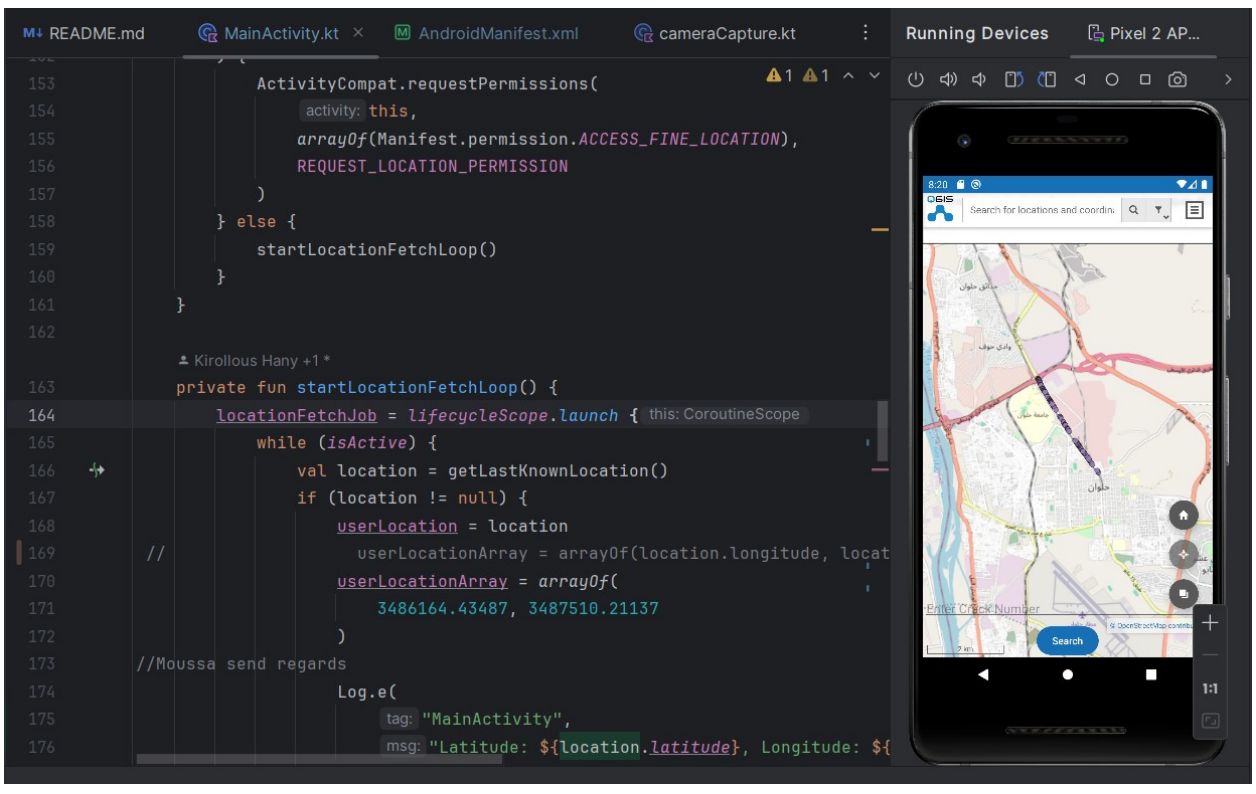
- **Description:**Finalizing the Data ArcGIS software (or similar GIS software) is used to create a new feature layer that incorporates the verified and potentially modified features (cracks, signs) identified during the workflow. This layer is then visually enhanced with clear symbology and labeling for easy interpretation. The final validation step involves comparing the new layer against the original data and video evidence to ensure accuracy. Once validated, the new feature layer is saved and documented for further analysis and utilization within the GIS project.
- **Tools:** GIS software.
- **Steps:**
  - Create a new feature layer in the GIS software.
  - Populate this layer with verified and modified features.
  - Set appropriate symbology and labeling for clarity.
  - Validate the new layer against original data and video evidence.
  - Save and document the new feature layer.

16. Publishing with QGIS

- **Description:**Sharing the Results Once the final feature layer containing verified road crack and sign data is created in the GIS software, it can be shared with a wider audience. This involves exporting the layer from the software and uploading it to a web-based platform that allows for online map creation. After configuring project settings and access permissions, the web map is published, making it accessible to authorized users online for further analysis or visualization purposes.
- **Tools:** QGIS Cloud.
- **Steps:**
  - Export the new feature layer from the GIS software.
  - Log in to QGIS Cloud and create a new project.
  - Upload the new feature layer to the QGIS Cloud project.
  - Configure the project settings and permissions for accessibility.
  - Publish the web map and ensure it is accessible online.

7) Map deployment

- **Description:**Mobile Access for Enhanced Functionality For improved data accessibility and interaction, an Android application can be developed or updated to integrate with the published web map containing road crack and sign data. APIs (Application Programming Interfaces) facilitate communication between the app and the web map, allowing the app to display the map data. The app can be further enhanced with features for real-time data updates and offline access functionality. User-friendly navigation and intuitive interaction capabilities are crucial for a seamless user experience. Following thorough testing, the app can be deployed for users to access and interact with the road condition data conveniently on their mobile devices.
- **Tools:** APIs, mobile development tools.
- **Steps:**
    1. Develop or update an Android application to interact with the web map.
    2. Use APIs to fetch and display the web map within the app.
    3. Implement features for real-time data updates and offline access.
    4. Ensure the app has user-friendly navigation and interaction capabilities.
    5. Test the app thoroughly and deploy it to users.



Conclusion: Building a Comprehensive Road Condition Mapping System

This document outlines a comprehensive workflow for creating and maintaining a detailed map of road cracks and signs using GIS technology and Machine Learning (ML).

Data Preparation and Visualization (1-5):

The process begins with **visualizing existing cracks and signs** from high-resolution imagery (1). ArcGIS software is used to create a base map with essential layers like satellite imagery and road networks (2). A geodatabase schema is then designed to efficiently store data on identified cracks and signs, including attributes like location and condition (3). Separate feature classes for cracks and signs are created within the geodatabase (4). Finally, feature layers are generated from these classes and populated with initial data if available. Symbology is then customized to visually differentiate features (5).

Efficient Data Collection and Integration (7-14):

To keep the data current, a methodology is established for adding new data points (8). This involves using mobile survey tools like QuickCapture (9) to collect real-time geospatial data while capturing video footage for additional details (10). ML models can be integrated to automate anomaly detection in the video frames, identifying potential road cracks and damaged signs (11). The detected anomalies are then compiled into a structured format like a CSV file for further analysis (12). This data is carefully aligned with the video footage using timestamps to ensure precise feature location within the GIS software (13). Finally, ArcGIS Pro is used to compare the ML-identified features with manually recorded data, refining the feature data and ensuring a comprehensive dataset (14).

