A Major-Project Report

On

TUNNEL ACCIDENT DETECTION USING YOLOv7

Submitted in partial fulfilment for the Degree of B. Tech.

In

Information Technology

By

A. Sri Vaishnavi [21911A1271]

G. Shiva Sai Reddy [21911A1288]

K. Vasantha [21911A1293]

K. Gayathri [21911A1294]

Under the guidance of

Dr. B. NANDITHA

Assistant Professor - Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

[Accredited by NAAC & NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH]

2024 – 2025

VIDYA JYOTHI
INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION

CERTIFICATE

This is to certify that the project report entitled "Tunnel Accident Detection Using YOLOv" **submitted by** ***A.Sri Vaishnavi [21911A1271],G. Shiva Sai Reddy [21911A1288], K. Vasantha [21911A1293], K. Gayathri [21911A1294]*** **to Vidya Jyothi Institute of Technology(An Autonomous Institution), Hyderabad, in partial fulfilment for the award of the degree of B. Tech. in Information Technology a** ***bonafide*** **record of project work carried out by us under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree.**

Signature of Supervisor	**Signature of Head of the Department**
Dr.B. NANDITHA	**Dr. A. Obulesh**
Assistant Professor	**Associate Professor**
Department of Informatio Technology	**Department of Information Technology**

Signature of External Examiner

DECLARATION

We declare that this project report titled Tunnel Accident Detection Using YOLOv7 submitted in partial fulfilment of the degree of B. Tech. in Information Technology is a record of original work carried out by us under the supervision of Dr. B. Nanditha , and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice of reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

A. Sri Vaishnavi [21911A1271]
G. Shiva Sai Reddy [21911A1288]
K. Vasantha [21911A1293]
K. Gayathri [21911A1294]

Date:

ACKNOWLEDGEMENTS

We extend our heartfelt thanks to **Dr. B. Nanditha,** Assistant Professor, for her invaluable mentorship and guidance throughout the project. Her expertise, encouragement, and insightful feedback significantly shaped our approach and methodology.

We are deeply grateful to Dr. A. Obulesu, Head of the Department, and all the faculty members of the department for their constant support and guidance throughout our academic journey.

Our sincere thanks to our beloved Dean of Accreditation and Rankings, Dr. A. Padmaja Madam, and Academic Co-originator, Smt. G. Srilatha Madam, whose unwavering support proved indispensable during the course.

We also express our heartfelt thanks to our Principal, Dr. A. Srujana madam, for her encouragement and continuous support throughout this endeavour.

We wish to express our heartful thanks to the Principal **Dr.A.Srujana**, whose support was indispensable to us during would like to sincerely thank our College Management for providing the necessary infrastructure and facilities that enabled us to carry out our work effectively.

We are thankful to our friends, well-wishers, for their valuable suggestions and constant willingness to help, which motivated us to give our best throughout this journey.

Finally, we would like to express our deepest gratitude to our parents , whose unwavering love, guidance, and support have been the foundation of our success and aspirations.

A.Sri Vaishnavi [21911A1271]
G. Shiva Sai Reddy [21911A1288]
K.Vasantha[21911A1293]
K.Gayathri [21911A1294]

ABSTRACT

This project Detecting accidents in tunnels is critical for ensuring the safety of commuters and maintaining the efficiency of transportation systems. This project proposes a deep learning-based approach utilizing YOLOv7 (You Only Look Once version 7) to identify accidents within tunnel environments. The methodology begins with importing essential libraries and acquiring a specialized dataset comprising tunnel accident images. The dataset undergoes thorough preprocessing to ensure consistency and quality, followed by a structured split into training, validation, and testing sets. This division is crucial to train the model effectively, fine-tune its parameters, and objectively evaluate its performance. The YOLOv7 model, known for its real-time object detection capabilities, is selected and trained on the prepared dataset, using provided annotations or bounding boxes to accurately localize and classify accident scenarios within the images.

Throughout the training process, progress is carefully monitored and logged to identify potential areas for improvement and ensure the model's optimal performance. Once trained, the model is evaluated using key performance metrics such as accuracy, precision, and recall, providing a comprehensive understanding of its ability to detect tunnel accidents reliably. The results demonstrate the promising potential of deep learning, particularly YOLOv7, in enhancing accident detection systems within tunnel environments.

