EE1222: Introduction to Automation Project

Real-Time Accident Detection

PREPARED BY

Dev Rishi Verma (2022BTech028)

Dhruv Sharma (2022BTech030)

Kashish Khangarot (2022BTech048)

FACULTY GUIDE

Mr. Divanshu Jain

Department of Computer Science and Engineering

Institute of Engineering and Technology

JK Lakshmipat University Jaipur

November 2024

CERTIFICATE

This is to certify that the project work entitled “**Real-Time Accident Detection** ” submitted by **Dev Rishi Verma, Dhruv Sharma, Kashish Khangarot** towards the fulfilment of the project for the degree of **Bachelor of Technology in Computer Science Engineering** of JK Lakshmipat University, Jaipur is the record of work carried out by them under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for the final submission.

Mr. Divanshu Jain

Assistant Professor

Department of Computer Science Engineering
Institute of Engineering & Technology

JK Lakshmipat University, Jaipur

Date of Submission: December 1, 2024

ACKNOWLEDGEMENTS

We would like to express our appreciation to our faculty and advisors, **Mr. Divanshu Jain**, Assistant Professor, IET, JK Lakshmipat University, Mr. Mahesh Saini Lab, Assistant, IET, JK Lakshmipat University, for their invaluable guidance, support and expertise throughout this project.

We would also like to extend our appreciation to **Dr Dheeraj Sanghi**, Vice Chancellor, JK Lakshmipat University, and **Dr Renu Jain**, Director, Institute of Engineering and Technology, JK Lakshmipat University for their ongoing encouragement and support during this project.

We would also like to thank our parents for their encouragement and support throughout the project.

Last but not least, we would like to thank our family and friends for their moral and emotional support throughout this project.

Sincerely,
Dev Rishi Verma
Dhruv Sharma
Kashish Khangarot

ABSTRACT

Road accidents are a critical global issue, requiring swift detection and response to minimize casualties and disruption. This project introduces a real-time accident detection system that identifies incidents promptly and automatically alerts emergency services enabling faster decision-making and emergency response. The system is designed to operate seamlessly, providing accurate and efficient detection to ensure timely communication with relevant authorities. By integrating cutting-edge technologies for real-time monitoring and alerting, this solution aims to enhance road safety, reduce response times, and support the development of smarter, more efficient urban traffic management systems. Ultimately, it seeks to mitigate the impact of accidents and improve overall traffic operations.

TABLE OF CONTENT

REAL – TIME ACCIDENT DETECTION USING CCTV CAMERA

CHAPTER 1: PROBLEM STATEMENT

The goal of this project is to design and implement a real-time accident detection system using CCTV cameras to enhance road safety and improve emergency response efficiency. Once an accident is detected, the system automatically alerts the control room via a server, ensuring timely notification and intervention. This project addresses the limitations of manual accident reporting by providing an automated, precise, and reliable solution, contributing to smarter and more responsive urban infrastructure.

CHAPTER 2: LITERATURE SURVEY

The rapid pace of development of advanced technologies for immediate detection and response to road crashes underscores the urgent need for such work. As a result, a great deal of research has been done on the application of machine learning, computer vision, and edge computing platforms, including Jetson Nano, to create machines that can identify the situation and alert emergency services.

1. Accident Detection Using Computer Vision

The use of computer vision and machine learning in accident detection using video data captured by cameras on the road or in the car is very common. The main application of these systems is the real-time detection of vehicles, pedestrians, and accidents from video or photo frames.

In this case, CNN is trained to determine the position and momentum of the vehicle and predict the incident. Although CNNs provide the highest accuracy, they have many uses, which makes instant applications difficult and especially challenging for devices with limited resources.

To overcome the above limitations, lightweight models such as YOLO (You Only Look Once) have attracted attention due to their effectiveness in generating images on the fly (2016). The real-time processing capability makes YOLO ideal for use in accident detection, where vehicles and incidents need to be quickly identified.

2. Real-time Object Detection with YOLO

YOLO is very popular for real-time object detection, mainly due to its fast and accurate algorithm. It provides versions such as YOLOv3, YOLOv4, YOLOv5, and now YOLOv8.

- **YOLOv8** is optimized for edge devices, achieving a good balance between model accuracy and computational performance.
- **YOLOv3 (2020)** achieved accurate vehicle detection based on variables such as lighting and weather conditions. However, its performance degrades in areas with low resolution or heavy traffic, requiring further model updates.
- **YOLOv4 and YOLOv5 (2022)** were embedded in vehicles to prevent collisions. Research highlighted the need for immediate processing to warn about unexpected traffic emergencies.

Real-time processing of video frames requires the use of YOLO in traffic monitoring.

3. Edge Computing for Real-Time Processing

Edge computing provides support for instantaneous event detection by processing data directly on the device instead of in the cloud, reducing latency and dependency. Devices such as the NVIDIA Jetson Nano are used to run deep learning models like YOLO, suitable for specific applications.

- **Wang et al. (2021)** showed that high speed and low power consumption could be achieved by optimizing models like YOLOv4-miniature on the Jetson Nano.
- **Feng et al. (2020)** developed a collision detection system for smart cities using Jetson Nano.

Edge computing reduces response time, which is critical for emergencies.

4. Model Optimization and Multi-Threading

Deep learning models need optimization to balance speed, resource efficiency, and performance on limited hardware like Jetson Nano.

- **Model Pruning**: A technique proposed in 2015, reduces the size of deep learning models by trimming less significant values, speeding up processing without sacrificing performance.
- **YOLOv8 Nano**: Designed with memory-efficient architecture, ensuring low computational requirements for edge devices.
- **Multi-Threading**: Techniques like parallel threading for preprocessing, inference, and postprocessing improve real-time processing efficiency.
- **Wu et al. (2019)** demonstrated that multi-threading significantly enhances object detection in real-time systems.

5. Challenges and Future Directions

Despite these advances in accident detection systems, there are still challenges that must be overcome to ensure accuracy for different conditions of poor lighting, weather, or occlusion through other vehicles.

Further, traffic accident detection systems have to be flexible about changing and dynamic environments. There would be a continuous update of training datasets from diverse conditions of traffic occurrences and their scenarios as well as the behavior of the vehicle for better generalization of such systems.

Edge computing is still very robust for the Jetson Nano platform. Future developments could be more oriented toward hardware and software aspects for further system performance. Enhancing the energy efficiency of edge devices and adding AI accelerators to enable speed up for deep learning inference can be part of further developments.

The integration of machine learning models like YOLO on edge computing devices has established strong momentum in real-time accident detection systems. Among the challenges that continue, despite the advancement in AI and edge computing with computer vision, is robustness across different conditions. This feature is expected to lead, on the other hand, to greater efficiency and reliability in solutions. Optimised models, sensor data, and hardware improvements are likely components for further research in accident detection operating in real-time environments.

CHAPTER 3: PROJECT OBJECTIVES

The objective of this project is to create a real-time accident detection system that automates accident identification and reporting for enhanced road safety.

- Real-Time Accident Detection: Use CCTV cameras with a YOLOv8 model to detect accidents instantly.
- Automated Alerts: Notify the control room/flask server for quick response using wifi .
- Improved Road Safety: Minimize response time and improve overall traffic management by automating accident detection.

CHAPTER 4: COMPONENTS USED

The materials required for this project are:

- **JetsonNano 2GB**

The Jetson Nano 2GB is a compact and affordable edge computing device developed by NVIDIA, designed for AI and machine learning applications. Despite its small size and cost-effectiveness, it delivers powerful performance for deploying models like YOLOv8. Its capabilities include real-time data processing, making it ideal for on-site accident detection with low latency. The 2GB version is specifically suited for lightweight AI tasks while maintaining energy efficiency.

- **USB Camera**

This USB camera features a 1/4 CMOS sensor with a resolution of 640x480 pixels, making it a reliable choice for capturing real-time video footage. Its compact design and collapsible cable make it easy to integrate into projects, especially for setups with limited space. The camera's compatibility with the Jetson Nano 2GB ensures smooth video feed acquisition for accurate accident detection using the YOLOv8 model.

CHAPTER 5: SCHEMATIC DIAGRAM

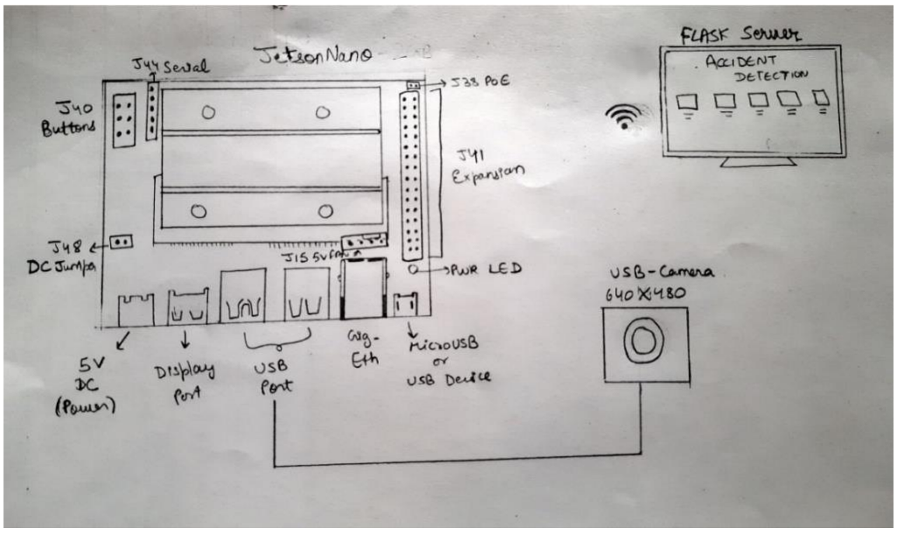


Fig: JetsonNano with camera connected to flask server

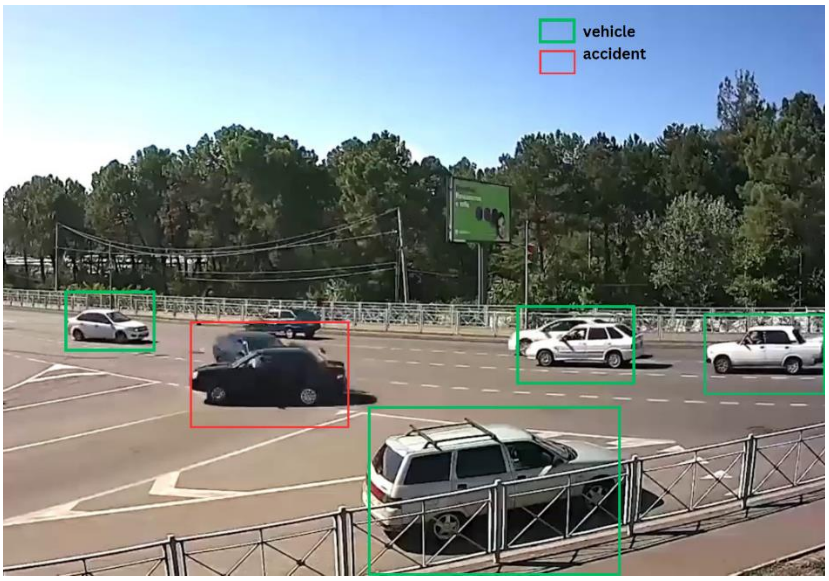


Fig: Annotation sample

CHAPTER 6: SOFTWARE AND LIBRARIES USED

1. **Jetson Nano 2GB OS Image**
   - The operating system image designed for Jetson Nano, based on Ubuntu, with NVIDIA's JetPack SDK pre-installed for AI and ML tasks.
2. **Python libraries**
   - Used to set up the environment, train the model, and integrate components.
3. **YOLOv8**
   - The object detection model used for accident detection, trained on the dataset and deployed on the Jetson Nano.
4. **Flask**
   - A lightweight web framework used to create the server will work using wifi.
5. **OpenCV**
   - A computer vision library to interface with the USB camera and process live video feeds.
6. **PyTorch**
   - The framework used for training the YOLOv8 model and integrate on Jetson Nano.
7. **CUDA Toolkit**
   - NVIDIA’s platform to fasten inference on the Jetson Nano GPU.
8. **JetPack SDK**
   - A software development kit that includes CUDA, cuDNN, and other tools optimized for Jetson devices.

