Smart Accident & Emergency Response System with Post-Accident Analysis

Presented by :

Mina Edwar Dawood Elias

Narden Gerges Adward Amin

Nada Abdelmoneem Ahmed

Youssef Hany Ezzat Aly

Yousef Emad El Din Ibrahim

Dalia Abd Elazim Mohamed

Table of Contents

List of Figures

List of Tables

List of Diagrams

1.Introduction

1.1Problem Definition

- Road accidents are a part of our everyday lives, and they could range from a simple bumper scratch to a huge highway disaster.
- Nearly 1.3 million deaths occurred in 2019 due to motor-vehicle injuries.
- While a lot of injuries might be instantly fatal, more than (%) of them could be saved if provided with the instant attention and actions they require, or if the emergency response was slightly faster, or more prepared
- A lot of the times accidents happen on empty roads or highways, where no one is there to react immediately or alert the authorities, fire fighters or paramedics.
- Inaccurate accident reporting could be disastrous. The impact of misinterpretation could lead to even more preventable deaths.
- Lack of knowledge and illiteracy of First-Aid basics are a major factor in the increased number of severe injuries or fatalities.

Our project aims to develop an intelligent accident detection and reporting system that utilizes real-time camera feeds and mobile applications to enhance emergency response. Upon detecting an accident, the system automatically analyzes the scene to estimate the number of vehicles involved and potential injuries. This information is then compiled into a comprehensive report, which is sent directly to the nearest hospital or emergency response unit, alongside the precise location of the accident.

Additionally, our mobile application will notify users in the vicinity of the accident, allowing them to stay informed and potentially contribute to the reporting process by providing supplementary images or details. Also, users will be alerted with proper first-aid action when scanning the scene if necessary. The First aid instructions will be generated accordingly by an ai assisted chatbot that is trained specifically on first aid. This collaborative reporting and response system ensures a more accurate and detailed assessment, supporting faster and more effective emergency response efforts.

2.Motivation

Our team is motivated to study the previous problems and create a professional application that bridges the gap between the road accidents which causes a lot of injuries & deaths and the suffering of the victims due to delays in emergency response or a lack of immediate on-scene assistance by providing a digital tool for accident detection, analysis, and first aid support.

By combining the technological interests with the project's humanitarian goals, we aim to not only create a powerful tool for accident response but also to push the boundaries of applied AI and mobile technology in critical situations.

3.Objectives

1. Communicates between Emergency Response Services network and the public, generates an accident report, Alerts and Notifies accident locations to the nearest emergency units.
- This system ensures fast and reliable communication, providing real-time accident reports and alerting nearby emergency units to improve response times.

2. Offer First-Aid assistance and immediate instructions within a short period of time.
- Users can receive step-by-step first-aid guidance immediately after an accident, helping save lives before professional help arrives.

3. Decrease the risk of fatal injuries.
- By offering timely assistance and notifying emergency responders, the app aims to significantly reduce the chances of fatal injuries.

4. Easy to use.
- The intuitive design makes it accessible for users of all ages, ensuring that critical help can be reached with minimal effort.

4.Acts

4.1 Act I : Accident Detection, Severity and Notification

4.1.1 Introduction

4.1.1.1 Problem Definition

One of the most critical challenges in accident response is the delay in detecting accidents, accurately assessing their severity as well as the lag in notifying the emergency response unit. This lag in response time can significantly impact emergency intervention, delaying medical assistance and law enforcement actions.

4.1.1.2 Background and Motivation

Automating accident detection and severity assessment enables a more proactive emergency response. Our system leverages fine-tuned **YOLOv8** for real-time accident detection, ensuring rapid identification of collision events. Once an accident is detected, a secondary deep learning model classifies the severity of the accident, distinguishing between minor, moderate, and severe incidents. This classification is critical for prioritizing emergency responses, ensuring that high-impact accidents receive immediate attention.

Furthermore, integrating a **GPS-based notification system** allows automatic alerts to be sent to nearby users, notifying them of potential hazards. Simultaneously, emergency services such as ambulances and law enforcement receive structured reports and relevant visual data frames before accident detected.

4.1.1.3 Overview of the Chapter

Accident detection and classification play a crucial role in improving road safety and emergency preparedness. Our system automates the detection process using deep learning, providing real-time classification and notification capabilities.

This chapter explores the methodologies, models, and integration workflows used in our system, covering:

- **Accident Detection Module:** YOLOv8 fine-tuned for real-time accident detection.
- **Severity Classification Module:** A deep-learning model trained to classify accidents into two severity levels.
- **GPS-Integrated Notification System:** A module that alerts nearby users and emergency units while providing accident frames for forensic analysis.