Data Sharing and Accessibility (15-17):

A new feature layer incorporating the verified and potentially modified features is created (15). This layer is visually enhanced and validated before being saved and documented. To share the results with a wider audience, the final feature layer is exported and uploaded to a web-based platform, allowing for online map creation and publishing (16). Furthermore, an Android application can be developed or updated to integrate with the published web map, enabling mobile access and interaction with the road condition data (17).

This comprehensive approach leverages GIS, Machine Learning, and mobile technologies to create and maintain an accurate and up-to-date map of road cracks and signs. This information can be crucial for road maintenance planning, ensuring safer and smoother journeys.

8-Data Retrieving Process

1. **Setup the Network Client**:
   - **HTTP Client Configuration**: The application sets up an HTTP client with specific timeout settings. These settings ensure that the client waits for a reasonable amount of time for responses and does not hang indefinitely if the server is slow.
   - **Retrofit Initialization**: Retrofit, a type-safe HTTP client for Android, is initialized with the base URL of the remote server. Retrofit simplifies network requests by converting the server response into Java/Kotlin objects.

2. **Define API Endpoints**:
   - **ApiCalls Interface**: An interface is defined with methods representing the API endpoints. Each method corresponds to an HTTP request, annotated with details like the HTTP method (GET), endpoint path, and query parameters.

3. **Make the API Call**:
   - **Coroutine Execution**: A coroutine is launched to make the network request asynchronously. This prevents the network operation from blocking the main thread, ensuring a smooth user experience.
   - **Request Execution**: The defined API endpoint method is called, sending a request to the server to fetch feature data (e.g., geographic points).

4. **Handle the Response**:
   - **Response Validation**: The server's response is checked for a successful status code (e.g., 200 OK). If the request is successful, the response body containing the feature data is processed.
   - **Data Extraction**: The feature data, which includes geographic coordinates, is extracted from the response. This data is converted into a format suitable for further processing (e.g., a 2D array of longitude and latitude values).

```
class MainActivity : AppCompatActivity() {

    private val qgisCloudMapUrl =
        "https://qgiscloud.com/Moussa03/modifiedGPmap/?l=modifiedCracksLayes&bl=mapnik&t=modifiedGPmap&e=3483012%2C3484240%2C3491140%2C3487960"
    private val baseUrl = "https://qgiscloud.com/Moussa03/modifiedGPmap/"
    private lateinit var fusedLocationClient: FusedLocationProviderClient
    private var userLocation: Location? = null
    private lateinit var pointsArray: Array<Array<Double>>
    private lateinit var userLocationArray: Array<Double>
    private var locationFetchJob: Job? = null
    private val locationFetchInterval = 10000L // 10 seconds
    private var mediaPlayer: MediaPlayer? = null
```

```
package com.example.mapping

> import ...

Abdel-Rahman
interface ApiCalls {
    Abdel-Rahman
    @GET("wms")
    suspend fun getFeature(
        @Query("SERVICE")SERVICE:String="WMS",
        @Query("VERSION")VERSION:String="1.3.0",
        @Query("REQUEST")REQUEST:String="GetFeatureInfo",
        @Query("LAYERS")LAYERS:String="modifiedCracksLayes",
        @Query("QUERY_LAYERS")QUERY_LAYERS:String="modifiedCracksLayes",
        @Query("INFO_FORMAT")INFO_FORMAT:String="application/json",
        @Query("CRS")CRS:String="EPSG:4326",
        @Query("feature_count") feature_count: String= "99999",
        @Query("I") I: String= "0",
        @Query("J") J: String= "0",


        ): retrofit2.Response<FeatureResponseModel>
}
```

9-Alarming Process

1. **Fetch User Location**:
   - **Location Services Client**: The application uses a location services client to access the device's current location. This requires permission from the user to access location data.
   - **Permission Handling**: The application checks if location permissions are granted. If not, it requests the necessary permissions from the user.

```
147         private fun checkLocationPermissionsAndFetchLocation() {
148             if (ContextCompat.checkSelfPermission(
149                     context: this,
150                     Manifest.permission.ACCESS_FINE_LOCATION
151                 ) != PackageManager.PERMISSION_GRANTED
152             ) {
153                 ActivityCompat.requestPermissions(
154                     activity: this,
155                     arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
156                     REQUEST_LOCATION_PERMISSION
157                 )
158             } else {
159                 startLocationFetchLoop()
160             }
161         }
162
```

2. **Continuous Location Monitoring**:
   - **Regular Updates**: The application sets up a loop to fetch the user's location at regular intervals (e.g., every 10 seconds). This ensures the application has the most recent location data.
   - **Non-blocking Operations**: Location fetching is done asynchronously within a coroutine, so the main thread remains responsive.

```
163         private fun startLocationFetchLoop() {
164             locationFetchJob = lifecycleScope.launch { this: CoroutineScope
165                 while (isActive) {
166                     val location = getLastKnownLocation()
167                     if (location != null) {
168                         userLocation = location
169                         //        userLocationArray = arrayOf(location.longitude, location.latitude)
170                         userLocationArray = arrayOf(
171                             3486164.43487, 3487510.21137
172                         )
173 //Moussa send regards
174                         Log.e(
175                             tag: "MainActivity",
176                             msg: "Latitude: ${location.latitude}, Longitude: ${location.longitude}"
177                         )
178                         checkProximityToPoints()
179                     } else {
180                         Log.e( tag: "MainActivity",  msg: "Failed to get location.")
181                     }
182
183                     delay(locationFetchInterval)
184                 }
185             }
186         }
```

3. **Check Proximity**:
   - **Distance Calculation**: For each location update, the application calculates the distance between the user's current location and each predefined point. This is typically done using the Haversine formula, which computes the great-circle distance between two points on the Earth.