CHAPTER 7: METHODOLOGY

Step 1: Dataset Preparation

1. Video to Frames Conversion

- Collected videos related to accidents and regular road scenes.
- Converted videos into individual frames to create a large dataset of images.
- Saved extracted frames in a designated folder.

2. Annotation Using LabelImg

- Annotated frames manually using the LabelImg tool to label objects of interest.
- Defined two classes:
  - Class 1: Vehicles
  - Class 0: Accidents
- Saved annotations in YOLO format, where each image has a corresponding `.txt` file with respected coordinates.

3. Splitting Data into Train, Validation, and Test Sets

- Organized the dataset into three subsets:
  - Training set: 70% of the data, used for training the model.
  - Validation set: 20% of the data, used for tuning hyperparameters and preventing overfitting.
  - Test set: 10% of the data, used for evaluating model performance.
- Created separate folders for images and labels for each subset.

4. Dataset Structure

- Images and labels were stored in separate directories with same filenames.
- Labels:
  - Class ID (1 for vehicles, 0 for accidents)
  - Bounding box dimensions to the image size: coordinate (x, y), width, and height.

Step 2: Setting Up Jetson Nano 2GB

1. Prepare the Jetson Nano 2GB Hardware

- Unbox the Jetson Nano 2GB and connect it to necessary peripherals:
    - Monitor via HDMI
    - Keyboard and mouse via USB
    - Micro-USB power supply (5V 4A)
    - MicroSD card (at least 16GB UHS-1) for the operating system

2. Download and Flash Jetson Nano 2GB OS Image

    - Download the official Jetson Nano image from NVIDIA's website: Jetson Nano Developer Kit SD Card Image.
    - Use tools like **Etcher** or **Balena Etcher** to flash the OS image onto the MicroSD card:
        - Insert the MicroSD card into your computer.
        - Open Ether and select the downloaded Jetson Nano image.
        - Select the MicroSD card as the target device.
        - Click **Flash** to burn the OS onto the MicroSD card.

3. Booting Up Jetson Nano

    - Once the MicroSD card is ready, insert it into the Jetson Nano.
    - Connect the power supply to the Jetson Nano and power it on.
    - We will see the system boot up and the initial setup process will begin.

4. Complete Initial Setup

    - Create a user account with a username and password.
    - Connect to a Wi-Fi network or Ethernet.

5. Update and Upgrade the System

    - Once the setup is complete and we are logged in, open the terminal and update the system:

```
sudo apt update
sudo apt upgrade
```

6. Install Dependencies and Libraries

   - Install necessary packages and libraries for setting up YOLOv8 and other components.

     - Install Python and pip:

       ```
       sudo apt install python3-pip
       sudo apt install python3-dev
       ```

     - Install other essential libraries:

       ```
       sudo apt install libopenblas-dev libblas-dev libatlas-base-dev
       sudo apt install libjpeg8-dev libpng-dev
       ```

7. Enable GPU Acceleration

   - To use GPU acceleration with TensorFlow or PyTorch, ensure that the necessary NVIDIA libraries are installed:

     ```
     sudo apt install nvidia-jetpack
     ```

At this point, the Jetson Nano 2GB is set up with the basic OS and libraries, ready for further software installations.

Step 3: Model Training on our system

1. Set Up YOLOv8 Environment
2. Training YOLOv8

   - Train the model on your dataset on epochs 100:

```
results =
model.train(data='C:\\Users\\devri\\OneDrive\\Desktop\\final_iot\\data.yam
l', epochs=100, imgsz=640,workers=2)
```

Step 4: Deploy YOLOv8 on Jetson Nano

1. Prepare the Environment

- To run YOLOv8 efficiently on the Jetson Nano, we directly imported the YOLOv8 library into our Python environment instead of using Docker. This approach allowed us to maintain a lightweight setup tailored to the Jetson Nano's capabilities.

2. Upload the Trained Model
   - After training the YOLOv8 model on our local machine, we transferred the best-trained model weights to the Jetson Nano. This ensured the model was ready for real-time inference.
3. Run YOLOv8 for Inference
   - With the environment set up and the model weights in place, we utilized the YOLOv8 library to perform inference directly on real-time data from a USB camera.
4. Start YOLOv8 for Inference
   - Using the YOLOv8 library, we ran the following script to perform object detection on the real-time camera feed:

```
from ultralytics import YOLO
model = YOLO('best.pt')
results = model.predict(source=0, show=True)
```

The script initiated video streaming from the camera, allowing YOLOv8 to perform real-time object detection. When an accident was detected, the system sent alerts to the Flask server for further processing, integrating detection with server communication seamlessly.

Step 5: Flask Server Setup

1. Install Flask

```
pip3 install flask
```

2. Create Flask Application
   - Create a Python script (`app.py`).
3. **Run the Flask Server**
   - Start the server:

```
python3 app.py
```

Step 6: Integration and Testing

1. Integrate Detection with Flask

    - Modify YOLOv8 code to send an alert to the Flask server when an accident is detected.

2. Run Full Setup

    - Run the YOLOv8 model and ensure the Flask server receives alerts when accidents are detected.

3. Testing

    - Simulate accident scenarios to validate the system's accuracy and ensure real-time alerts.

4. Multithreading Implementation: To keep running the continuous accident detection and send the alert at same time, we used multithreading.

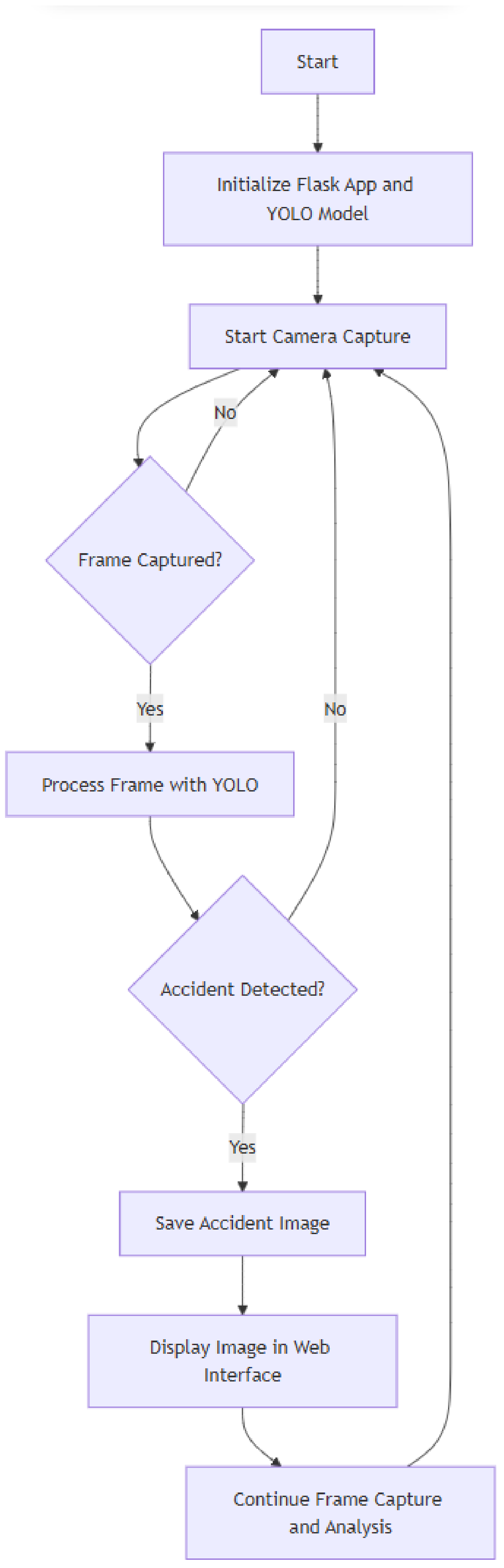


Fig: Real-Time Accident Detection System Workflow

CHAPTER 8: RESULTS

Metric	Value
Precision	0.9046
Recall	0.8785
mAP@50	0.9217
mAP@50-95	0.6883
Fitness	0.7116

Table: Accident Detection Model Performance Metrics

1. Precision (0.9046): 90.46% of projected accidents/vehicles were true, demonstrating high confidence in positive forecasts.

2. Recall (0.8785): The model successfully detected 87.85% of actual accidents/vehicles, but missed 12.15% (false negatives).

3. mAP@50 (0.9217): 92.17% mean average precision at an IoU threshold of 0.5, indicating high object detection and localisation.

4. mAP@50-95 (0.6883): 68.83% mAP spanning IoU levels of 0.5 to 0.95, demonstrating acceptable performance but space for improvement at tougher thresholds.

5. Fitness (0.7116): A combined metric score of 71.16% indicates acceptable model performance with space for improvement.

Evaluation curves:

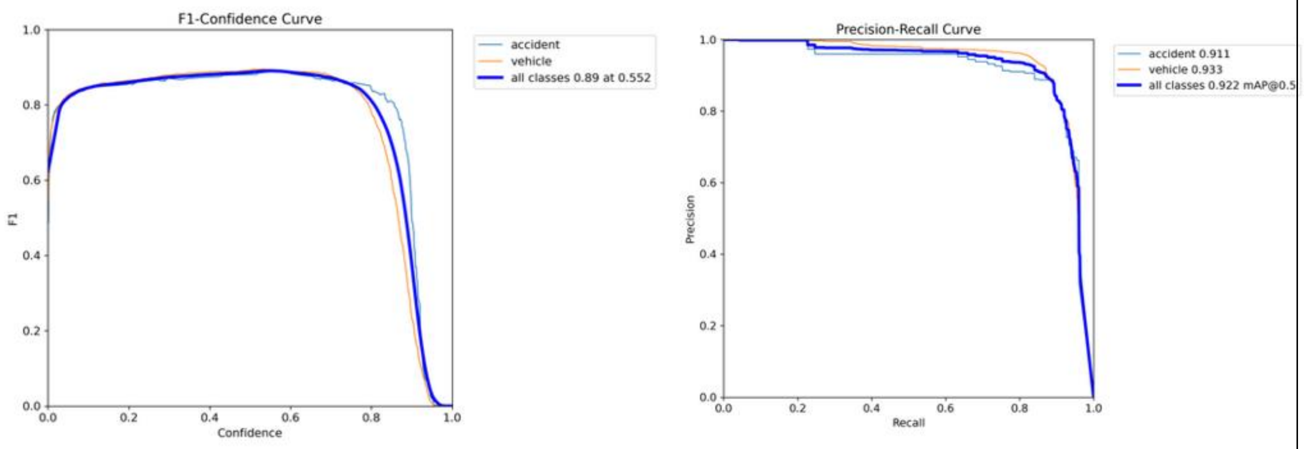


Fig:F1-Confidence and Precison-Recall Curve over test set

1. **High Precision:** Accident and vehicle classes have high precision (~0.91 and 0.93), indicating excellent detection with minimal false positives.