Looking ahead, a major future enhancement for this project involves the integration of an automated emergency alert system. This system would be designed to instantly notify nearby hospitals, police stations, and emergency services whenever an accident is detected, ensuring rapid response times and potentially saving lives. Such integration would transform this detection system into a fully operational, life-saving solution for modern transportation infrastructures.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Introduction

Ensuring the safety of commuters in tunnels is paramount, and detecting accidents promptly is critical for effective response. This paper presents a deep learning-based approach utilizing YOLOv7 for accident identification in tunnel images. The proposed method encompasses importing essential libraries, acquiring and preprocessing the dataset, dividing it into training, validation, and testing sets, selecting and training the YOLOv7 model, and assessing its performance. The model is fine-tuned on a tunnel accident dataset to predict accidents based on supplied annotations or bounding boxes. The training progress is monitored and logged, and the model's performance is evaluated using metrics including accuracy, precision, and recall. This study highlights the potential of deep learning for accident detection in tunnels, paving the way for safer and more efficient transportation systems.

Ensuring commuter safety in tunnels is crucial. Detecting accidents quickly allows for effective emergency response. This project proposes using a deep learning approach YOLOv7 to detect accidents in tunnel images. It includes steps like importing libraries, data preprocessing, model training, and evaluation using metrics such as accuracy, precision, and recall.

Tunnel safety is a major concern worldwide, as tunnels have confined spaces with limited escape options, making accidents particularly dangerous. Rapid and accurate accident detection is crucial to reduce casualties and property loss. Traditional methods, such as manual CCTV monitoring, are often slow and inefficient, especially in low-light or unclear tunnel environments. To overcome these limitations, this project proposes an automated accident detection system using deep learning techniques. Specifically, the YOLOv7 model (You Only Look Once - Version 7) is fine-tuned to detect accidents in tunnel images. By utilizing object detection algorithms and real-time analysis, the system aims to significantly enhance safety measures in tunnels.

1.2 Problem Statement

The Existing tunnel accident detection systems, while helpful, have several limitations that reduce their effectiveness. Most notably, they require a massive amount of labeled data to train the deep learning models effectively, which is a costly and time-consuming process. Furthermore, these models often struggle with changes in lighting conditions, obstructions, and environmental variations inside tunnels, leading to inaccuracies in detection. Computational demands are another major concern, as real-time video processing requires powerful hardware resources that may not always be feasible. Moreover, accidents that occur outside of camera views or under very poor visibility conditions often go undetected. Thus, there is an urgent need for a robust, efficient, and accurate system that can overcome these challenges and ensure reliable accident detection in tunnels.

Partial Detection: Some systems fail to detect accidents occurring outside the camera frame or in blind spots. Thus, there is a need for a **robust, real-time, and efficient** accident detection system that performs reliably under varying conditions.

1.3 Existing Systems

Current systems for tunnel accident detection mainly use deep learning-based video analysis techniques. Typically, pre-processed still images extracted from video streams are fed into object detection models like Faster R-CNN or other convolutional neural networks. These models can classify objects such as cars, fires, or people and predict their location through bounding boxes. While effective to some extent, these systems suffer from several shortcomings. They heavily rely on the quality and quantity of the training datasets, and their accuracy drops significantly in poor lighting or when occlusions are present. Furthermore, real-time application remains a challenge due to the computational burden associated with processing high volumes of video data.

1.4 Objective

The objective of this project is to develop an intelligent system that can accurately and quickly detect accidents inside tunnels to minimize human casualties and property damage. The system aims to automate the monitoring of tunnel conditions, reducing dependency on manual surveillance and ensuring that emergencies are detected without delay. By integrating deep learning-based accident detection with real-time alert mechanisms, the project seeks to enhance tunnel safety significantly. Another important is to contribute towards the broader vision of smart transportation systems by enabling real-time accident detection as a part of intelligent infrastructure solutions. Ultimately, the project aspires to save lives, minimize economic loss, and promote safer travel through tunnels.

CHAPTER 2
LITERATURE SURVEY

2.1 Introduction

This project report titled **"Tunnel Accident Detection Using YOLOv7"** is submitted as a part of the curriculum requirement for the partial fulfilment of the award of the degree. The aim of this project is to develop an intelligent, efficient, and real-time accident detection system specifically designed for tunnel environments using deep learning techniques. By training and fine-tuning the YOLOv7 object detection model on tunnel accident datasets, we seek to address the challenges faced by existing manual surveillance and traditional detection systems. This report is organized into systematic chapters covering the background, literature review, system design, implementation, and testing phases. I hope that the study presented in this report contributes positively towards future developments in intelligent transportation and tunnel safety systems.