4.1.2 Research Background & Related Work

Research 1 : **Deep Learning applied to Road Accident Detection with Transfer Learning and Synthetic Images**

Research 2 : **Accident Prone System Using YOLO**

Research 3 : **ACCIDENT DETECTION USING YOLO**

Research 4 : **CAR CRASH DETECTION AND ALERT SYSTEM WITHOUT HUMAN**

Research 5 : **Visual Detection of Traffic Incident through Automatic Monitoring of Vehicle Activities**

POC / Research	Research 1	Research 2	Research 3	Research 4	Research 5
Date	2022	2023	2023	2024	2024
Data Augmentation	✓	✗	✓	✗	✓
Study Model	[1]MobileNetv2 [2]EfficientNetB1	[1]SSD, [2] YOLOv5m	YOLOv3	YOLO-CA	YOLOv8
Frame Extraction (Software/Algo)	NULL	FFMPEG software	NULL	NULL	Deep-SORT
Metrics	Train Acc / Val Acc / Test Acc 0.86 / 0.72 / 0.82 0.88 / 0.94 / 0.88	[1] Training 76.73% validation 69.69% [2] Training 99.4 % validation 85.2%	mAP: 0.94 for vehicle rollover detection. mAP: 0.93 for rear-end collision detection. mAP : 0.92 for head-on collision detection	High accuracy not mentioned by numbers.	98.4%

Table 1: Comparing others' related work

Research Takeaways

- **Single-Stage vs Two-Stage Detectors**: Single-stage detectors like YOLO models outperform two-stage detectors (ex R-CNN) in speed and accuracy, making them ideal for real-time detection.
- **Trade-Offs in YOLO Models:** YOLO offers exceptional real-time speed and accuracy, but performance depends on factors like GPU power, dataset volume, deployment flexibility, and environmental constraints.
- **Efficiency and Accuracy:** Models like MobileNetv2 and EfficientNet prioritize computational efficiency but lack the high accuracy and speed provided by YOLOv8, which excels in real-time scenarios.
- **Enhancing Model Performance:** Techniques such as data augmentation and transfer learning with pre-trained models on (COCO dataset) significantly improve accuracy. Algorithms like Deep-SORT are also used for extracting video frames for better training and detection.

4.1.3 Modules

Module 1: Accident Detection

Data

Collection

Data was collected from different resources like Kaggle, Roboflow. The initial dataset (1) was 16914 images, after that an edge case was discovered, we will talk about it in trials, made us add more images dataset (2) so the overall data contains 33253 images.

Preprocessing

Each image was resized to 640 x 640 (width and height) pixels.

Labeling

We used Roboflow for some data and the auto-label grounding DINO model to start auto-labeling the images to 0 or 1 classes (Accident & Non Accident respectively), We also started using local scripts for the other parts of data as we fixed some formats and converted them to YOLOv8 format, also some labels and converted them to our desired label, We also renamed each pic from 1 to and its corresponding label to maintain any data issues while merging the different data sources.

Augmentation

Augmentations applied on some of the datasets as some were collected with their augmentations, The type of augmentations that we did on the dataset was:

Grayscale 15%, Saturation -25% and 25%, Brightness -15% and 15%, Exposure -10% and 10%, Noise addition 0.1%.

Split

The **initial dataset (1)** was split into 82.3% train with 13926 images, 11.7% valid with 1991 images and 6% test with 997 images. The **final dataset (2)** was split into 83.3% train with 27707 images, 11.3% valid with 3787 images and 5.4% test with 1759 images.

Sample

Figure 1: Accident dataset

1a) noisy conditions 1b) rainy conditions 1c) high-angle view 1d) different-angle view

Model & Methodologies

Model Selection

After reading some papers about the Accident and object detection models using CNN, YOLO with different versions and also the Faster-RCNN we decided that the YOLOv8 from Ultralytics library will be the best fit to our problem as we need strong efficiency and speed in the inference on the images compared to the Faster-RCNN that takes more time to infer, so we started by using the YOLOv8m.pt pretrained weights and started finetuning these weights with our data.