2. **Drop at High Recall:** As recall exceeds 0.9, precision decreases, indicating that the model gets more sensitive but includes more false positives.

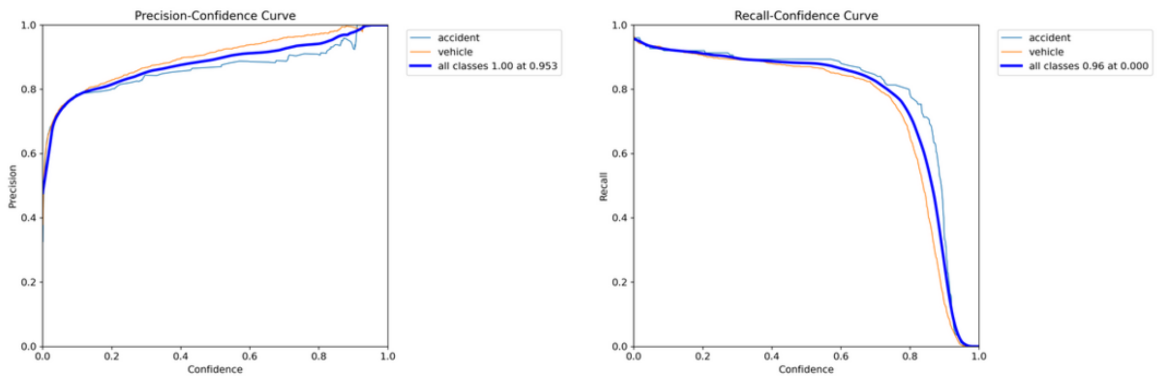


Fig: Precison-Confidence Recall-Confidence curve

1. **Precision Increases with Confidence**: As confidence increases, precision improves for both accident and vehicle classes, with fewer false positives.
2. **Recall Decreases with Confidence**: Higher confidence leads to lower recall, as the model becomes more selective, missing more detections (false negatives).

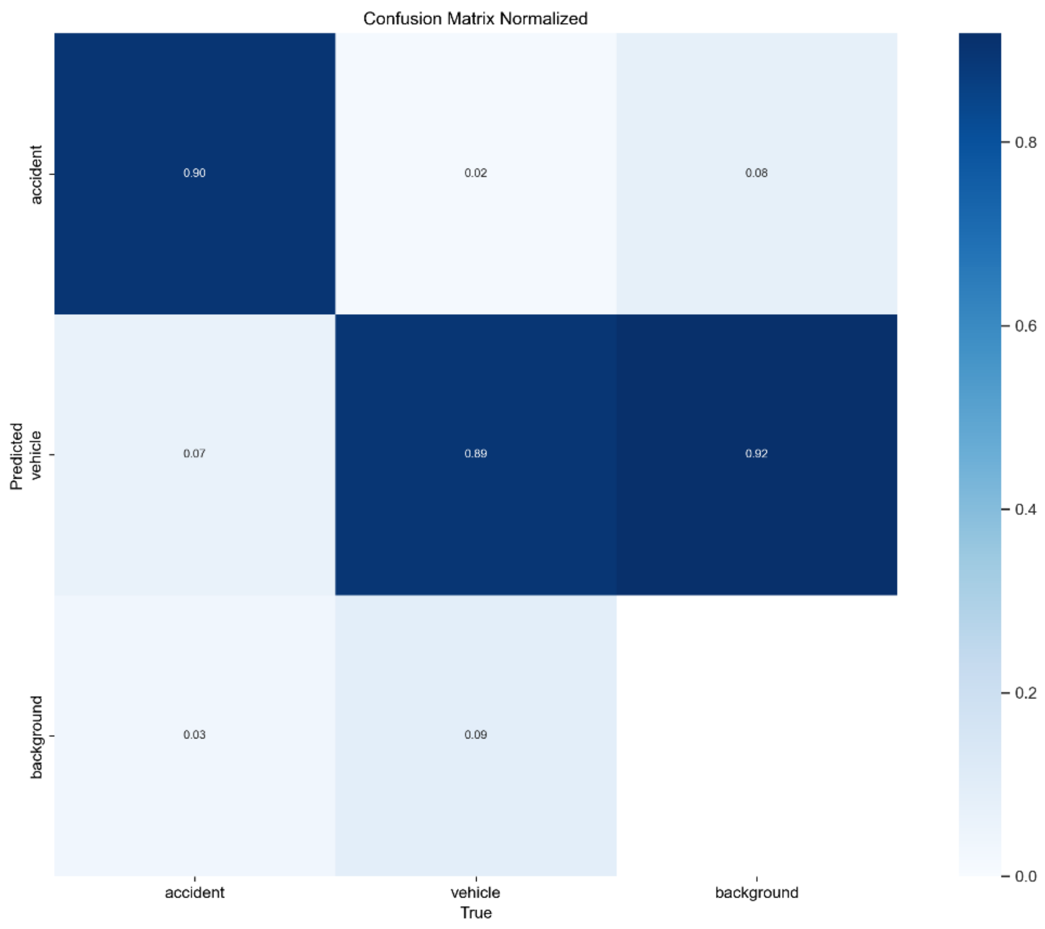
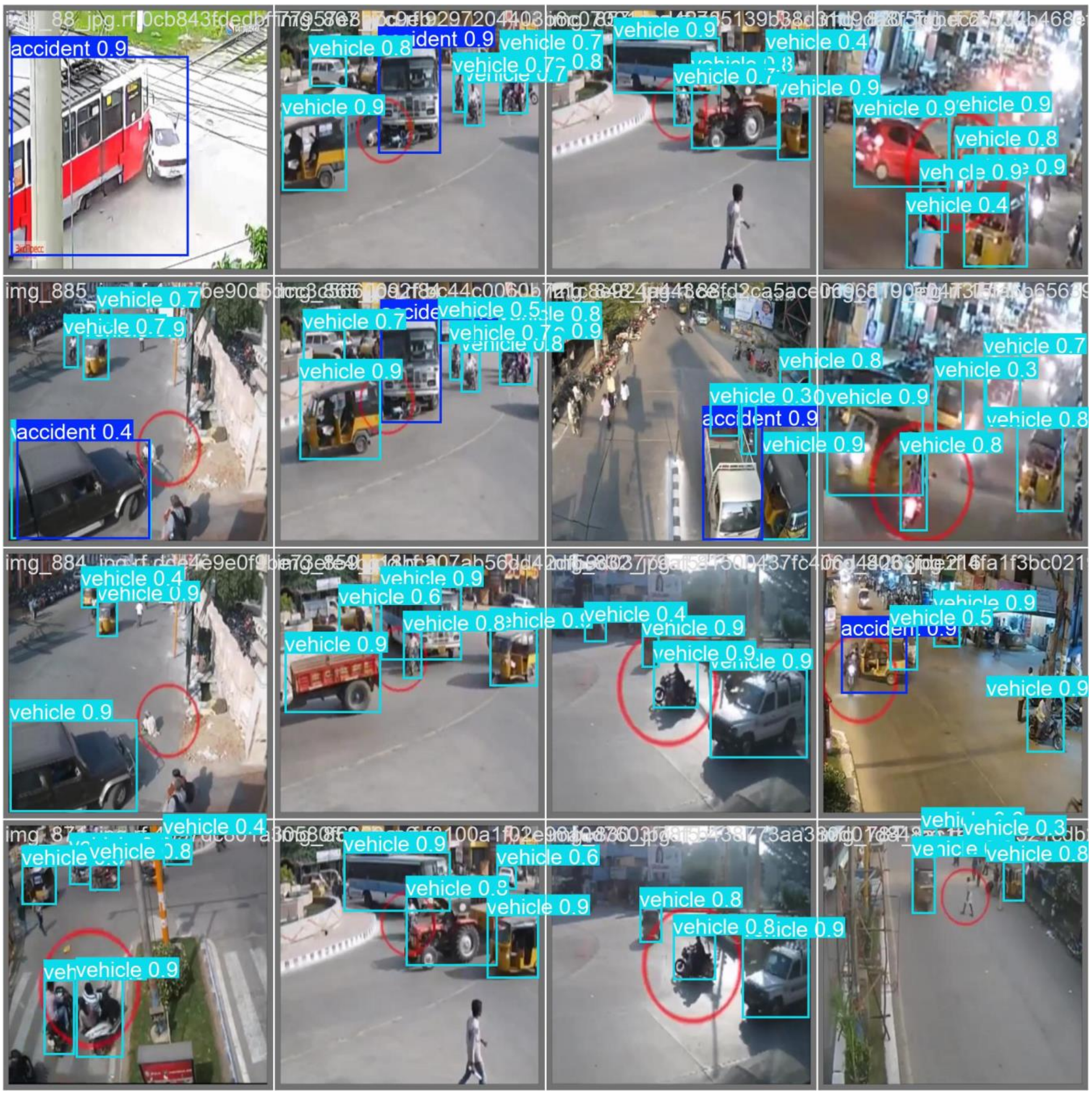
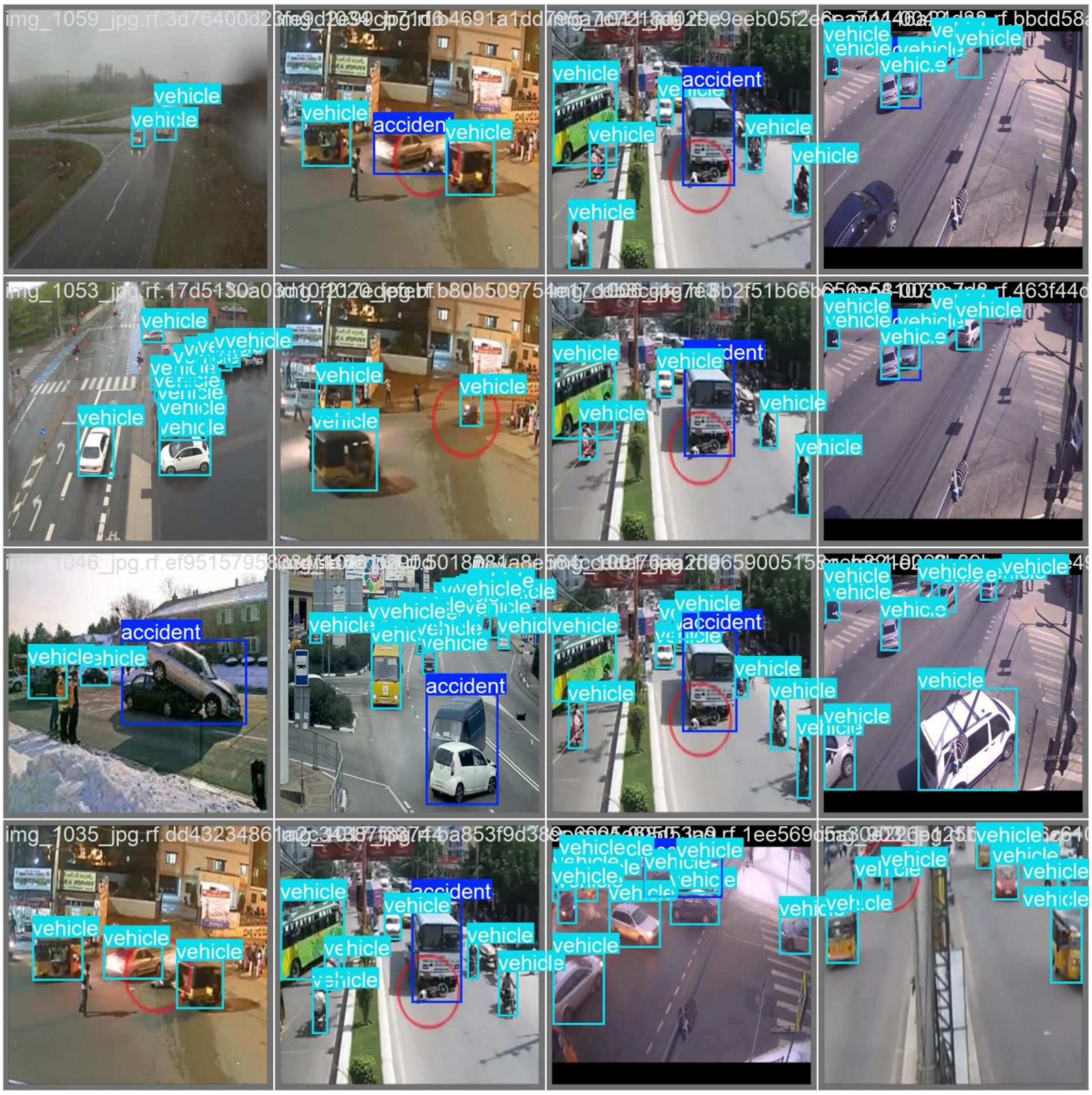