[1]. Alex, B., Zongyuan, Ge, Lionel Ott, Fabio Ramos, Ben Upcroft Proceedings of the IEEE International Conference on Image Processing (ICIP) In this paper, the authors address the challenge of multiple object tracking (MOT) for online and real-time systems. They propose a simple yet highly efficient approach that primarily focuses on object association using familiar techniques such as the Kalman Filter for motion prediction and the Hungarian algorithm for data association. An important observation from this study is that the quality of the object detector greatly influences the tracking performance, showing that up to an 18.9% improvement can be achieved by using better detectors. Even though the proposed tracker relies on basic algorithms without any complex deep learning models, it outperforms many existing state-of-the-art trackers in speed, achieving an update rate of 260 Hz. This paper highlights the trade-off between tracking complexity and system responsiveness, providing valuable insights for real-time accident detection systems where fast reaction times are critical.

[2] Davis, J., Goadrich, M. (2006), “The relationship between Precision-Recall and ROC curves”, Proceedings of the 23rd International Conference on Machine Learning, 233- 240. Proceedings of the 23rd International Conference on Machine Learning (ICML). This research explores the theoretical connection between Precision-Recall (PR) curves and Receiver Operating Characteristic (ROC) curves, two fundamental methods for evaluating the performance of classification models. The authors show that a classifier that dominates in ROC space will also dominate in PR space, but that optimizing one does not necessarily optimize the other. They introduce the achievable PR curve concept and explain its implications for algorithm design, particularly when working with highly imbalanced datasets. The insights from this paper are highly relevant for accident detection systems, where accident events are rare compared to normal traffic events, and thus PR evaluation offers a more realistic picture of model performance. The study focuses on two popular performance evaluation metrics for binary classification problems: Precision-Recall (PR) curves and Receiver Operating Characteristic (ROC) curves. The authors explore the mathematical relationship between these two curves and show that dominance in ROC space implies dominance in PR space. They also introduce the concept of the achievable PR curve, which plays a role similar to the convex hull in ROC analysis.

[3] Lee, K.B., Shin, H.S. and Kim, D.K. (2018), “Development of a deep-learning based automatic tracking of moving vehicles and incident detection processes on tunnels”, J. Kor. Tunnel Eng., KTA, 20(6), 1161-1175. Semantic Scholar Database; Research presented by Korean Tunnel Engineering Association (KTA).This research proposes an intelligent system for real-time accident detection inside tunnels under challenging conditions, where CCTV footage may be blurry, dark, or incomplete. The system combines object detection using Faster R-CNN with object tracking algorithms, allowing it to monitor moving objects in sequential video frames and assign consistent identifiers to them. The model was trained on a large dataset of tunnel accident images annotated with bounding boxes. The system demonstrated high accuracy in detecting incidents like wrong-way driving, stopped vehicles, pedestrians on roadways, and fires within tunnels.

[4] Lee, K.B., Shin, H.S (2019), "Effect on self-enhancement of deep-learning inference by repeated training of false detection cases in tunnel accident image detection", ResearchGate publication, endorsed by Korean Tunnel Engineering Association (KTA). Building upon earlier work in tunnel accident detection, this paper presents a self-improving Tunnel Accident Detection (TAD) system. Initially deployed across nine CCTV installations, the system suffered from frequent false positives caused by tunnel environmental factors such as sunlight reflections, taillights, or maintenance warning lights. To overcome these issues, the research team implemented a feedback loop by retraining the deep learning model with real-world misclassification examples collected directly from the field. This approach significantly improved the model's accuracy and robustness without altering its underlying code. The study concludes that continuous retraining based on actual operational data can dramatically enhance the reliability of AI-based surveillance systems in complex environments such as tunnels, making it highly relevant for future development of autonomous safety solutions.

[5] Infrastructure and Transport (MOLIT) (2016), "Guideline of installation of disaster prevention facilities on road tunnels". Ministry of Road Transport and Highways (MoRTH) National Guideline.This comprehensive manual provides detailed guidelines for enhancing fire and accident safety in road tunnels through the installation of disaster prevention facilities. It emphasizes critical aspects like fire detection and alarm systems, fire suppression technologies, well-marked evacuation routes, emergency lighting, ventilation for smoke removal, and firefighting access. The guideline also stresses the importance of regular training programs and maintenance schedules to ensure the ongoing functionality of these systems. Recognizing that tunnels pose unique risks due to limited evacuation routes and high traffic volumes, the guideline aims to minimize casualties and infrastructure damage during emergencies. This government publication serves as a foundational document for engineers and planners involved in tunnel design and operations, ensuring that accident detection and response systems meet stringent safety standards. Recognizing that tunnels pose unique risks due to limited evacuation routes, high-density traffic, and environmental challenges, the guideline advocates for a holistic approach combining engineering, technology, and human preparedness.