Architecture

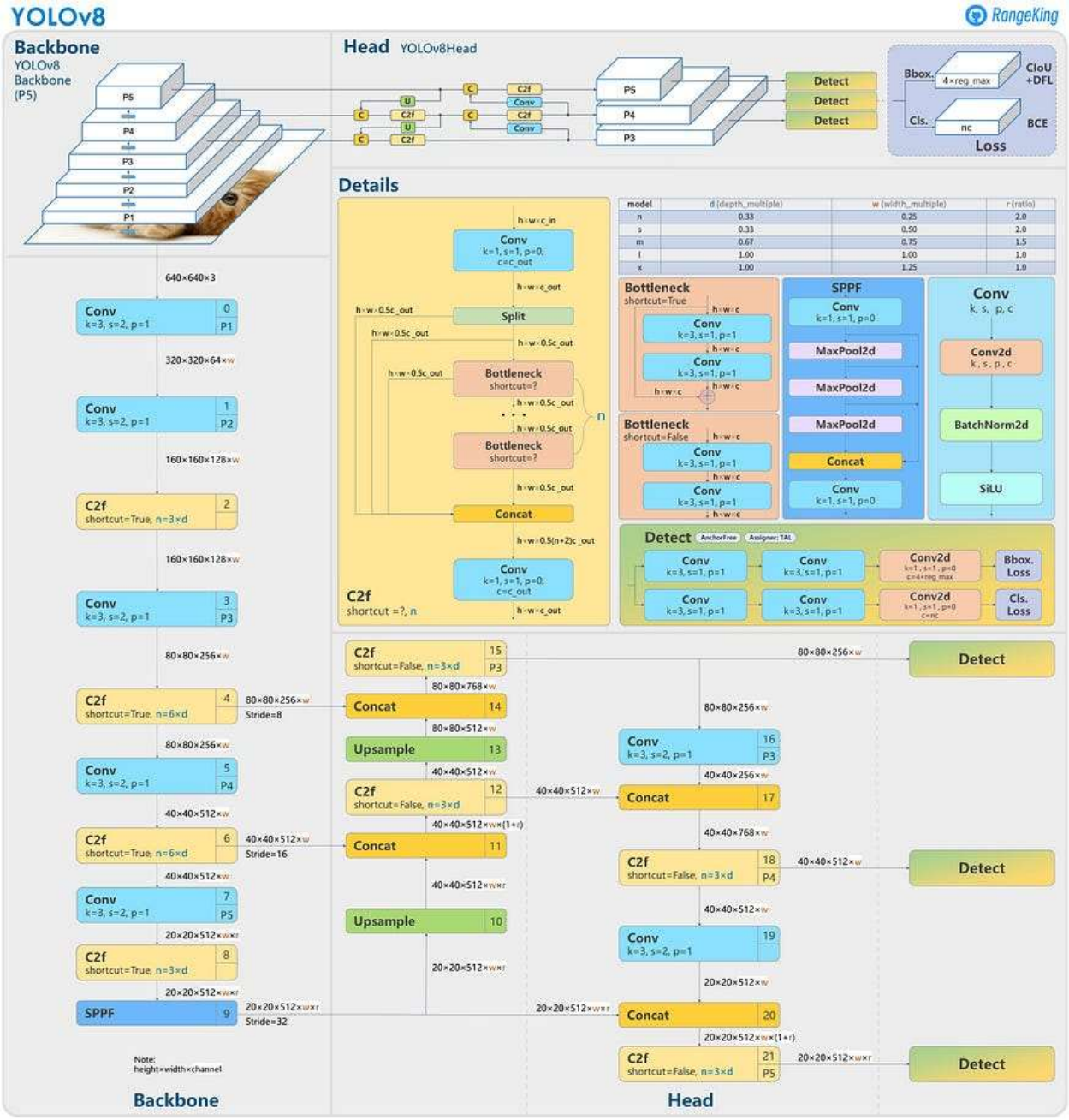

Figure 2: Architecture of YOLOv8 referenced from Ultralytics library

Training Process

We used Kaggle GPUs to start our training (fine-tuning) process for 50 epochs on the initial dataset and then another 50 epochs on the second dataset with imgsz=640, batch=32 and save_period=5, also we edited the data.yaml file to include our 2 classes: Accident class 0 and Non Accident class 1.