Fig: Confusion Matrix

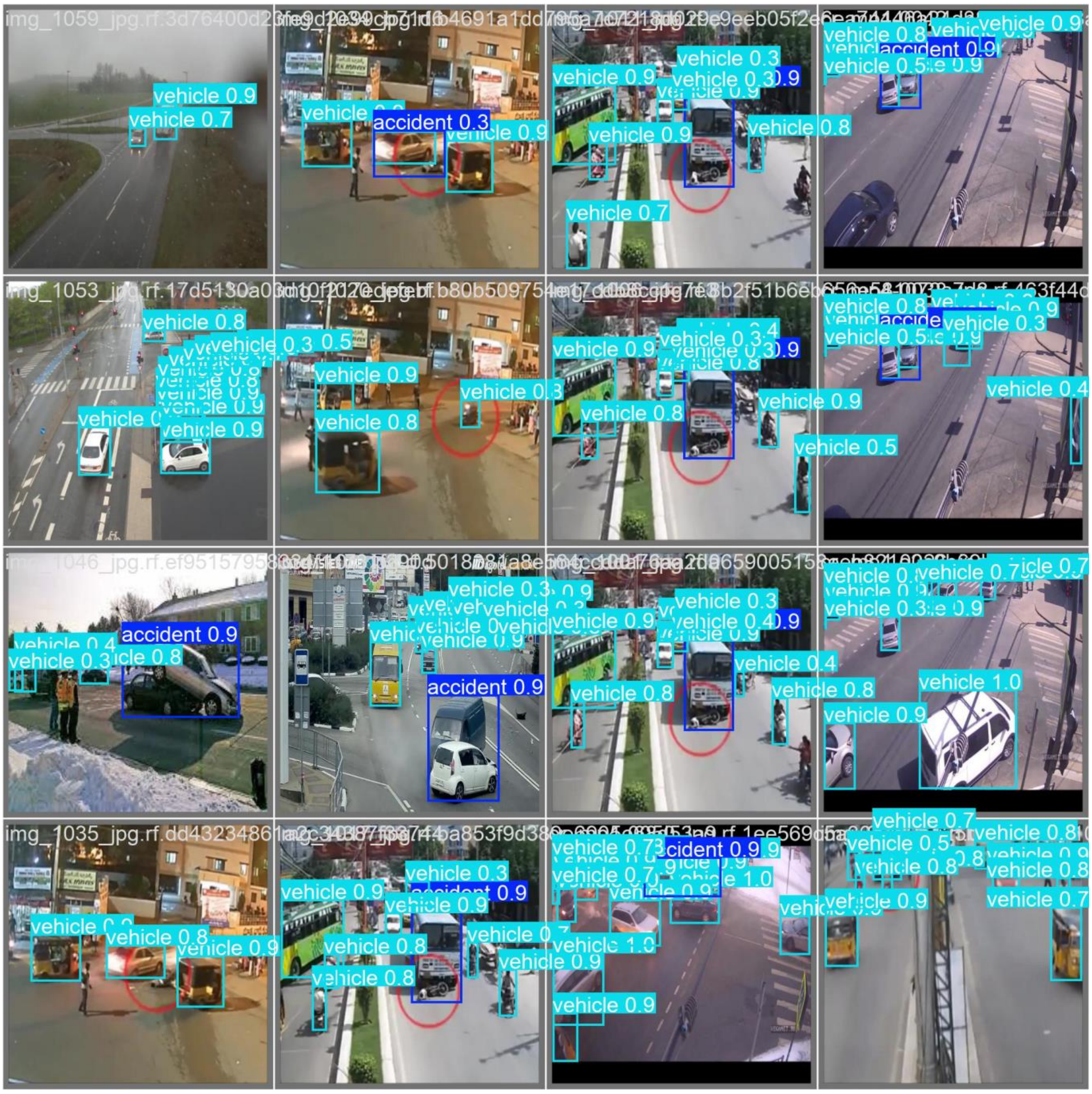
1. **Accident Detection:** The model identifies accidents correctly 90% of the time, which is strong. However, it sometimes mistakes accidents for the background (8%) or vehicles (2%), which could be improved.
2. **Vehicle Detection:** Vehicles are detected accurately 92% of the time, showing excellent performance. Occasionally, though, vehicles are confused with accidents (7%).
3. **Background Detection:** The model correctly identifies background elements 89% of the time, which is solid. However, it sometimes mistakes the background for accidents (3%) or vehicles (9%), indicating some confusion with foreground objects.

Fig: All Figure below are output images got by running the trained model onto the test dataset. Below are the images with the annotation/ respective labels that model recognized.

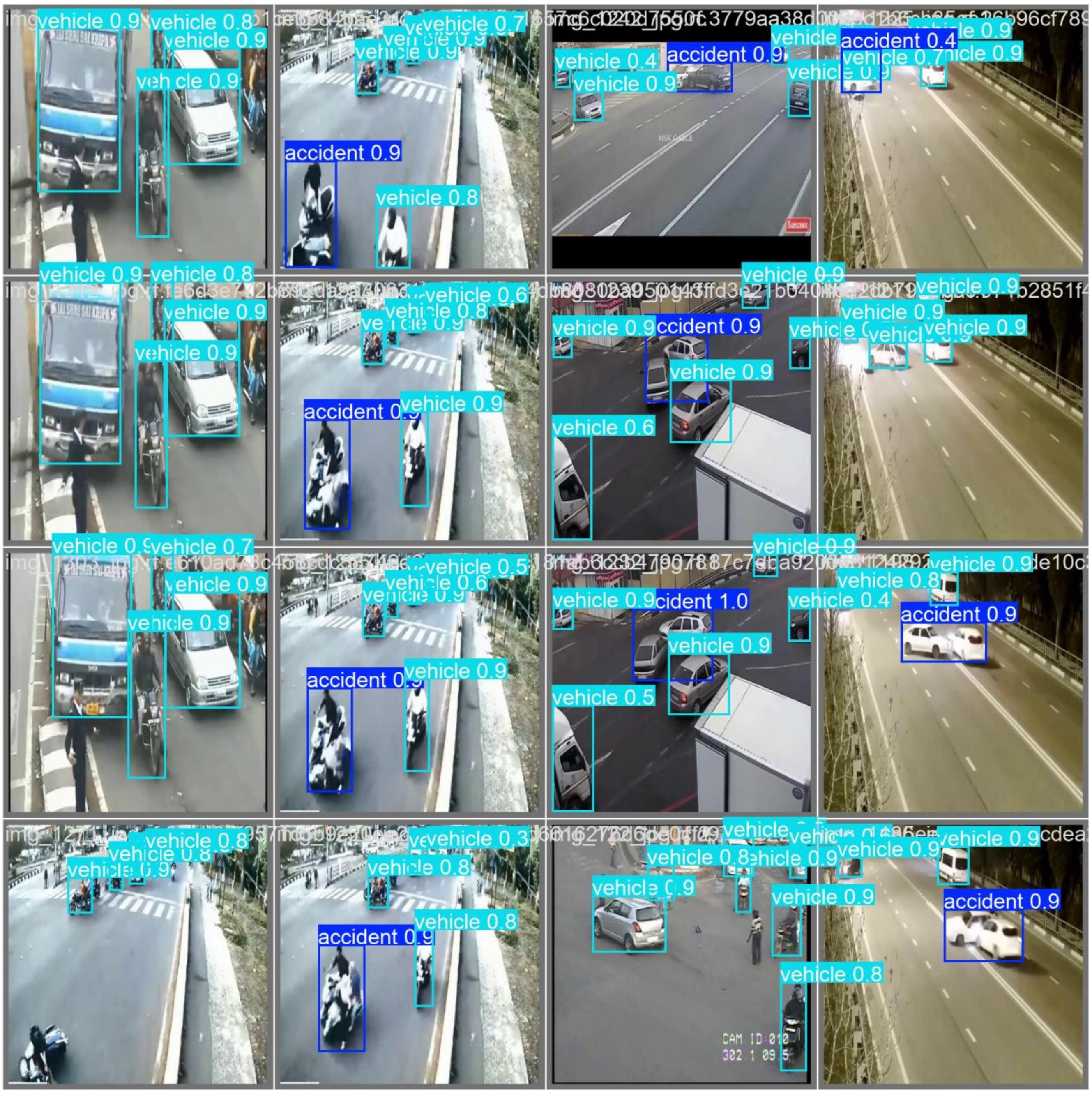




accident
vehicle


vehicle 0.9
vehicle 0.7
accident 0.3
accident 0.9
vehicle 0.8
vehicle 0.3
vehicle 0.5
vehicle 0.4
vehicle 1.0

vehicle
accident


vehicle 0.9
vehicle 0.8
accident 0.9
accident 0.4
accident 1.0
vehicle 0.5

Link to Video: Project_Demonstration.mp4

Our model is currently making some mistakes, like identifying non-accident images as accidents. This happens because our dataset is smaller than what's really needed for the model to perform at its best. We built the dataset by manually going through video frames and annotating them one by one, which was a big effort. While this allowed the model to start detecting accidents, it's still not as reliable as we'd like but it is generalizing with respect to different scenarios.

To improve accuracy and make the model more precise, we need to work with a larger and more diverse dataset. With a better dataset, the model will not only become more accurate but could also open up a world of possibilities for even more powerful applications.

CHAPTER 9: CONCLUSION

In conclusion, this project demonstrated the successful development of a real-time accident detection system leveraging YOLOv8 on the Jetson Nano. By processing live video feeds and accurately identifying accidents and vehicles in dynamic environments, the system proved both effective and practical. The integration of Flask-based server communication further enhanced its utility by enabling real-time alerts for immediate action in critical scenarios.