- **Proximity Threshold**: The application checks if the calculated distance is within a specified threshold (e.g., 500 meters). If the user is within this range of any point, the application decides to trigger an alarm.

```
209     private fun checkProximityToPoints() {
210         var location = 0
211         for (point in pointsArray) {
212             val distance = calculateDistance(
213                 userLocationArray[1], userLocationArray[0],
214                 point[1], point[0]
215             )
216             if (distance <= 500) {
217                 triggerAlarm()
218                 break
219             } else {
220                 Log.e( tag: "MainActivity",  msg: "ALARM TRIGGERED")
221                 location++
222             }
223         }
224     }
225
```

```
226     private fun calculateDistance(lat1: Double, lon1: Double, lat2: Double, lon2: Double): Double {
227         val earthRadius = 6371000 // meters
228         val dLat = Math.toRadians(lat2 - lat1)
229         val dLon = Math.toRadians(lon2 - lon1)
230         val a = sin( x: dLat / 2) * sin( x: dLat / 2) +
231                 cos(Math.toRadians(lat1)) * cos(Math.toRadians(lat2)) *
232                 sin( x: dLon / 2) * sin( x: dLon / 2)
233         val c = 2 * atan2(sqrt(a), sqrt( x: 1 - a))
234         return earthRadius * c
235     }
```

4. **Trigger Alarm**:
   - **Alarm Activation**: If proximity criteria are met, an alarm is triggered. This usually involves playing a pre-defined alarm sound to alert the user.
   - **Media Player Handling**: A media player instance is used to play the alarm sound. The sound is played for a fixed duration (e.g., 5 seconds) and then stopped. The media player is properly managed to release resources after use to prevent memory leaks.

```
237     private fun triggerAlarm() {
238         mediaPlayer?.stop()
239         mediaPlayer?.release()
240         mediaPlayer = MediaPlayer.create( context: this, R.raw.alarm_sound)
241         mediaPlayer?.start()
242
243         // Stop the sound after 5 seconds
244         lifecycleScope.launch {  this: CoroutineScope
245             delay( timeMillis: 5000)
246             mediaPlayer?.stop()
247             mediaPlayer?.release()
248             mediaPlayer = null
249         }
250
251     }
```

Detailed Process Flow

1. **Application Launch**:
   - On launching, the application initializes the necessary views and components, such as the web view for displaying maps and the location services client for fetching user location.

2. **User Interaction**:
   - The user may interact with the application by, for example, entering a search term and pressing a search button. This triggers data fetching and location monitoring processes.

3. **Data Fetching**:
   - The application makes an API call to fetch feature data (e.g., locations of interest) from a remote server. The retrieved data is processed and stored for further use.

4. **Location Monitoring**:
   - The application continuously fetches the user's current location at regular intervals. Each location update is compared against the predefined points to check for proximity.

5. **Alarm Triggering**:
   - If the user comes within the specified range of any predefined point, the application triggers an alarm to alert the user. The alarm sound plays for a fixed duration and then stops.

4.2) Experimental /Simulation setup

1- Crack detection model

1-Data description : The datasets our model trained and validated in were publicly available in the reference paper we mentioned earlier in this chapter.
Yet, It needed a lot of pre-processing as we mentioned in this chapter.
The Japan dataset is used for training the model as it contains more images and and also Augmented but for the validation we used the India dataset as it has much more images contains D40 and D20 crack types
2-Deep learning model details : The model used is YOLO v8 , we trained it from scratch fine tuning just the last layer of the detection .
3-Training parameters :

- Optimizer used : Adam
- Learning rate : initial by 0.001 and final by 0.01
- Batch size : 8
- Number of training epochs : 50 epoch
- Image size : 704 *704
- dropout: 0.0
- iou: 0.7
- box: 7.5
- cls: 0.5
- dfl: 1.5

4-Hardware and software environment :
This model trained on CPU 8 cores , using only the ultralytices package ,it takes total time 12.5 hours for training .

1- Sign detection model

1-Data description : The dataset used here is extracted from famous public available TT100k dataset , it applied to pre-processing techniques and the training and validation was from same dataset by ratio .
2-Deep learning model details : The model used is YOLO v8 , we trained it using the pre-trained model YOLO v8 L .
This model has large weights that gives better accuracy in fast time and it
3-Training parameters :

- Optimizer used : Adam
- Learning rate : initial by 0.001 and final by 0.01
- Batch size : 8

- Number of training epochs : 60 epoch
- Image size : 416 *416
- dropout: 0.0
- iou: 0.7
- box: 7.5
- cls: 0.5
- dfl: 1.5

4-Hardware and software environment :

This model trained on Colab using its GPU ,using only the ultralytices package ,it takes total time 7.45 hours for training .