Output

Accident Bounding Boxes and Class detected text on each frame

3a) frame pre-accident 3b) frame of detected accident 3c) frame post-accident

3d) frame pre-accident 3e) frame of detected accident 3f) frame post-accident

3g) frame pre-accident 3h) frame of detected accident 3i) frame post-accident

Figure 3: Pre and Post accident detection frames

Evaluation

Our model shows incredible performance compared to another models in terms of mAP, mAP 50:95 as shown in the following Table:

Class	Images	Instances	Box(P	R	mAP50	Map50-95)
All	3787	9212	0.901	0.866	0.922	0.765
Accident	1993	2193	0.942	0.954	0.979	0.827
Non Accident	1843	7019	0.86	0.779	0.865	0.704

Table 2: Shows the evaluation metrics of our model

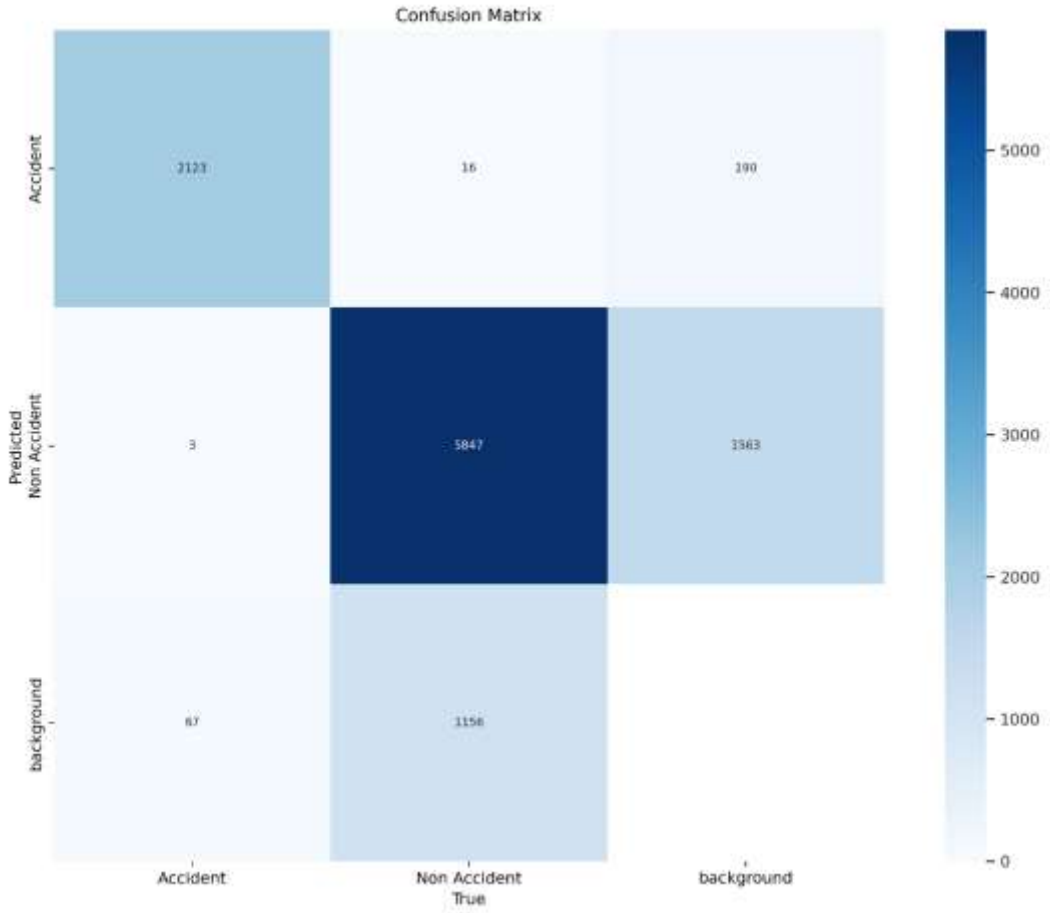

Figure 4: Previewing the confusion matrix of our model detections

Trials

Trial 1 :

We started with the initial dataset which handled the case of car accidents images only and we started fine-tuning and our model performed well but after that we started testing some cases from outside the distribution of the data like vehicle and people involved in the accident cases, also we found that it missed a lot of non accident images so we noticed that it didn't give the expected performance on them so we did a 2nd trial.

Trial 2 :

We started by collecting images of all the edge cases we found during the testing of trial 1 and then we started fine-tuning our trial 1 model with these images and it gave the expected performance while testing with excellent evaluation metrics.

Next Steps

-We are currently working on using this inference as a real-time inference after we exported our weights as onnx weights.

-We are also working on deploying the model on a cloud service to link it with the incoming application.

Module 2: Accident Severity Classification

Data

Collection

We obtained the dataset from Roboflow(dataset), consisting of 940 images for training and 214 images for validation.

Preprocessing

Each image was resized to 224 x 224 (width and height) pixels.

Labeling

We used local scripts to classify images into Low (Classes 0 & 2) and High (Classes 1 & 3), convert labels to YOLOv8 format, and rename files for consistency, ensuring smooth data integration.