Through this work, we gained valuable insights into the importance of high-quality dataset preparation and annotation, as well as the need for optimizing deep learning models for resource-constrained environments like edge devices. Additionally, we learned the significance of seamless system integration to ensure functionality and reliability. This project highlights how advanced AI models combined with efficient edge computing solutions can contribute to meaningful advancements in safety-critical applications.

CHAPTER 10: LEARNING OUTCOMES

1. Practical Implementation of YOLOv8 for Real-Time Detection

- We gained hands-on experience in deploying YOLOv8, a state-of-the-art object detection model, on an edge device like the Jetson Nano. This involved managing real-time video feeds and performing inference to efficiently detect accidents in dynamic environments.

2. Edge Computing with Jetson Nano

- We explored the process of setting up the Jetson Nano for running deep learning models, utilizing its GPU for acceleration. Additionally, we learned how to integrate Docker for environment management and containerization, simplifying deployment and enhancing application portability.

3. Data Preparation and Model Training

- We developed a solid understanding of preparing datasets for object detection tasks. Using LabelImg, we annotated data to train YOLOv8 models, fine-tuning them to detect vehicles and accidents effectively. This preparation was crucial for achieving accurate and reliable model predictions.

4. Efficient Use of Resources for Deployment

- We optimized deep learning models for deployment in low-resource environments like the Jetson Nano. By carefully balancing resource consumption and performance, we ensured the models operated reliably without significant compromises.

5. Server Communication for Alerts

- We set up server-side communication using Flask to send alerts when accidents were detected. This helped us understand the integration of system components and enabled efficient data transmission to external servers for real-time alerts.

6. Real-Time Object Detection and Alert System

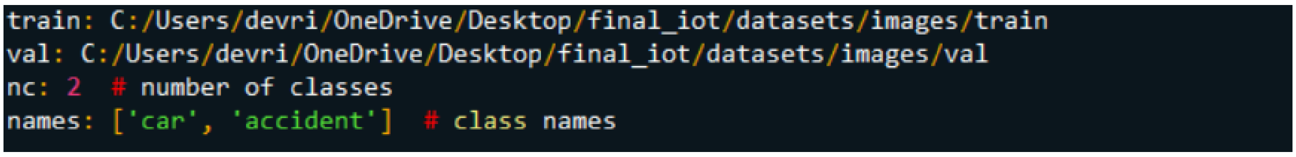
- We explored how real-time object detection can be applied to practical scenarios like accident detection. This project highlighted the potential of such systems in improving response times and enhancing safety measures in critical situations.

REFERENCES

1. *Accident Detection through CCTV Surveillance*. (2022, July 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9887656/
2. *Road Accident Detection from CCTV Footages using Deep Learning*. (2022, October 20). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9951920
3. *Accident detection from CCTV footage*. (2020, July 29). Kaggle. https://www.kaggle.com/datasets/ckay16/accident-detection-from-cctv-footage
4. *Accident detection model Object Detection Dataset and Pre-Trained Model by Accident detection model. (2024, April 8). Roboflow. https://universe.roboflow.com/accident-detection-model/accident-detection-model*
5. *CADP:A Novel Dataset for CCTV Traffic Camera based Accident Analysis. (n.d.). https://ankitshah009.github.io/accident_forecasting_traffic_camera*
6. *NVIDIA Jetson Nano 2GB Developer Kit - Get Started. (n.d.). NVIDIA Developer. https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit*
7. *Ratnayake, A. (2023, February 6). Set up Jetson Nano 2GB Developer Kit using Windows Computer. Medium. https://medium.com/@ashanvabs/set-up-jetson-nano-2gb-developer-kit-using-windows-computer-9b4e271df151*
8. *Digital image processing and analysis : human and computer vision applications with CVIPtools / author, Scott E Umbaugh.*
9. *YOLOV8: a new State-of-the-Art Computer Vision model. (n.d.). https://yolov8.com/*
10. *Ultralytics. (2024, October 19). Docker Image. Ultralytics YOLO Docs. https://docs.ultralytics.com/yolov5/environments/docker_image_quickstart_tutorial/*
11. *Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.*

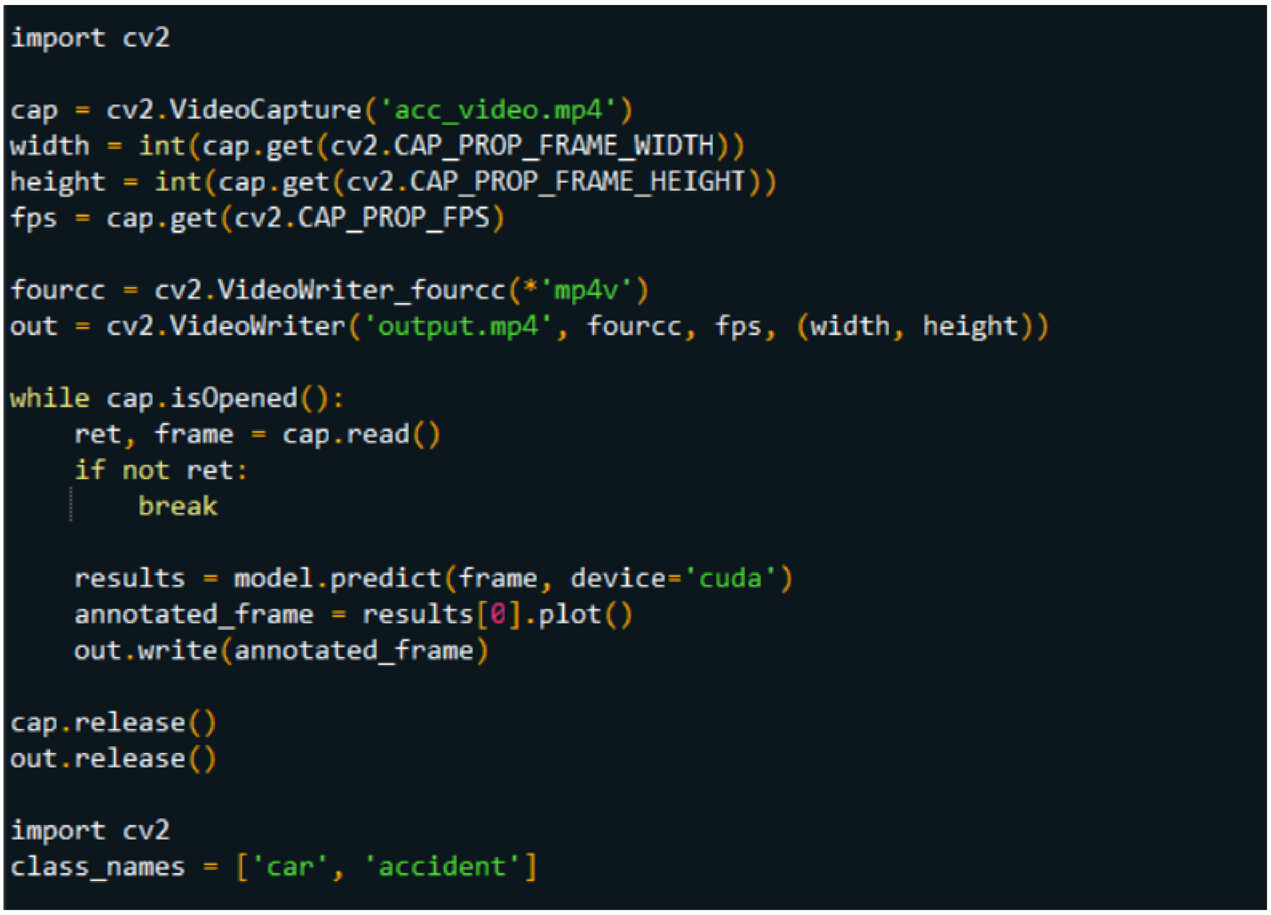
12. *Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
13. *Li, Y., & Jiao, L. (2019). Real-time Object Detection with Deep Learning: Challenges and Opportunities. IEEE Transactions on Neural Networks and Learning Systems, 30(3), 567-578.*
14. *Zhang, H., & Li, H. (2020). Development of a Real-Time Traffic Accident Detection System using Deep Learning and Camera Surveillance. IEEE Intelligent Transportation Systems Conference (ITSC).*

APPENDIX

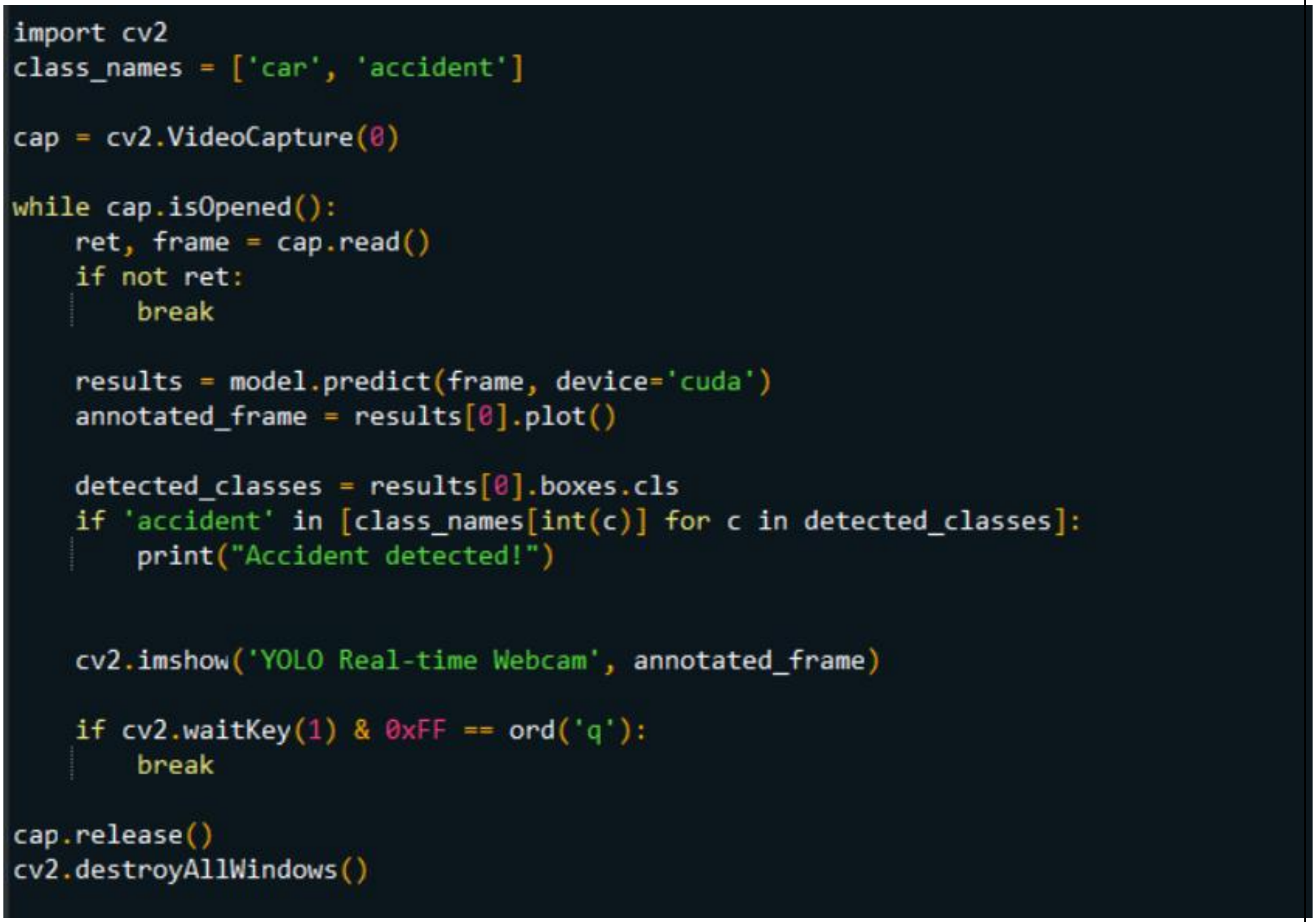


```
train: C:/Users/devri/OneDrive/Desktop/final_iot/datasets/images/train
val: C:/Users/devri/OneDrive/Desktop/final_iot/datasets/images/val
nc: 2  # number of classes
names: ['car', 'accident']  # class names
```

Code Breakdown: data.yaml extension file for model training data access.