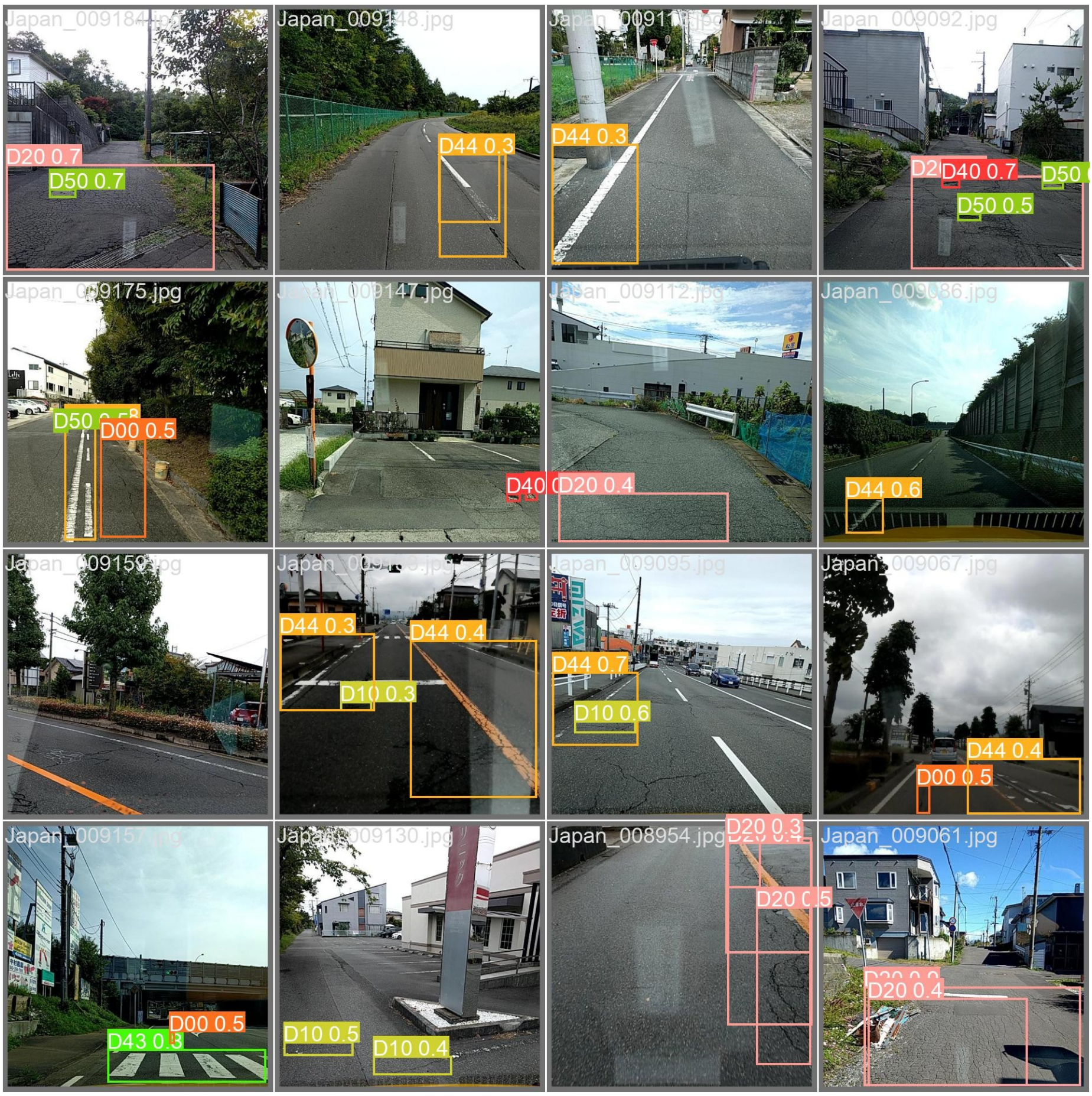
4.3) Conducted results

Sign detection model prediction results

This results are only captured through validation set as we could not test this model on real data because the scanning of the road signs in Egypt needs to scan at least a whole government to get reliable results.

Crack detection model prediction results

This model applied on many real world data , we predicted its performance on the validation dataset and on real world images from Helwan university road (these are frames takes from a video detected).
The detection done on both video and images.