Augmentation

Augmentations applied on some of the datasets as some were collected with their augmentations, The type of augmentations that we did on the dataset was:

Grayscale 15%, Saturation -25% and 25%, Brightness -15% and 15%, Exposure -10% and 10%, Noise addition 0.1%.

Split

The dataset was structured into two subsets, with the training set containing 940 images across two classes and the validation set comprising 214 images. This split ensures a balanced distribution for effective model training and evaluation.

Sample

Figure 5: Samples of severe accidents

Model & Methodologies

Model Selection

We selected **EfficientNetB0** as the backbone due to its balance between accuracy and computational efficiency. EfficientNetB0 employs a compound scaling method that optimally balances depth, width, and resolution, making it an ideal choice for deep learning tasks with limited computational resources. Given the need for a lightweight yet powerful model, EfficientNetB0 provides high feature extraction capabilities while maintaining a relatively small number of trainable parameters.

Architecture

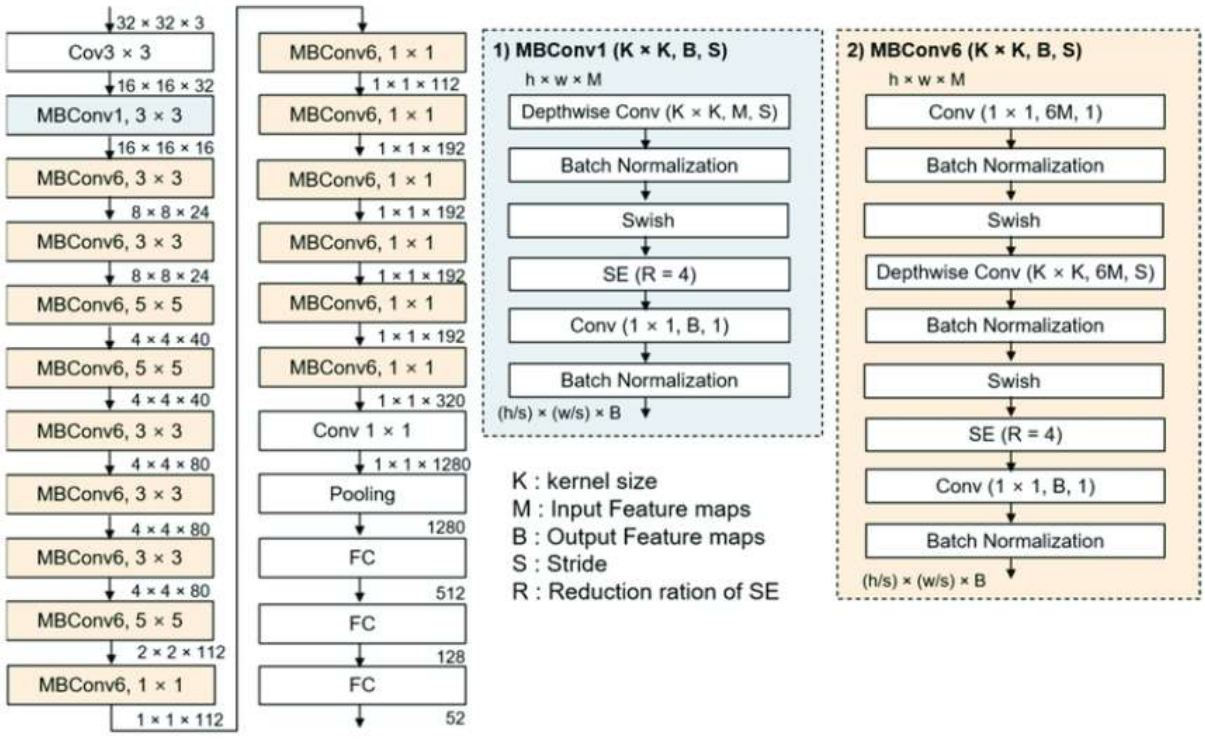

Figure 6: EfficientNetB0 architecture referenced

global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	top_activation[0][0]
dense (Dense)	(None, 128)	163,968	global_average_poolin…
dense_1 (Dense)	(None, 1)	129	dense[0][0]

Total params: 12,556,960 (47.90 MB)
Trainable params: 4,171,645 (15.91 MB)
Non-trainable params: 42,023 (164.16 KB)
Optimizer params: 8,343,292 (31.83 MB)

Figure 7: Our Model classifier head on the backbone

Training Process

We trained a **severity classification model** using **Google Colab's GPU** in two phases. First, we trained for **30 epochs** with a frozen **EfficientNetB0** backbone, using binary_crossentropy loss, the Adam optimizer (LR=0.001), and data augmentation. Next, we fine-tuned for another **30 epochs** by unfreezing the backbone and reducing the learning rate to **1e-4**. The dataset was preprocessed into **High Severity** (Moderate/Severe) and **Low Severity** (Minor/No Accident) classes, split into training and validation sets. The final model, saved as severity_classification.h5, achieved high accuracy and can predict severity (e.g., Severity: High, Confidence: 0.92). This ensures robust accident severity classification.