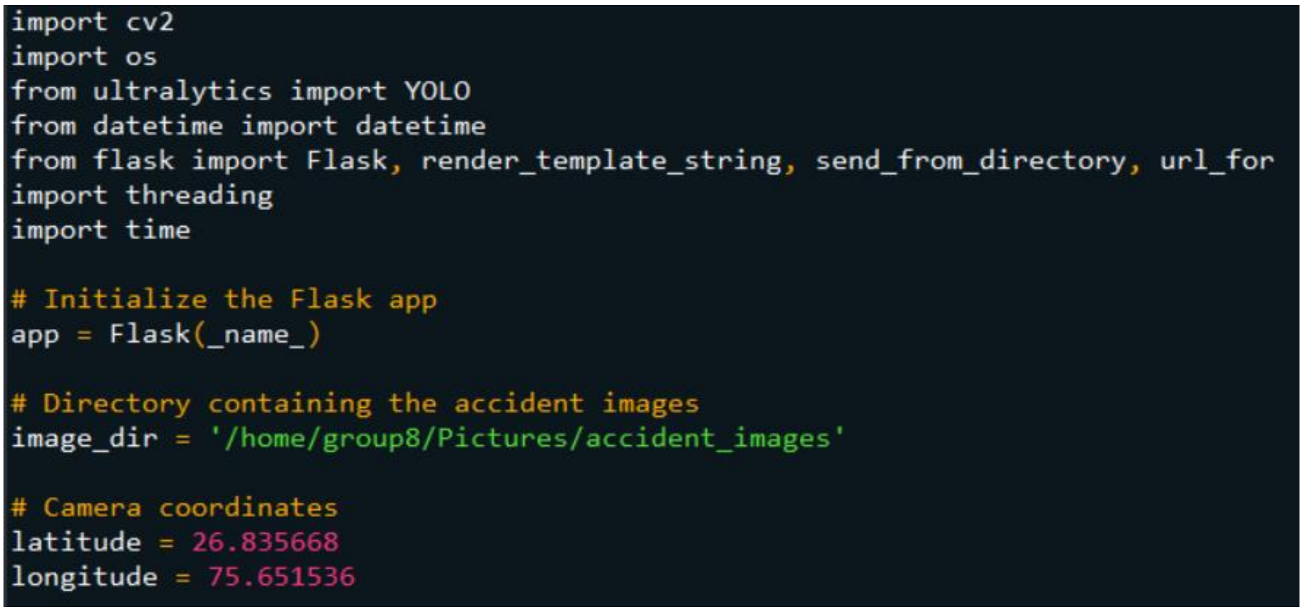
```
import cv2

cap = cv2.VideoCapture('acc_video.mp4')
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output.mp4', fourcc, fps, (width, height))

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model.predict(frame, device='cuda')
    annotated_frame = results[0].plot()
    out.write(annotated_frame)

cap.release()
out.release()

import cv2
class_names = ['car', 'accident']
```



```python
import cv2
class_names = ['car', 'accident']

cap = cv2.VideoCapture(0)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model.predict(frame, device='cuda')
    annotated_frame = results[0].plot()

    detected_classes = results[0].boxes.cls
    if 'accident' in [class_names[int(c)] for c in detected_classes]:
        print("Accident detected!")


    cv2.imshow('YOLO Real-time Webcam', annotated_frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Code Breakdown:

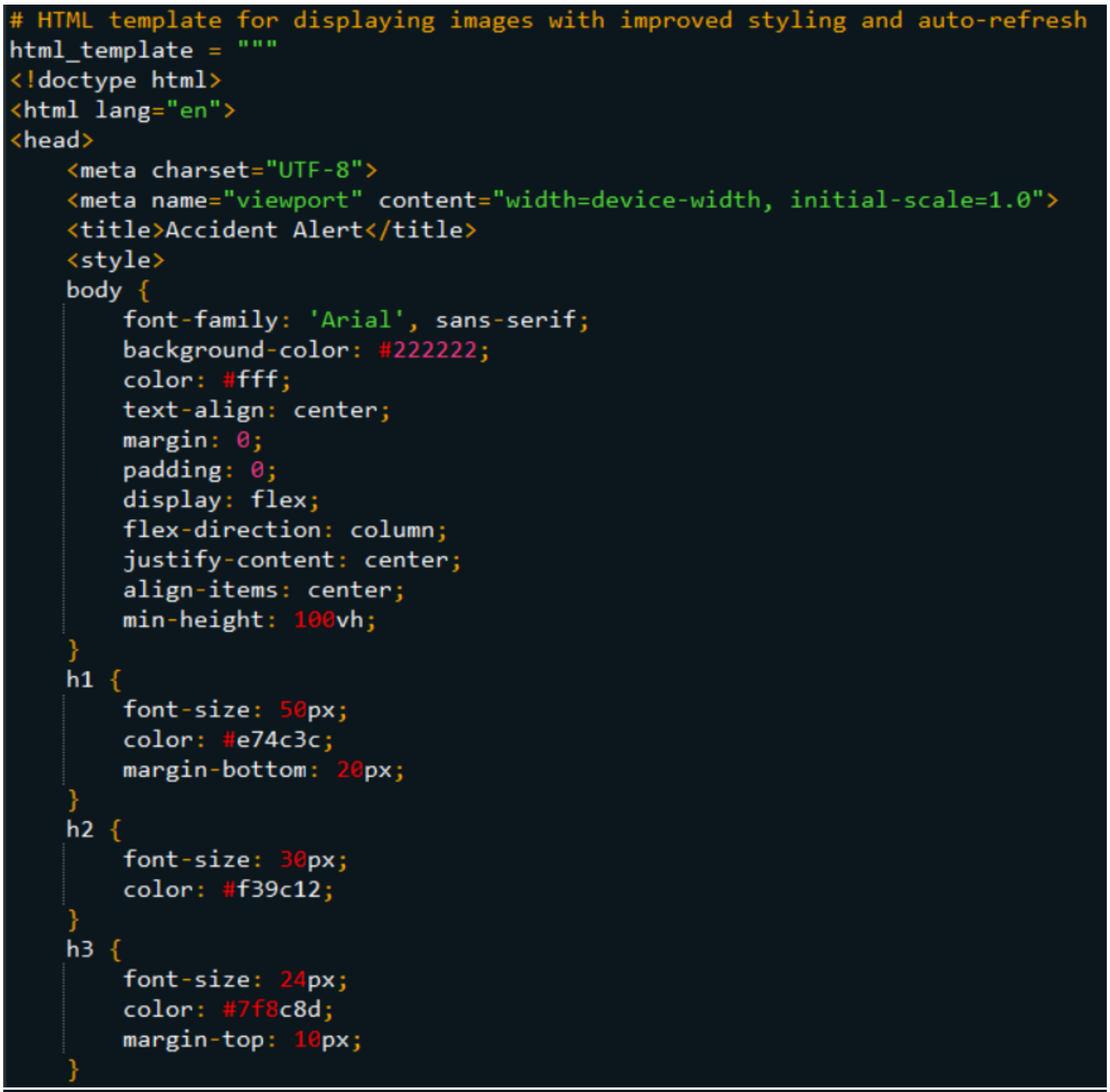
1. Generationg Output video on test dataset by name of output video.
2. Ongoing Accident Detection and on detection of accident print accident detection.



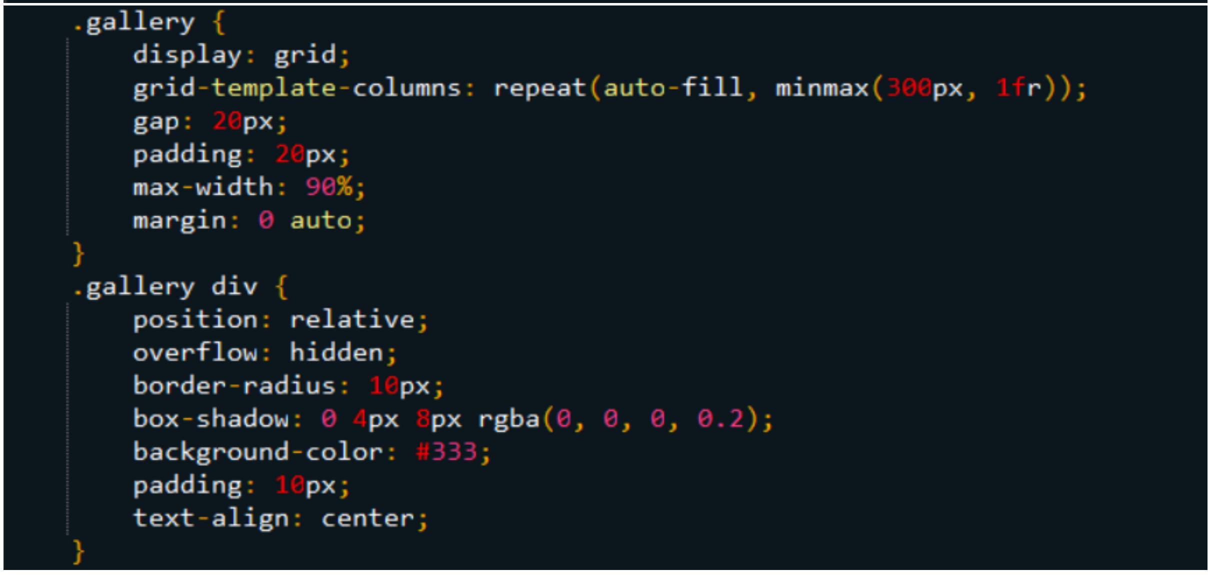
```python
import cv2
import os
from ultralytics import YOLO
from datetime import datetime
from flask import Flask, render_template_string, send_from_directory, url_for
import threading
import time