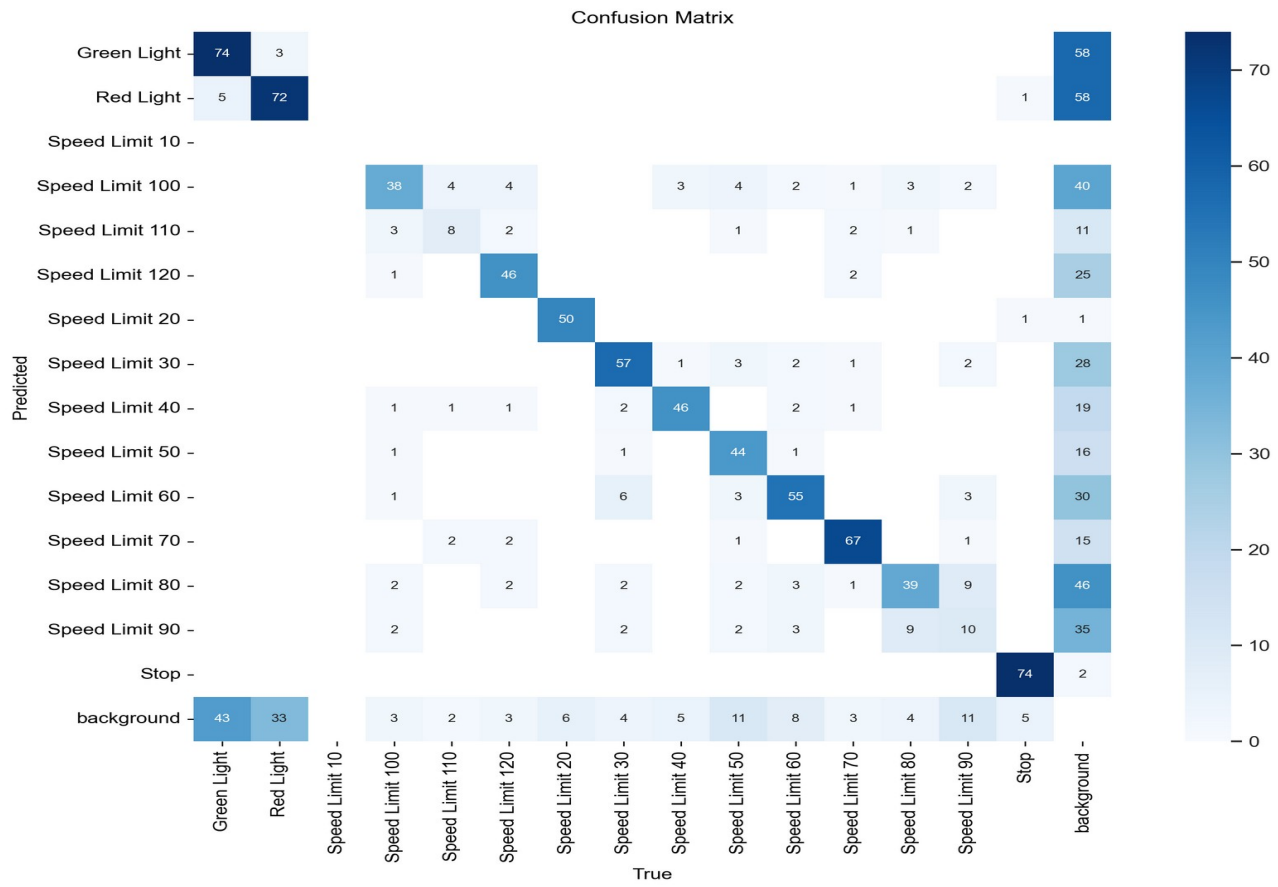


4.4) Testing and Evaluation

Sign model Final Evaluation

1-Performance matrices

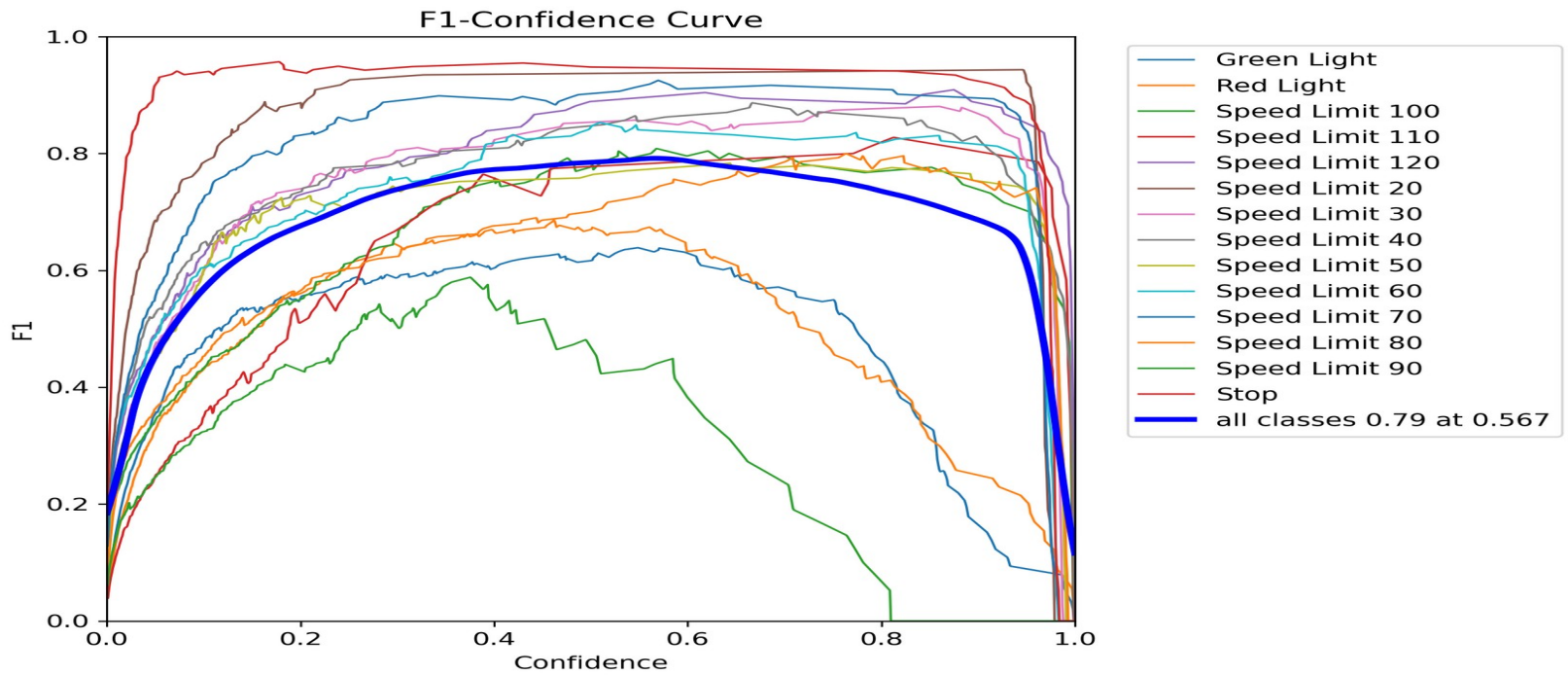
a) Confusion Matrix



b)F1 score :

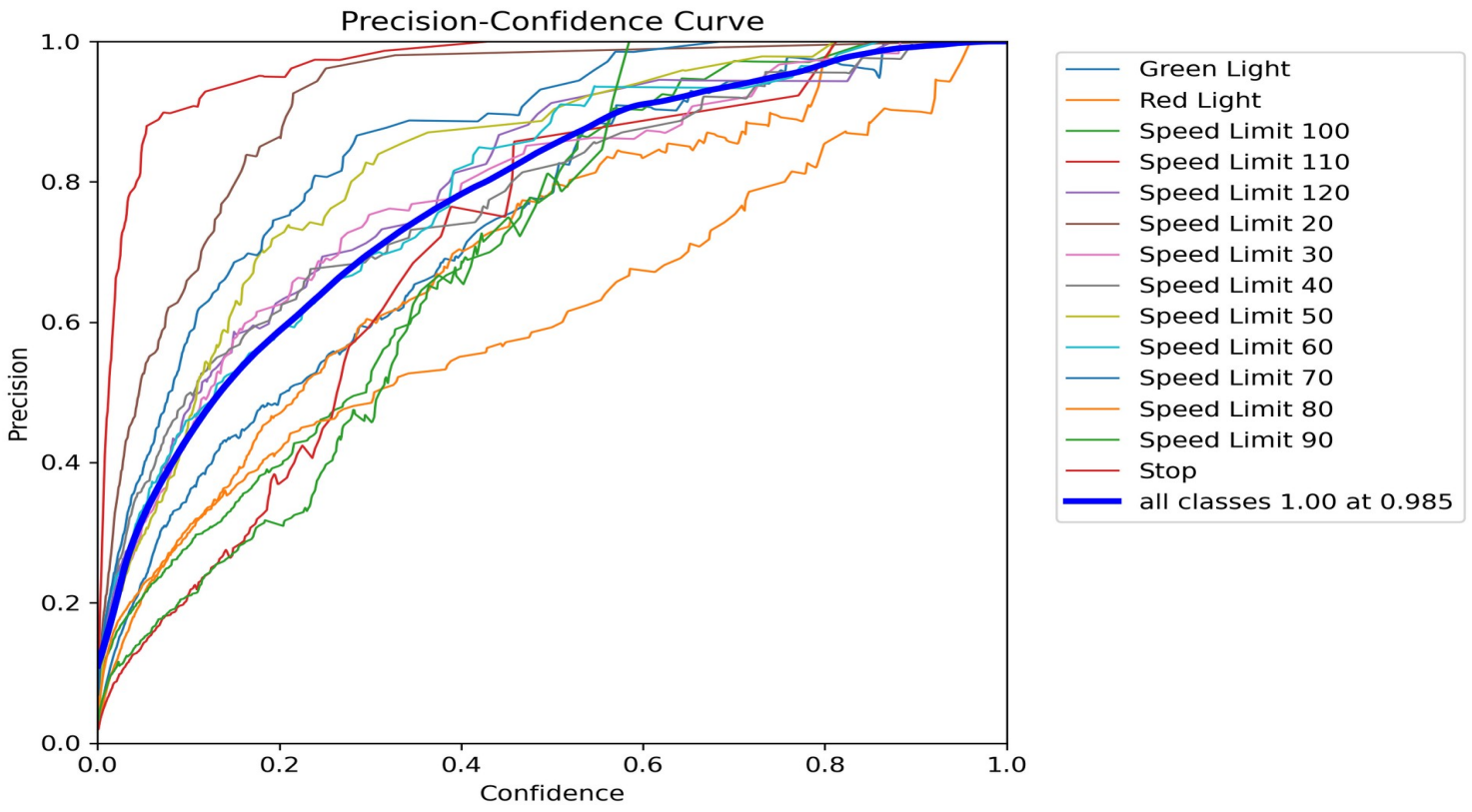
This is the curve that visualize the change in F1 score according to the precision and recall changes