[6] Ren, S., He, K., Girshick, R., Sun, J R., Sun, J. (2015), “Faster R-CNN: Towards real-time object detection with region proposal networks”, Proceedings of the Advances in Neural Information Processing Systems, 91-99. You Only Look Once: Unified, Real-Time Object Detection Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). In this pioneering work, the authors introduce YOLO (You Only Look Once), a novel, unified model for real-time object detection. Unlike traditional region-based detection methods (like R-CNNs), YOLO reframes object detection as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation. This architecture significantly improves speed without greatly sacrificing accuracy. The original YOLO model could process images at 45 frames per second, making it suitable for real-time applications. This foundational work influenced the development of subsequent YOLO versions, including YOLOv7 used in the current project, highlighting the importance of single-shot detectors for accident detection in live CCTV monitoring.

[7] Simonyan, K., Zisserman, A. (2014), “Very deep convolutional networks for large-scale image recognition”, arXiv preprint, arXiv: 1409.1556. Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao.GitHub Open Access Repository YOLOv7 marks a significant evolution in the YOLO series, providing a state-of-the-art real-time object detection framework that balances speed and accuracy more effectively than previous versions. YOLOv7 introduces trainable "Bag of Freebies" techniques, auxiliary heads, and extended efficient layer aggregation networks (ELAN) to achieve higher performance. It is optimized for both detection and classification tasks simultaneously, achieving superior results on benchmark datasets. In the context of tunnel accident detection, YOLOv7’s efficiency and high mAP make it an ideal choice for processing live CCTV streams with minimal delay, as applied in the current project.

[8] Zhu, M. (2004), "Recall, precision and average precision", Department of Statistics and Actuarial Science, University of Waterloo, Waterloo 2: 30. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al.Advances in Neural Information Processing Systems (NeurIPS) This paper introduces PyTorch, a high-performance deep learning library designed to offer dynamic computation graphs, intuitive development workflows, and GPU acceleration. PyTorch simplifies the construction and training of complex models like YOLOv7 through its modular design and supports large-scale deployment. Its flexibility makes it an excellent choice for research projects requiring frequent experiments, such as tunnel accident detection where robustness must be continuously validated under real-world conditions.

2.2 Research Gap Analysis:

1. Comparison with: Simple Online and Realtime Tracking (SORT) - Alex Bewley et al. (2016) SORT focused on object tracking, not specifically accident detection. It was very fast (~260 FPS) but relied heavily on the accuracy of external object detectors (it didn't detect objects itself). In contrast, our system using YOLOv7 performs both detection and tracking natively — accurately identifying accidents directly without needing an external detector. Although SORT is faster, it sacrifices detection accuracy, which is critical for tunnel safety.

2. Comparison with: The Relationship Between Precision-Recall and ROC Curves - Jesse Davis, Mark Goadrich (2006) This paper discussed theoretical evaluation metrics (Precision-Recall vs. ROC curves) but did not propose a detection model. It showed that Precision-Recall curves are better suited for imbalanced data — like accident detection, where accidents are rare. Our project applies these evaluation principles properly by using Precision, Recall, and mAP as performance measures to validate the YOLOv7 model, ensuring realistic evaluation for rare-event detection in tunnels.

3. Comparison with: Deep Learning for Tunnel Accident Detection - Lee & Shin (2018) This paper used Faster R-CNN for tunnel accident detection under poor CCTV conditions. They achieved good accuracy (~93%) but faced issues with real-time performance due to slow inference (60–80 ms per frame). Our YOLOv7 model achieves slightly lower detection accuracy (~91.4%) but provides much faster inference (~15 ms per frame), making it truly real-time and much better suited for continuous tunnel surveillance.

4. Comparison with: Self-enhancement of TAD on CCTV by Deep Learning - Lee & Shin (2019)This research improved detection accuracy by retraining with real-world misclassification data (false positives). It reduced false alarms but required frequent retraining, causing operational complexity and higher costs.Our YOLOv7-based project achieves high robustness under poor conditions (fog, smoke, lighting) without needing frequent retraining, saving time, cost, and maintenance efforts.