# Initialize the Flask app
app = Flask(_name_)

# Directory containing the accident images
image_dir = '/home/group8/Pictures/accident_images'

# Camera coordinates
latitude = 26.835668
longitude = 75.651536
```

Code Breakdown:

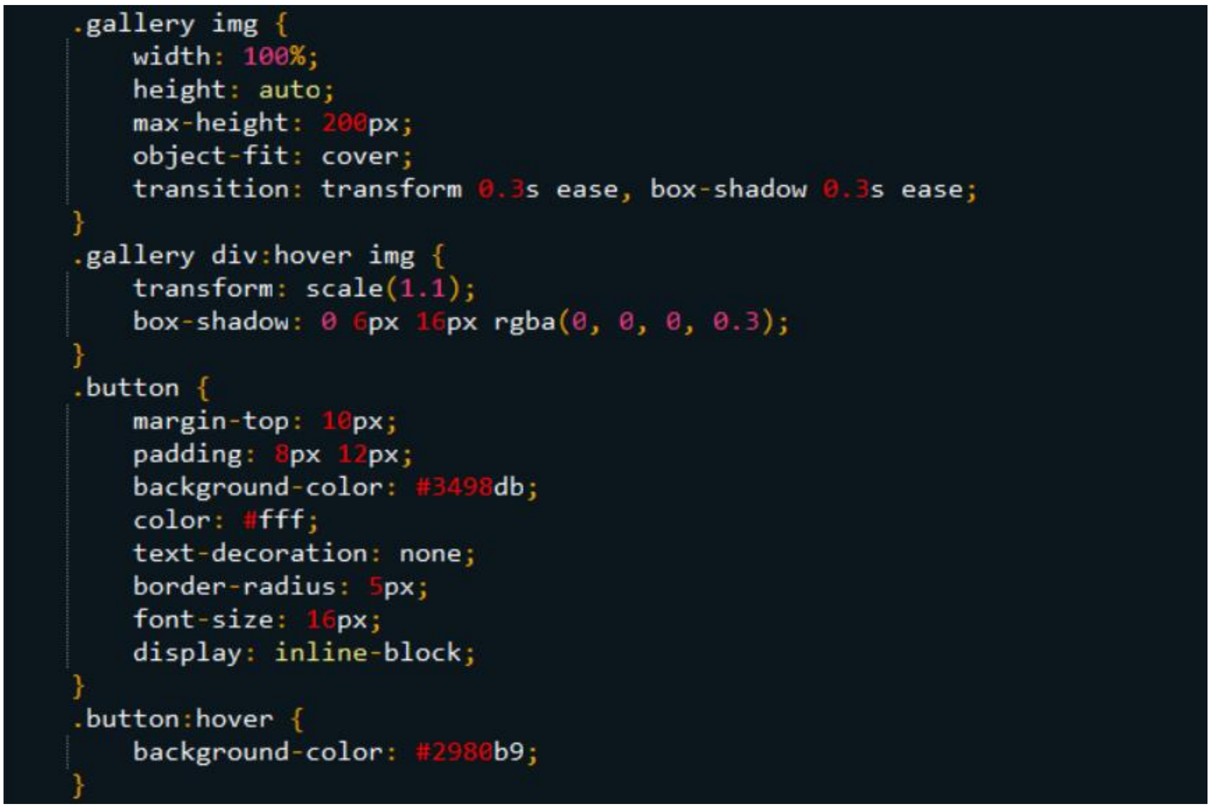
1. Flask Application Initialization:
   The Flask object is initialized using Flask(__name__), which creates the web application instance.
2. Image Directory Setup:
   The directory path /home/group8/Pictures/accident_images specifies where detected accident images will be saved and retrieved.
3. GPS Coordinates:
   latitude and longitude store the fixed location of the camera, which could be used for geotagging or reporting accidents.



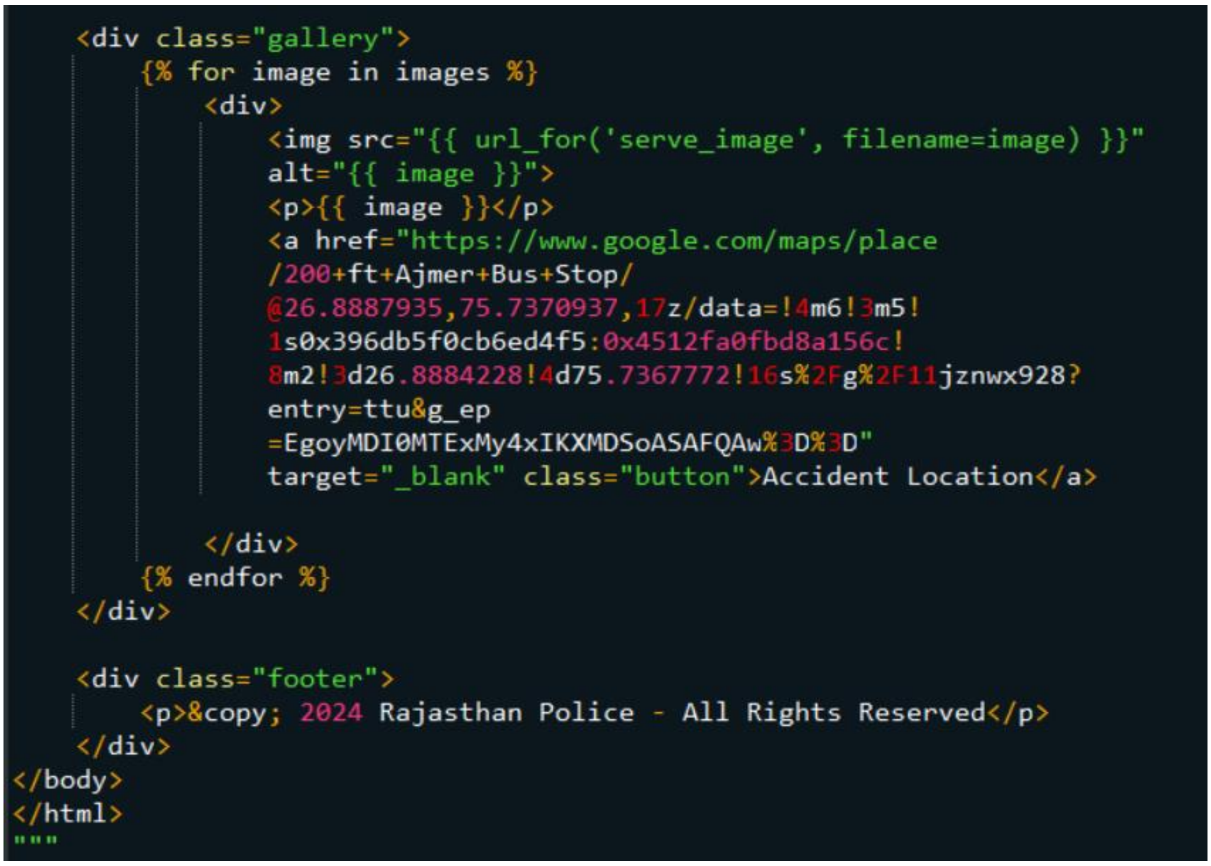
```
# HTML template for displaying images with improved styling and auto-refresh
html_template = """
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Accident Alert</title>
    <style>
    body {
        font-family: 'Arial', sans-serif;
        background-color: #222222;
        color: #fff;
        text-align: center;
        margin: 0;
        padding: 0;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        min-height: 100vh;
    }
    h1 {
        font-size: 50px;
        color: #e74c3c;
        margin-bottom: 20px;
    }
    h2 {
        font-size: 30px;
        color: #f39c12;
    }
    h3 {
        font-size: 24px;
        color: #7f8c8d;
        margin-top: 10px;
    }
```



```
    .gallery {
        display: grid;
        grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
        gap: 20px;
        padding: 20px;
        max-width: 90%;
        margin: 0 auto;
    }
    .gallery div {
        position: relative;
        overflow: hidden;
        border-radius: 10px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
        background-color: #333;
        padding: 10px;
        text-align: center;
    }
```



```
    .gallery img {
        width: 100%;
        height: auto;
        max-height: 200px;
        object-fit: cover;
        transition: transform 0.3s ease, box-shadow 0.3s ease;
    }
    .gallery div:hover img {
        transform: scale(1.1);
        box-shadow: 0 6px 16px rgba(0, 0, 0, 0.3);
    }
    .button {
        margin-top: 10px;
        padding: 8px 12px;
        background-color: #3498db;
        color: #fff;
        text-decoration: none;
        border-radius: 5px;
        font-size: 16px;
        display: inline-block;
    }
    .button:hover {
        background-color: #2980b9;
    }
```

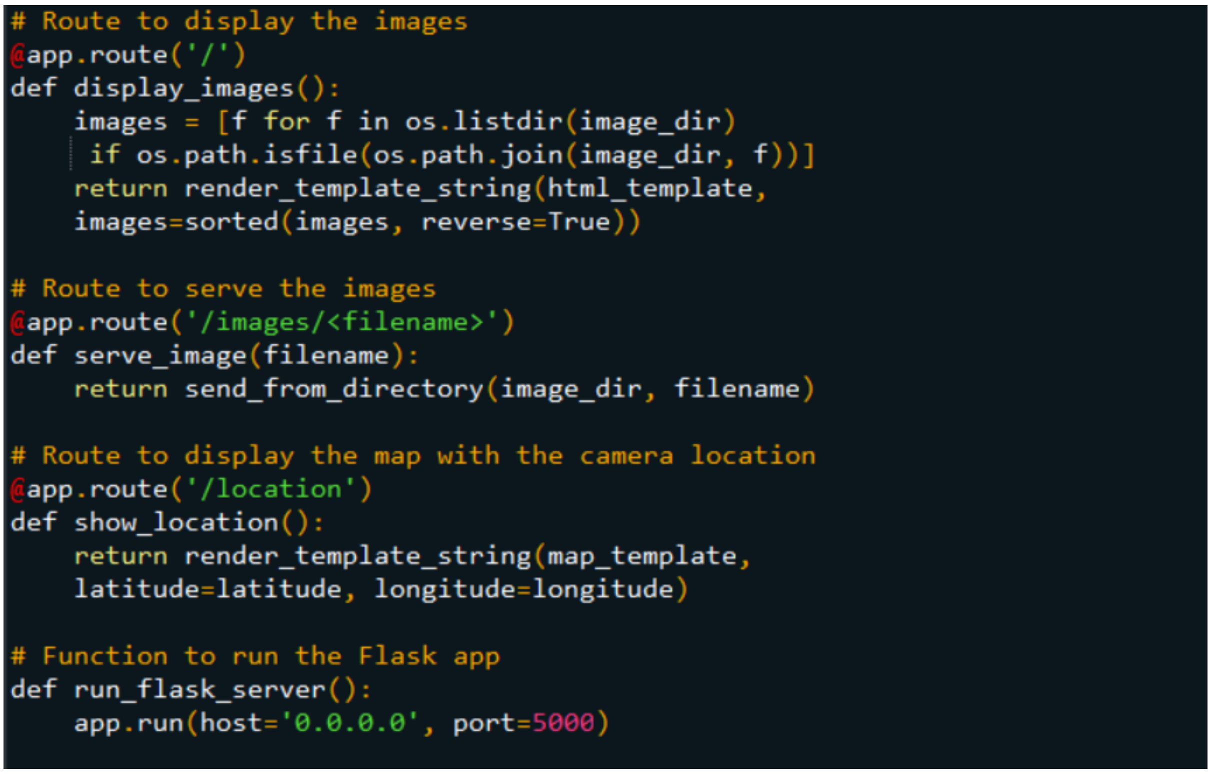


```
    <div class="gallery">
        {% for image in images %}
            <div>
                <img src="{{ url_for('serve_image', filename=image) }}"
                alt="{{ image }}">
                <p>{{ image }}</p>
                <a href="https://www.google.com/maps/place
                /200+ft+Ajmer+Bus+Stop/
                @26.8887935,75.7370937,17z/data=!4m6!3m5!
                1s0x396db5f0cb6ed4f5:0x4512fa0fbd8a156c!
                8m2!3d26.8884228!4d75.7367772!16s%2Fg%2F11jznwx928?
                entry=ttu&g_ep
                =EgoyMDI0MTExMy4xIKXMDSoASAFQAw%3D%3D"
                target="_blank" class="button">Accident Location</a>