The calculated F1 score is 81 %



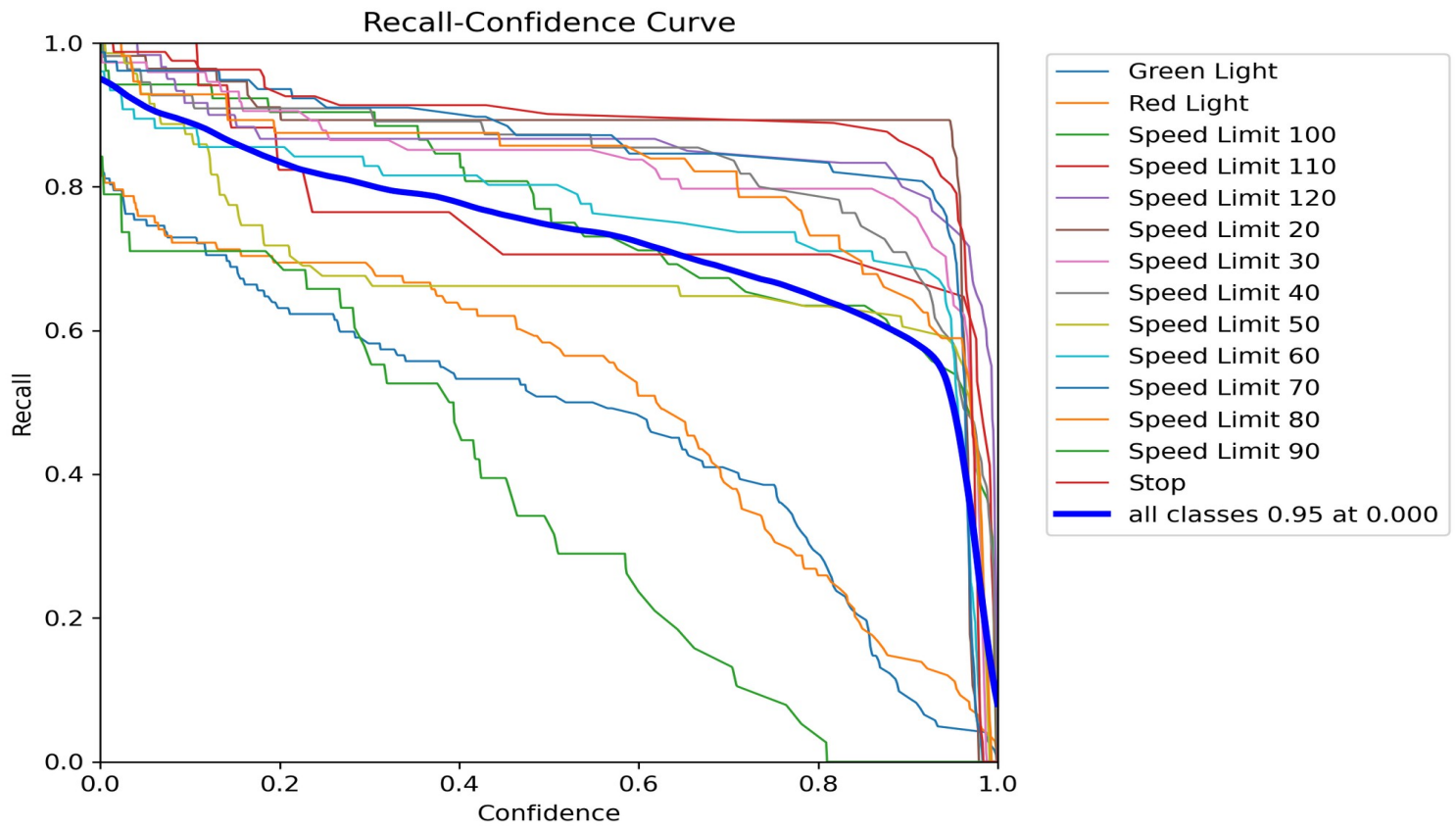
c) Precision curve

The calculated overall precision is 0.88093 approximately 88 %



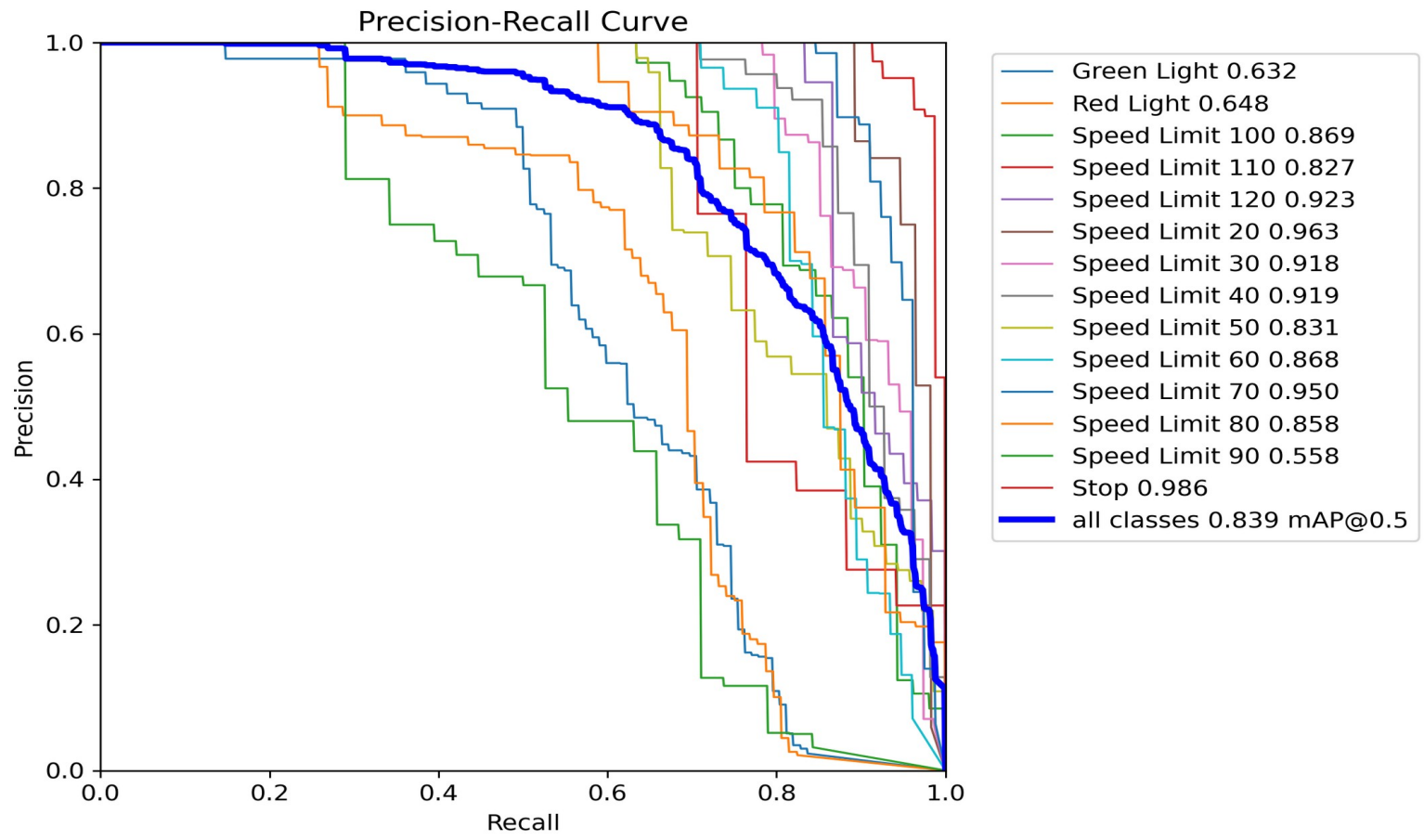
d) Recall curve :