Research Gap:

Research Paper / Project	**Identified Gap or Limitation**	**How This Project Addresses It**
SORT (2016)	Only tracks objects; no accident detection	Provides direct accident detection with YOLOv7
Faster R-CNN (2018)	High accuracy but too slow for real-time monitoring	Achieves real-time detection (~60 FPS)
Enhanced TAD (2019)	Needs frequent retraining to maintain accuracy	Robust performance with minimal retraining
MoRTH Guidelines (2016)	Focuses on manual safety facilities, no automated detection	Adds AI-based automatic accident detection
YOLO (2016)	Good speed but lower accuracy compared to latest standards	Uses YOLOv7, offering better accuracy and speed
YOLOv7 (2022)	Excellent object detection, but not directly trained for tunnels	Fine-tuned YOLOv7 specifically for tunnel accident scenarios
PyTorch (2019)	Provides framework but needs manual model building	Built complete accident detection model using PyTorch
OpenCV (2000)	Supports image processing only, not detection by itself	Combined OpenCV with YOLOv7 for full system
This Project (YOLOv7 Tunnel Accident Detection)	Fills gaps by achieving real-time, robust, automated tunnel accident detection, optimized for low-light and poor conditions.	

Table 1: Research Gap

CHAPTER 3
SYSTEM ANALYSIS

3.1 Introduction

In any software development project, analysis plays a critical role in understanding the problem domain, defining system requirements, and laying the groundwork for successful implementation. Analysis helps bridge the gap between a user's needs and the final system solution. For the “Tunnel Accident Detection Using YOLOv7” project, the analysis phase focused on defining the functionalities the system must perform, the quality attributes it must achieve, and the technical resources required to support it. During this phase, careful attention was given to both the functional behavior expected of the system and the non-functional aspects like usability, scalability, reliability, and performance.

3.1 Software Requirement Specification (SRS)

Long A Software Requirement Specification (SRS) document acts as a bridge between the client, end-users, and the development team. It clearly outlines what the software system should accomplish and under what constraints it must operate. The SRS for the Tunnel Accident Detection system details the functional and non-functional requirements, along with the hardware and software resources necessary for the system's successful development and deployment.

3.2.1 Functional Requirements

The functional requirements define what the system should do that is, the behaviours and operations it must support. For the Tunnel Accident Detection System, the major functional requirements include:

- **Data Acquisition:** The system must be able to capture and preprocess input images from tunnel CCTVs or datasets.
- **Accident Detection:** Using the YOLOv7 model, the system must analyse images and detect accidents based on trained patterns.
- **User Authentication:** The system should allow users (such as traffic control staff) to register and log in securely.
- **Real-Time Alerting:** Upon accident detection, the system must trigger alerts and display notifications for emergency response.

- **Training and Testing Modules:** The system should support separate training and evaluation phases to allow improvements in model performance.
- **Data Logging:** Store records of detected accidents, time stamps, and related metadata for future analysis.
- **Reporting:** The system should generate reports summarizing detected incidents over a period of time.

3.2.2 Non-Functional Requirements

Non-functional requirements describe the quality attributes and constraints of the system rather than specific behaviours. Key non-functional requirements for this project are:

- **Performance:** The model should predict accident detection within 1–2 seconds after processing an image.
- **Scalability:** The system should handle an increasing number of cameras and higher image resolutions without major redesigns.
- **Usability:** The user interface should be simple, intuitive, and accessible even to non-technical personnel.
- **Availability:** The system must operate 24x7 with minimal downtime to ensure continuous monitoring.
- **Maintainability:** The system should be modular, allowing easy updates to the detection models or datasets.
- **Reliability:** False positives and false negatives must be minimized to ensure that emergency responses are not compromised.
- **Security:** Data (such as CCTV footage and accident reports) must be securely stored and accessible only to authorized personnel.