            </div>
        {% endfor %}
    </div>

    <div class="footer">
        <p>&copy; 2024 Rajasthan Police - All Rights Reserved</p>
    </div>
</body>
</html>
"""
```

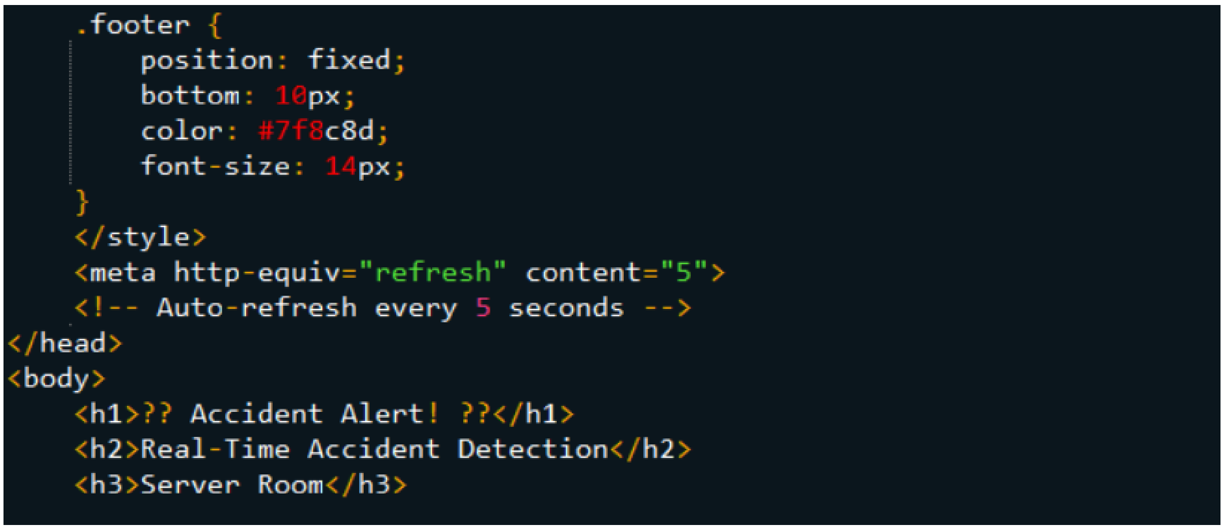
Code Breakdown:

The HTML and CSS code creates a user-friendly interface for the accident detection system by :

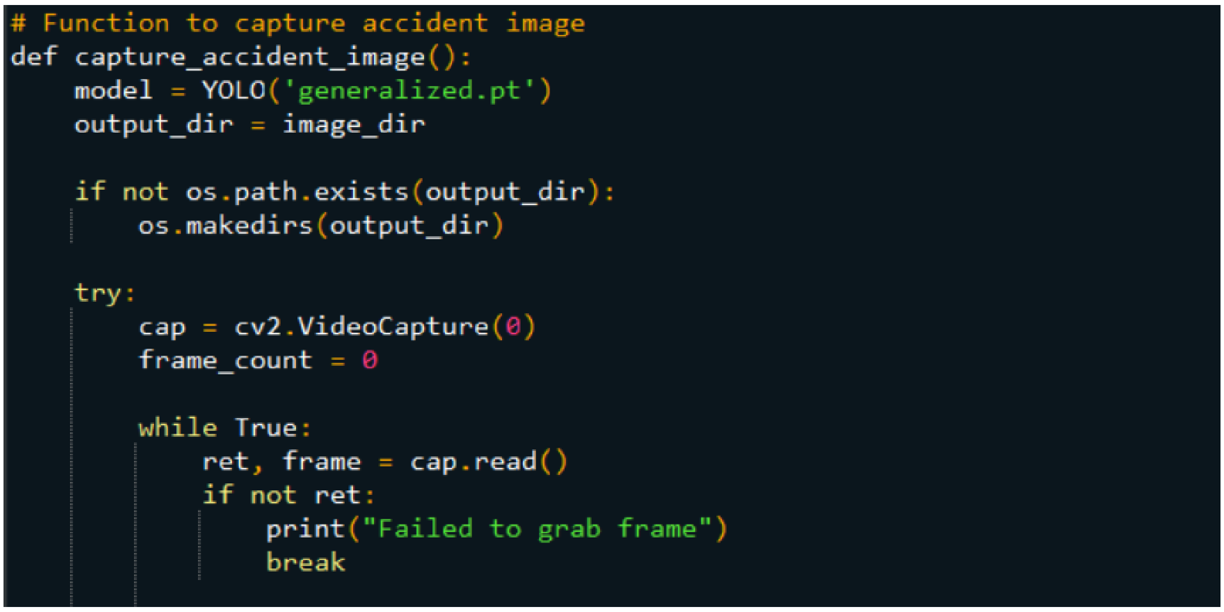
1. Displaying accident images in a responsive grid for easy viewing on all screen sizes.
2. Adding hover effects and buttons for smooth interactivity.
3. Ensuring a clear and professional design to help users quickly analyze information.



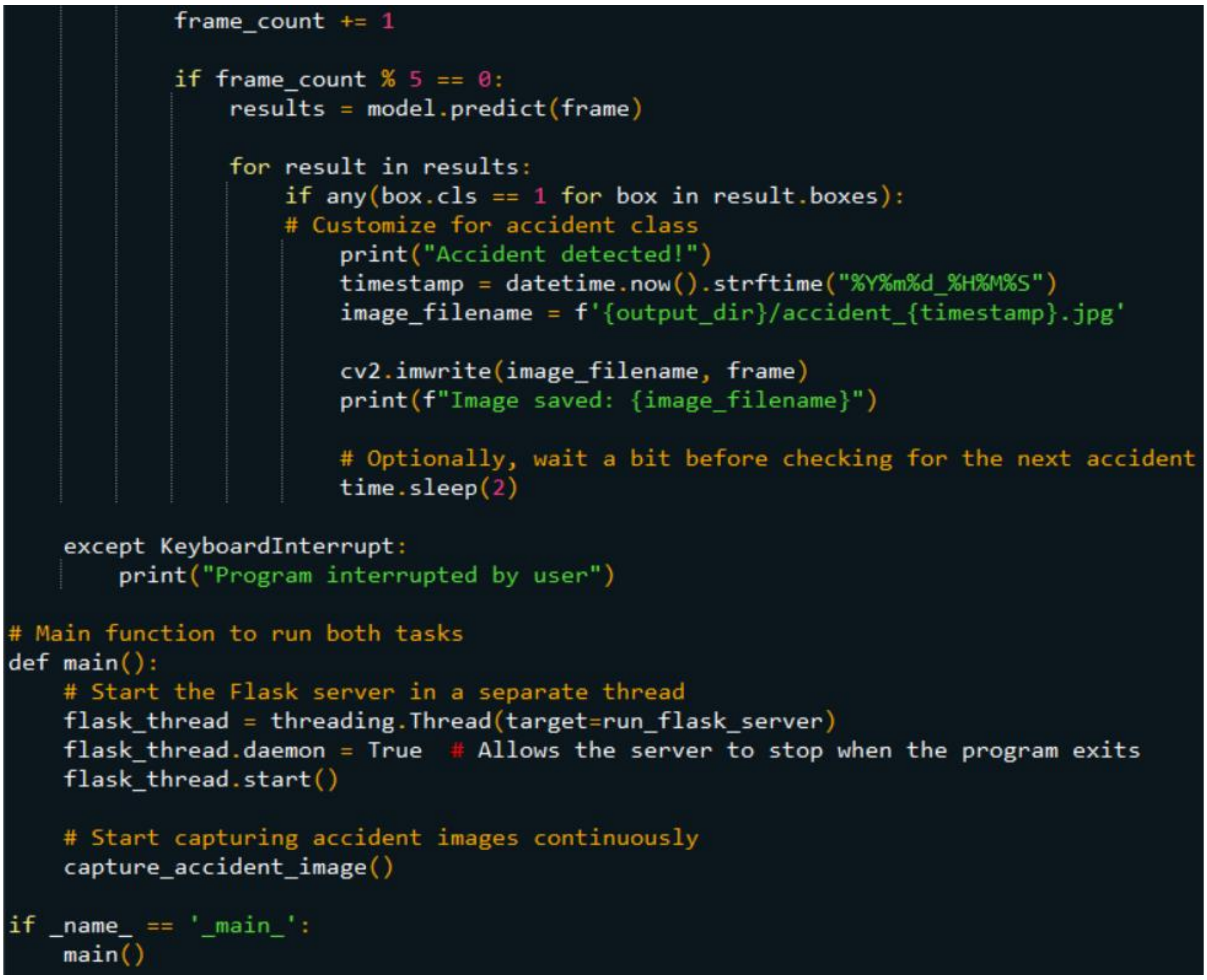
```
# Route to display the images
@app.route('/')
def display_images():
    images = [f for f in os.listdir(image_dir)
     if os.path.isfile(os.path.join(image_dir, f))]
    return render_template_string(html_template,
    images=sorted(images, reverse=True))

# Route to serve the images
@app.route('/images/<filename>')
def serve_image(filename):
    return send_from_directory(image_dir, filename)

# Route to display the map with the camera location
@app.route('/location')
def show_location():
    return render_template_string(map_template,
    latitude=latitude, longitude=longitude)

# Function to run the Flask app
def run_flask_server():
    app.run(host='0.0.0.0', port=5000)
```



```
    .footer {
        position: fixed;
        bottom: 10px;
        color: #7f8c8d;
        font-size: 14px;
    }
    </style>
    <meta http-equiv="refresh" content="5">
    <!-- Auto-refresh every 5 seconds -->
</head>
<body>
    <h1>?? Accident Alert! ??</h1>
    <h2>Real-Time Accident Detection</h2>
    <h3>Server Room</h3>
```



```python
# Function to capture accident image
def capture_accident_image():
    model = YOLO('generalized.pt')
    output_dir = image_dir

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    try:
        cap = cv2.VideoCapture(0)
        frame_count = 0

        while True:
            ret, frame = cap.read()
            if not ret:
                print("Failed to grab frame")
                break
```



```python
            frame_count += 1

            if frame_count % 5 == 0:
                results = model.predict(frame)

                for result in results:
                    if any(box.cls == 1 for box in result.boxes):
                    # Customize for accident class
                        print("Accident detected!")
                        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                        image_filename = f'{output_dir}/accident_{timestamp}.jpg'

                        cv2.imwrite(image_filename, frame)
                        print(f"Image saved: {image_filename}")

                        # Optionally, wait a bit before checking for the next accident
                        time.sleep(2)

    except KeyboardInterrupt:
        print("Program interrupted by user")

# Main function to run both tasks
def main():
    # Start the Flask server in a separate thread
    flask_thread = threading.Thread(target=run_flask_server)
    flask_thread.daemon = True  # Allows the server to stop when the program exits
    flask_thread.start()

    # Start capturing accident images continuously
    capture_accident_image()

if _name_ == '_main_':
    main()
```

Code Breakdown:

1. Web Interface with Flask: The Flask app serves a webpage where accident images are displayed in a grid format. It also shows the camera's location on a map using latitude and longitude.
2. Accident Detection: The YOLO model (generalized.pt) processes the video feed from a connected camera (via OpenCV). When the model detects an accident (based on a specific class), it saves the image with a timestamp in the designated directory.
3. Concurrent Operation: The Flask server runs in a separate thread to ensure smooth operation while the accident detection continues in the background. This allows the system to both detect and display accidents in real-time.