Output

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Confidence value: [[0.00021557]]
None
Severity: Low
```

8a) Low Severity predicted by our model

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Confidence value: [[0.9999721]]
Severity: High
```

8b) High Severity predicted by our model

Figure 8: Severity Classification

Evaluation

```
Epoch 29/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 14s 350ms/step - accuracy: 0.9980 - loss: 0.0099 - val_accuracy: 0.9159 - val_loss: 0.3319
Epoch 30/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 20s 371ms/step - accuracy: 0.9972 - loss: 0.0083 - val_accuracy: 0.9206 - val_loss: 0.3457
```

Figure 9: Evaluation metrics of the model

Trials

Trial 1: Initial Model Training with 15 Epochs
The model was trained for 15 epochs using the EfficientNetB0 backbone. However, it exhibited underfitting, with low accuracy and high loss on both training and validation datasets. This indicated that 15 epochs were insufficient for the model to learn effectively.

Trial 2: Extended Training with 50 Epochs
To address underfitting, the model was trained for 50 epochs. While training accuracy improved, validation accuracy plateaued and then decreased, indicating overfitting. The model memorized the training data instead of generalizing well.

Trial 3: Optimized Training with 40 Epochs
The team reduced the epochs to 40. The model performed better than in Trial 2 but still showed signs of overfitting, as validation accuracy began to diverge from training accuracy toward the end of training.

Trial 4: Final Training with 30 Epochs
Training the model for 30 epochs yielded the best results. The model achieved high accuracy and balanced loss values on both training and validation datasets, with no significant divergence between metrics. This demonstrated that 30 epochs provided the optimal balance, avoiding both underfitting and overfitting.
Conclusion
Through iterative trials, the team found that 30 epochs was the optimal number for training the model, ensuring robust performance and generalization on the severity classification task.

Next Steps

We are currently working on deploying the model on a cloud service to link it with the incoming application.

Module 3: Accident Notification (GPS Module)

Overview

This module is part of the first phase of our system, and its main job is to keep people safe by alerting them about accidents. When a serious accident is detected and confirmed, this module sends a real-time notification to users who are near the accident location. The notification includes the accident's location so that people can avoid the area or take necessary action.
The module is built using Express.js and follows the MVC (Model-View-Controller) architecture, which makes it easy to organize and maintain. It uses Firebase Cloud Messaging (FCM) to send notifications quickly and reliably to users' devices.
In short, this module is all about quickly sharing critical information when it matters most, helping users stay informed and safe.

System Architecture

Overview:

The system architecture is designed to support real-time accident reporting and notification. The GPS module plays a central role in identifying users near the accident location and sending them notifications. The system is built using the **Model-View-Controller (MVC)** architecture, ensuring a clean separation of concerns and making it scalable and maintainable.

Key components:

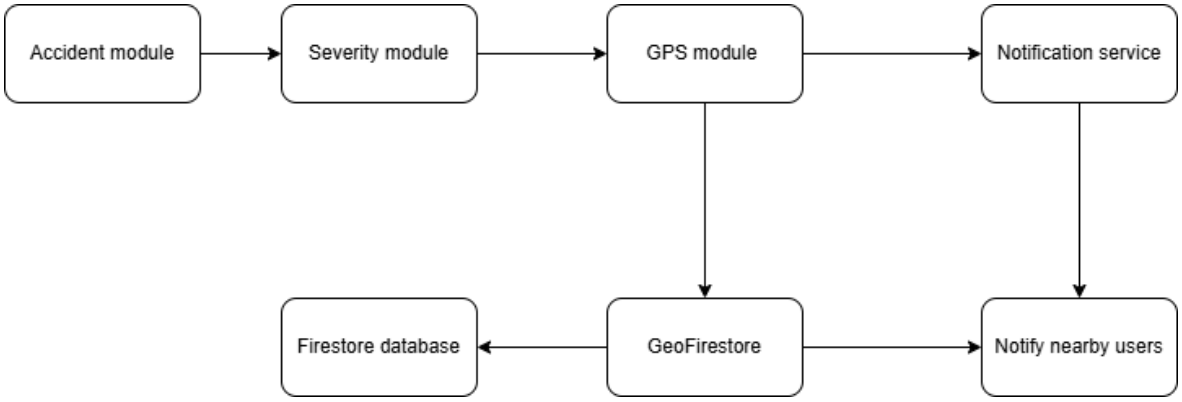

Diagram 1: Illustration of the key components of the GPS module

- **Accident module**: this component is responsible for Accident detection and reporting.
- **Severity module**: Represents the component that assures that the accident is of high severity so that we can continue our pipeline and alert the GPS module.
- **GPS module:** queries GeoFirestore and retrieves a list of nearby users to send to the notification service.
- **Notification service:** sends notification to nearby users via Firebase Cloud Messaging.
- **GeoFirestore:** stores the Geospatial location of users and handles queries to find nearby users.
- **Firestore database:** the main application database that stores users locations and accidents details.

Data flow:

- **The accident module** detects an accident and sends the detections to the severity module.
- The **severity module** determines if the accident is of high severity and then sends its location to the **GPS module.**
- The **GPS module** receives the accident and then searches the **GeoFirestore** for nearby users and then sends a list of Active users to the **notification service (FCM).**
- The **notification service** then verifies that the user has permitted the application notifications and then proceeds with the alerts containing location of the accident.

Module design

This module mainly depends on the efficiency and real-time responsiveness of **GeoFirestore database**, which stores the Geospatial locations of users in real-time and updates them in a reasonable frequency so that it can be both precise and efficient, it also depends on **Firebase Cloud Messaging** service which receives a list of FCM tokens of the nearby users and sends them a notification indicating an accident nearby with its location.

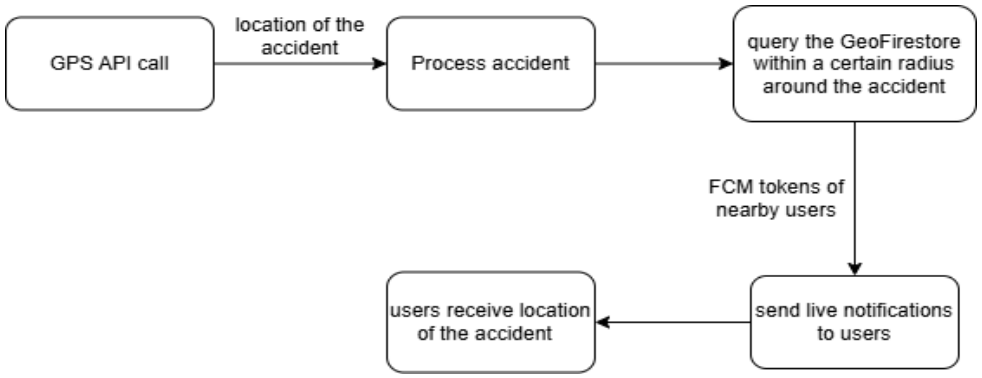

Diagram 2: The GPS module design

GeoFirestore database

GeoFirestore is a database designed to store and manage user locations efficiently. It uses **geopoints** (latitude and longitude) to represent locations and converts them into **geohashes**—a string that represents a specific geographic area. The longer the geohash, the more precise the location it represents.
For example, the geohash for San Francisco (37.7749, -122.4194) might look something like 9q8yy.
GeoFirestore uses these geohashes to **quickly index and search for locations** within a certain radius. It works by dividing the world into a grid of cells, where each cell has a unique geohash. Nearby locations share similar geohash prefixes, making it easy to find users or points of interest close to a specific location.
One of the best features of GeoFirestore is its support for **real-time updates**. This means we can continuously update users' locations as they move, ensuring we always have accurate and up-to-date information. When an accident occurs, we can use this data to notify only the users who are closest to the incident, making the system both efficient and precise.Firebase Cloud Messaging (FCM)FCM is a Google service that lets us send **real-time notifications** to users of our app. When a user downloads and opens the app for the first time, FCM generates a unique **FCM token** for that specific device and app instance. Think of this token as a unique address that allows us to send messages directly to that user's device.
We store this FCM token in our database, along with the user's location. When an accident happens, we can use this token to send an **accident alert** to users who are near the incident. The notification can include details like the accident's location, helping users stay informed and avoid the area if needed.
So the data flow is from the API call which indicates that there is an accident with high severity , the location of the accident (longitude and latitude) is passed in the call , our controller handles the call and extracts the location from the request body and makes it the center of the radius we search with in the

GeoFirestore database, we call the API to query the database for the locations in the radius and receive a list of FCM token of the closest users to the accident , and passes these tokens to the notifications service along with the accident location to continue and alert users.