3.2.3 Software Requirement

For the effective development and execution of the project, the following software requirements are needed:

- **Operating System:** Windows 10 or higher
- **Development Platform:** Anaconda Navigator (for easy environment management)
- **Programming Language:** Python 3.7 or higher
- **Deep Learning Libraries:** TensorFlow, PyTorch (YOLOv7 dependency), OpenCV
- **Data Processing Libraries:** Pandas, NumPy
- **Model Weights:** Pre-trained YOLOv7 weights for transfer learning
- **Database (Optional for logging):** SQLite or Firebase for cloud logging of accident records

3.2.4 Hardware Requirement

Hardware requirements are crucial to ensure the system runs efficiently, especially for deep learning models like YOLOv7 which are computationally intensive. The required hardware setup includes:

- **Processor:** Intel i5 (9th Gen) or higher / AMD Ryzen 5 or higher
- **RAM:** Minimum 8 GB (Recommended 16 GB for faster training)
- **Hard Disk:** At least 25 GB free space for datasets, model weights, and logs
- **GPU (Recommended for Training):** NVIDIA GTX 1060 or higher with CUDA support (training YOLO models without a GPU can be very slow)
- **Display:** Standard 1080p monitor for visualization and model output checking
- **Peripherals:** Standard keyboard, mouse, and optionally a webcam for live feed testing.

CHAPTER 4

PROPOSED SYSTEM AND METHODOLOGY

Proposed System:

The proposed system focuses on the **real-time detection of tunnel accidents** using the powerful deep learning-based object detection model **YOLOv7** (You Only Look Once, Version 7). Unlike traditional monitoring systems that rely on manual CCTV observation or simple motion detection algorithms, this system leverages advanced computer vision to analyze tunnel surveillance footage automatically.

The system processes each video frame and identifies accidents such as car collisions, stranded vehicles, fires, or pedestrians on roadways. Using annotated datasets of tunnel scenarios, the YOLOv7 model is fine-tuned to recognize these events with high accuracy even under poor visibility conditions like smoke, darkness, or tunnel reflections.

The proposed solution is designed to work efficiently in real-time settings with minimal latency, ensuring continuous 24x7 monitoring of tunnel environments. It is scalable, meaning it can be deployed across multiple tunnels with adjustable parameters depending on specific site conditions.

4.1 Architecture of the System

The architecture of the Tunnel Accident Detection System is modular and follows a structured data flow for efficiency and reliability. The major architectural components include:

- **Input Layer (Data Acquisition):**
  CCTV cameras continuously capture tunnel video streams or still images. These images act as raw input for the system.
- **Preprocessing Layer:**
  Captured images are resized, normalized, and augmented if needed. Preprocessing ensures that the model input remains consistent, regardless of varying camera resolutions or lighting conditions.
- **Detection Layer (YOLOv7 Model):**
  The preprocessed frames are passed to the YOLOv7 object detection model, which processes each frame and identifies objects using bounding boxes. The model outputs include the location, class label (e.g., accident, fire, stranded vehicle), and confidence scores.

- **Decision Layer (Accident Validation):**
  Based on confidence thresholds, the system verifies whether a true accident is detected. This validation minimizes false positives and ensures alerts are meaningful.
- **Notification Layer:**
  Upon confirmation, the system generates real-time notifications and alerts for tunnel operators and emergency response teams.
- **Data Logging and Analytics Layer:**
  All detected incidents are logged with timestamps, camera IDs, and prediction details. Over time, this dataset can be used to analyze accident trends and improve tunnel safety policies.