The calculated Recall is 0.7499 approximately 75 %



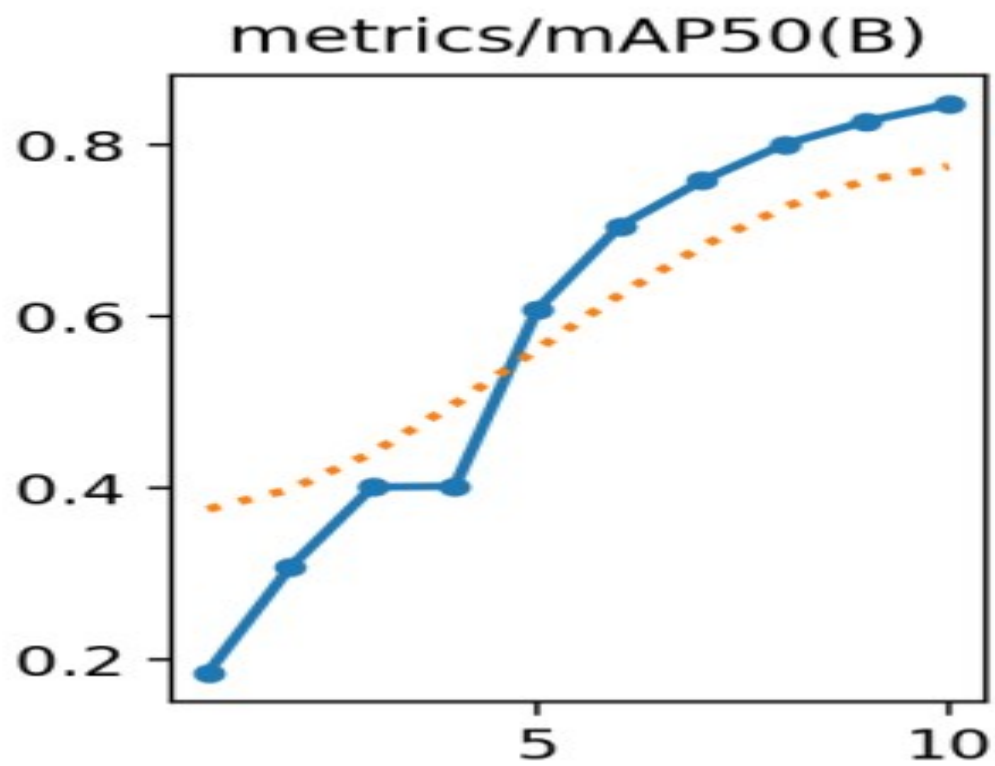
e) PR curve

Shows the relation between precision and recall

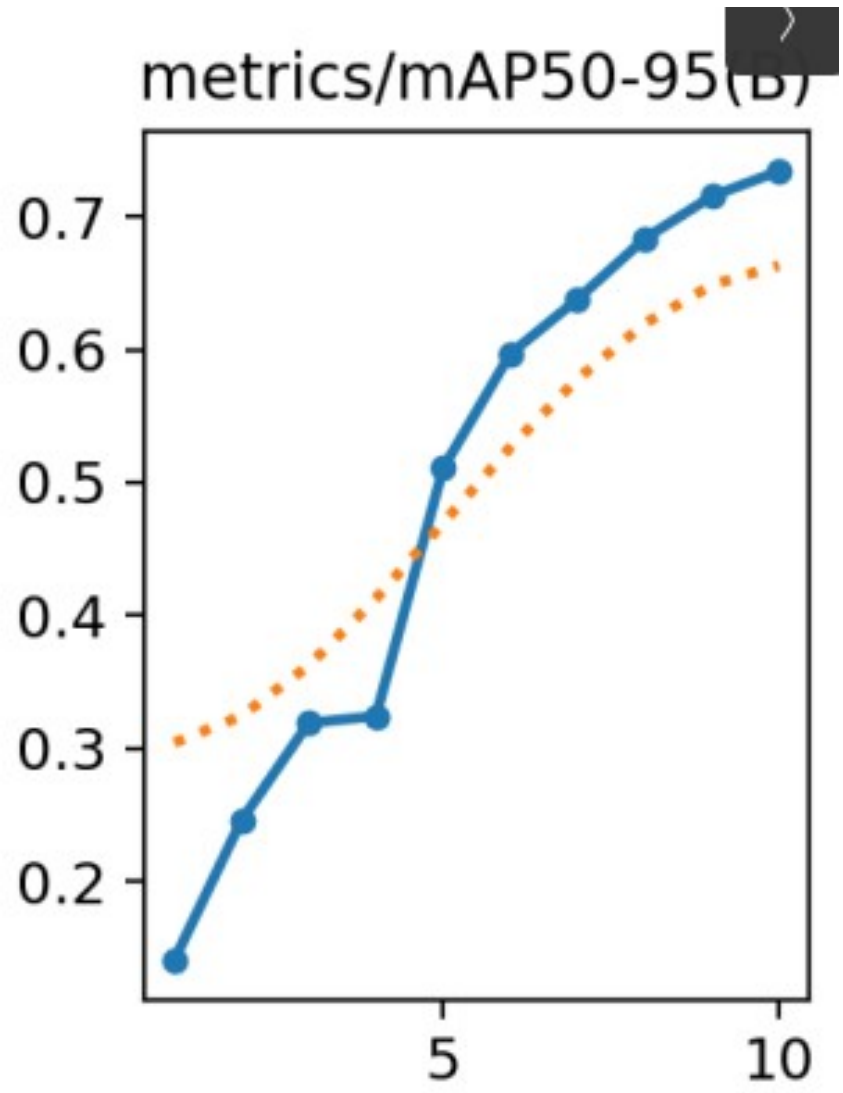


f) mean average precision at 0.5

It is calculated exactly 0.84767 approximately 85 %



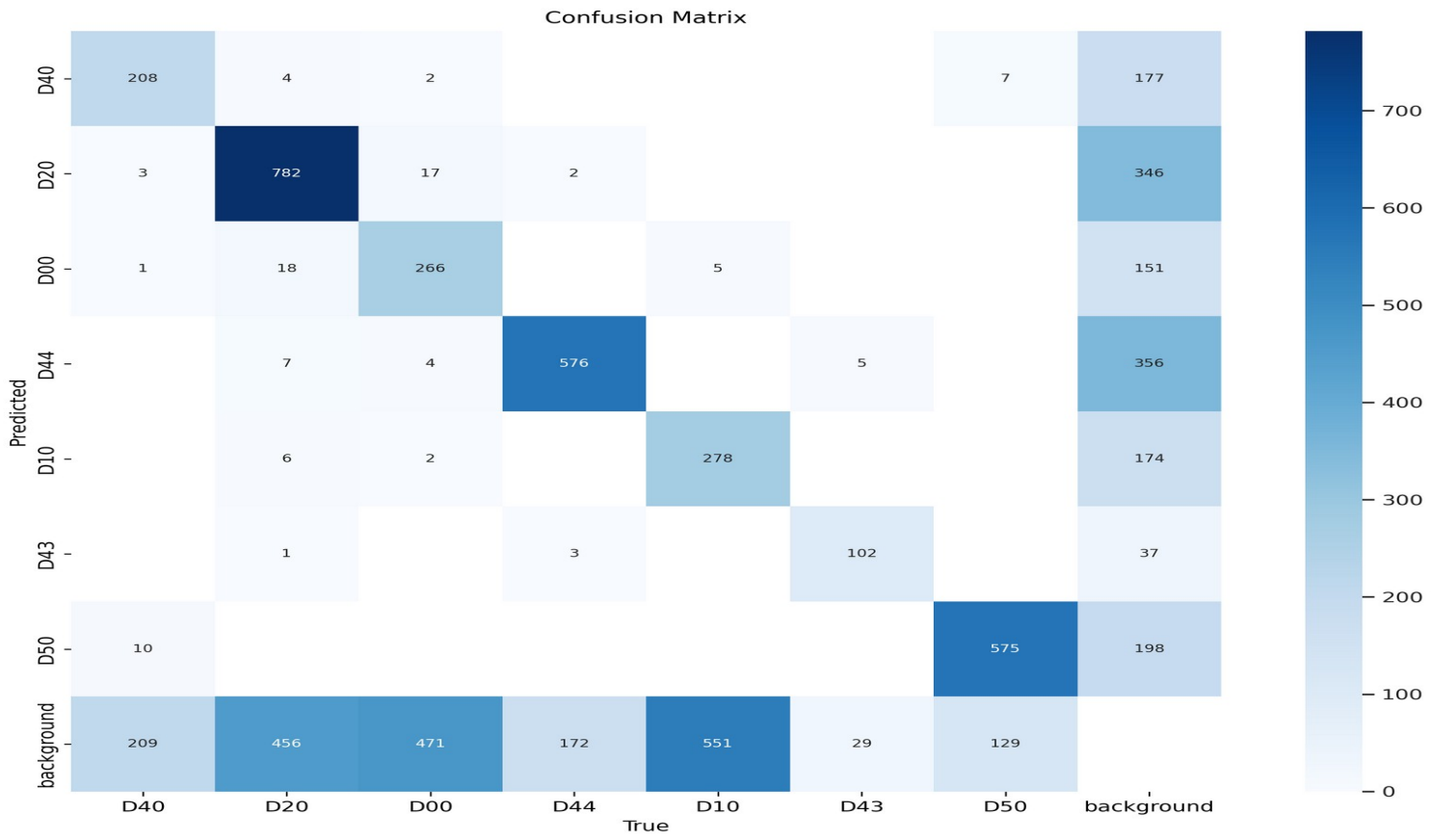
g) mean Average precision from 0.5 to 0.95
Its calculated exactly 0.73414 approximately 73 %



Crack model Final Evaluation

1-Performance matrices

a) Confusion Matrix



b)F1 score :

This is the curve that visualize the change in F1 score according to the precision and recall changes

The calculated F1 score is 85.4 % with both detection and classification which is higher than the reference paper.

c) Precision curve

The calculated overall precision is 0.881 approximately 88 % this according to the last training results

d) Recall curve :

The calculated Recall is 0.8356 approximately 83.5 %

f) mean average precision at 0.5

It is calculated exactly 0.84767 approximately 85 %

g) mean Average precision from 0.5 to 0.95

Its calculated exactly 0.7012 approximately 70 %

Chapter 5

Conclusion

In this chapter we will sum up all the workflow of our project, and our intended future work in addition to some discussion about the results.

5.1 Discussion

Achieved results :

This project achieved the core resulted that was our focus. Helping drivers to avoid road cracks as much as can to protect vehicles from damages and decrease traffic congestion as much as we can , and also gather more data about roads cracks.

Limitations :

This project is only for visualizing the cracks on map it does not navigate or route users to destinations.

Results reflection:

Our map does not contain any road sign location and the notification about them is not implemented here , as scanning a region was really hard because signs unlike cracks located far from each other and at least you have to scan an entire government to locate this points on Map.

5.2 Summary and conclusions:

An Android application for crack and road sign detection can significantly enhance road safety by incorporating several advanced features. The app will use cameras to detect cracks on the road . When the vehicle approaches a detected crack, the app will provide an audible alert to the driver, helping them avoid potential hazards. Additionally, users can enter their location into the app, which will be displayed on an interactive map. This map will continuously update to show the user's current position and save locations where cracks have been detected, allowing for easy navigation and reference.

The app will also include a survey feature where users can report new cracks by capturing images. These images will be analyzed by a back end system to verify whether they indeed show a road crack, leveraging machine learning techniques for accurate detection. The back-end system will store data on detected cracks and user-reported issues, helping to build a comprehensive database that can be used for further analysis and improvement of road conditions. By combining real-time detection, user input, and data analysis, this app aims to enhance road safety and contribute to long-term road maintenance and planning.

5.3 Future works:

Our future work plan is to add navigation and routing for user to be able to select a destination . In addition to adding a feature for the user to be able to show route statistics meaning that user could see how many cracks on this road and road speed limit.

If the tools will be available we will definitely add road signs locations and notify user while approaching them .