4.1.4 Integration

This system is built using independent modules that communicate with each other through **API requests**. Each module is deployed using **FastAPI**, and they work together to detect accidents, assess their severity, and notify nearby users and emergency responders. Here's how the flow works:

1. **Severity Module**:
   This module takes in an array of image files and checks if any of them show a **high-severity accident** using a severity prediction model. If at least one image is flagged as high severity, the module responds with **"High"**. Otherwise, it responds with **"Low"**.

2. **Accident Detection Module**:
   When this module detects an accident, it calls the **Severity Module's API** and sends the images it captured. It receives a response of “High” or “Low” from the severity module.

3. **High-Severity Accident**: If the Severity Module responds with **"High"**, the system proceeds with the following steps:
   -The location of the accident is sent to the **GPS Module** and to **emergency responders**.
   -5 pre-accident frames (saved during detection) are sent to authorities for further investigation.

4. **GPS Module**:
   The GPS Module uses the accident location to search for nearby users within a specific radius. It sends a **real-time notification** to these users, alerting them about the accident
   and providing the location.

Figure 10: Integrated sample of our process

Flow diagram

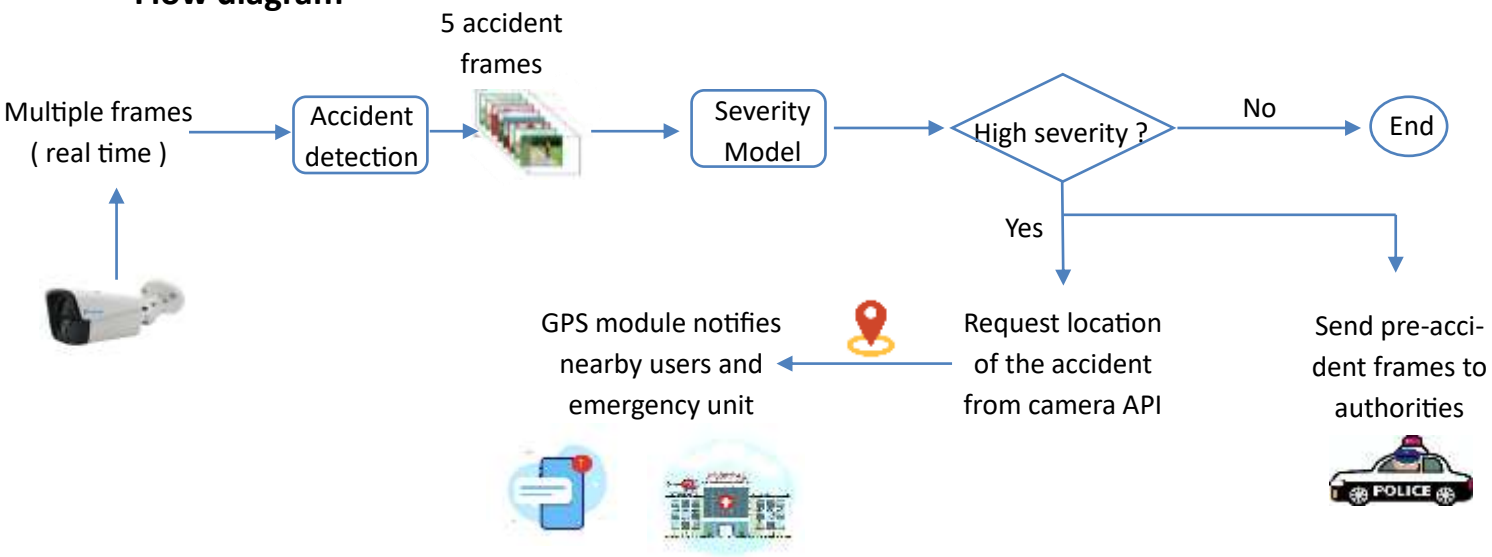

Diagram 3: The flow diagram of our process

4.1.5 Conclusion

In this chapter, we introduced an advanced accident detection and notification system that leverages deep learning and GPS-based alerts to enhance emergency response efficiency. By fine-tuning YOLOv8 for real-time accident detection and integrating a severity classification model, our approach ensures accurate assessment and prioritization of accident cases. Additionally, the GPS module facilitates instant notifications to nearby users and emergency units, while providing law enforcement with crucial accident footage for further investigation. This automated system not only reduces response time but also improves decision-making for medical teams and authorities, ultimately saving lives and enhancing road safety. 🚀 **Stay tuned for the next chapter, where we talk about our automated solutions for the Injuries Detection, Segmentation and Reporting** 🔥 👀