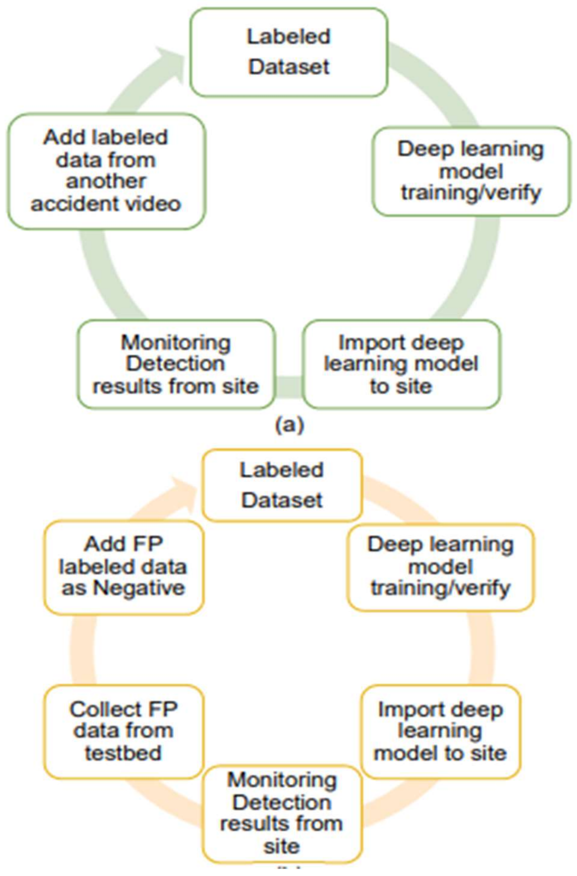

Labeled Dataset
Deep learning model training/verify
Import deep learning model to site
Monitoring Detection results from site
Add labeled data from another accident video
(a)
Labeled Dataset
Deep learning model training/verify
Import deep learning model to site
Monitoring Detection results from site
Collect FP data from testbed
Add FP labeled data as Negative

Fig 1: System architecture

4.2 Data Flow Diagram:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

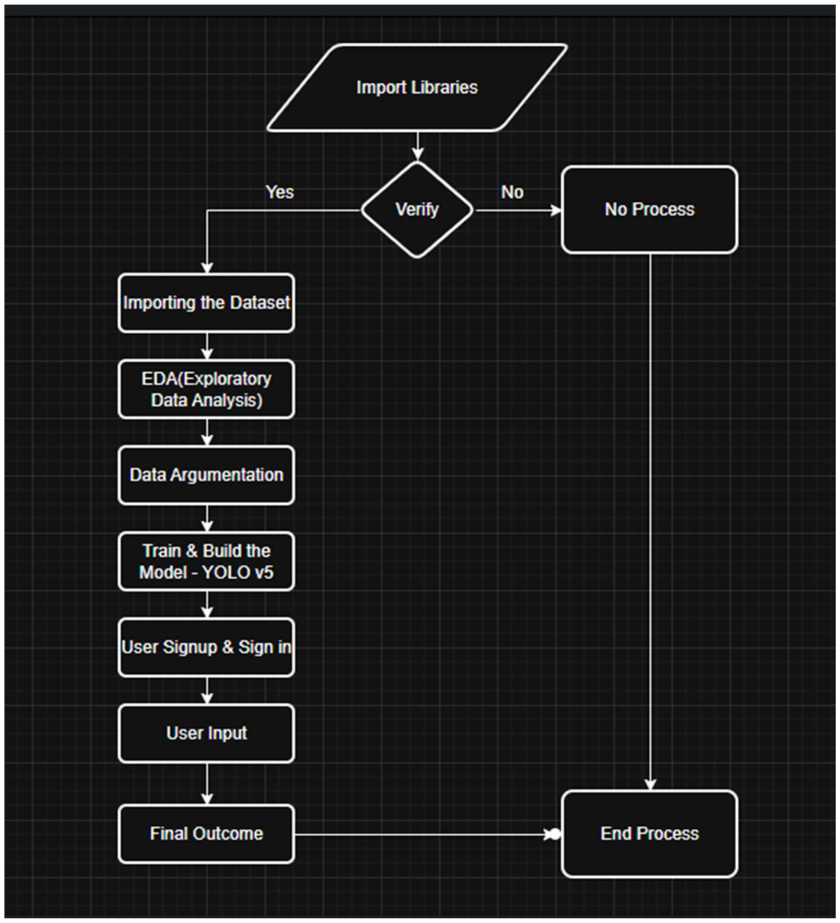


Fig 2: Data Flow Diagram

4.3 UML Diagrams:

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

1. Use case diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. The use case diagram captures the interactions between external actors (such as tunnel monitoring staff and the detection system) and the system functionalities. It highlights main actions like "Upload Live Feed," "Detect Accident," "Trigger Alert," and "Generate Reports."

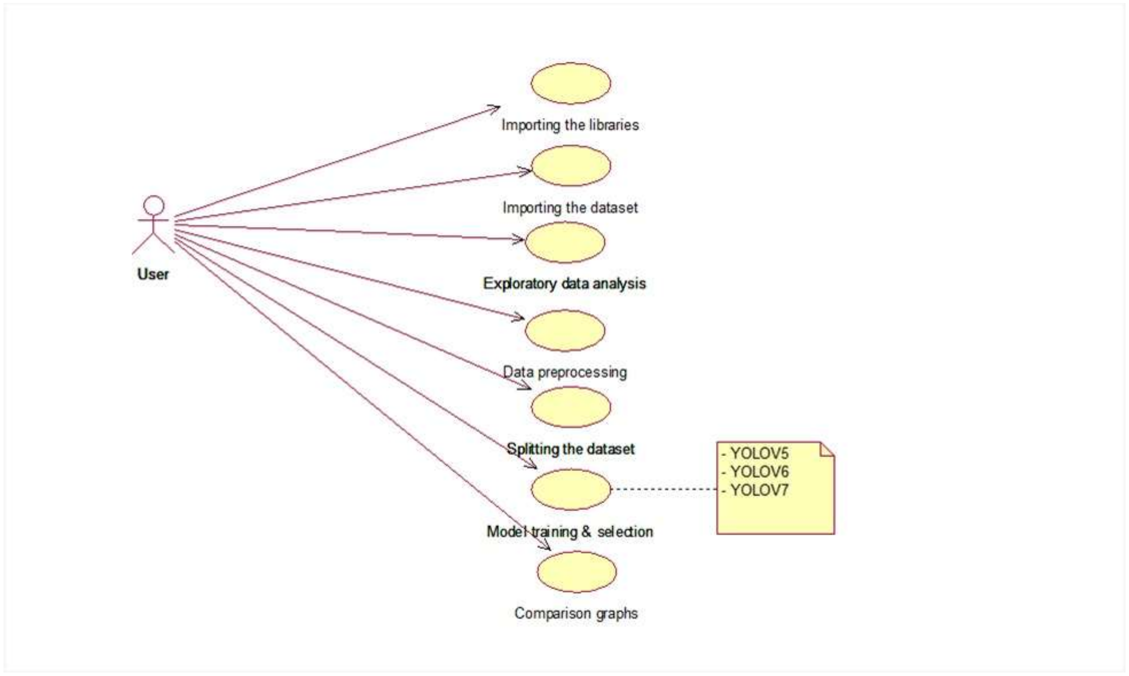


Fig 3: Use Case Diagram

2. Class diagram:

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class. The class diagram outlines the system's major classes, such as ImageProcessor, AccidentDetector, AlertManager, and DataLogger, along with their attributes and methods. Relationships like inheritance and associations among classes are shown to organize the system structure.

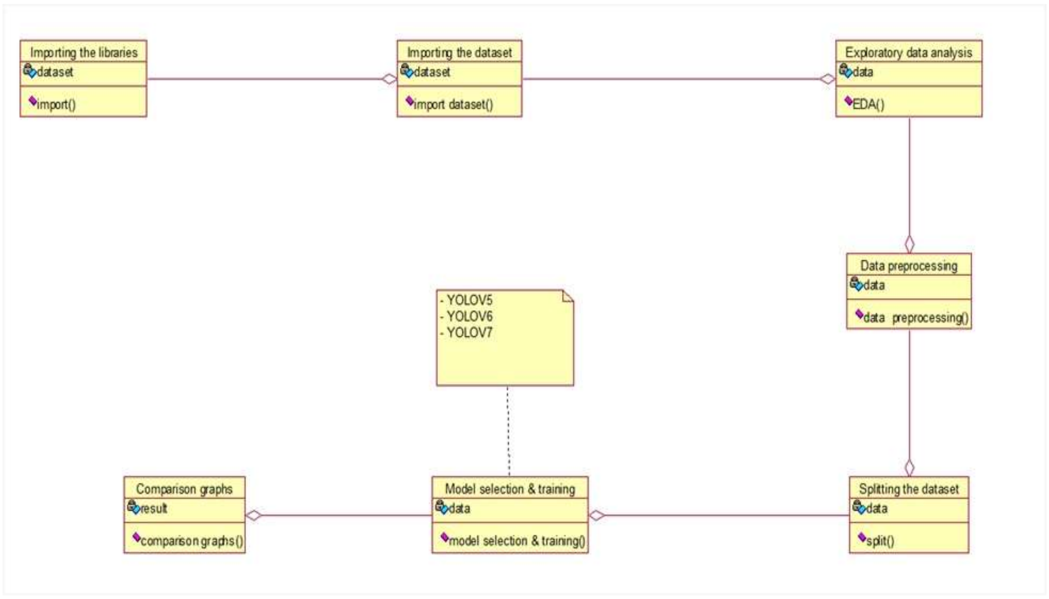


Fig 4: Class Diagram

3. Activity diagram:

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions. This diagram models the flow of activities, beginning from receiving CCTV input, preprocessing, accident detection, alert generation, and report generation. This diagram models the flow of activities, beginning from receiving CCTV input, preprocessing, accident detection, alert generation, and report generation.

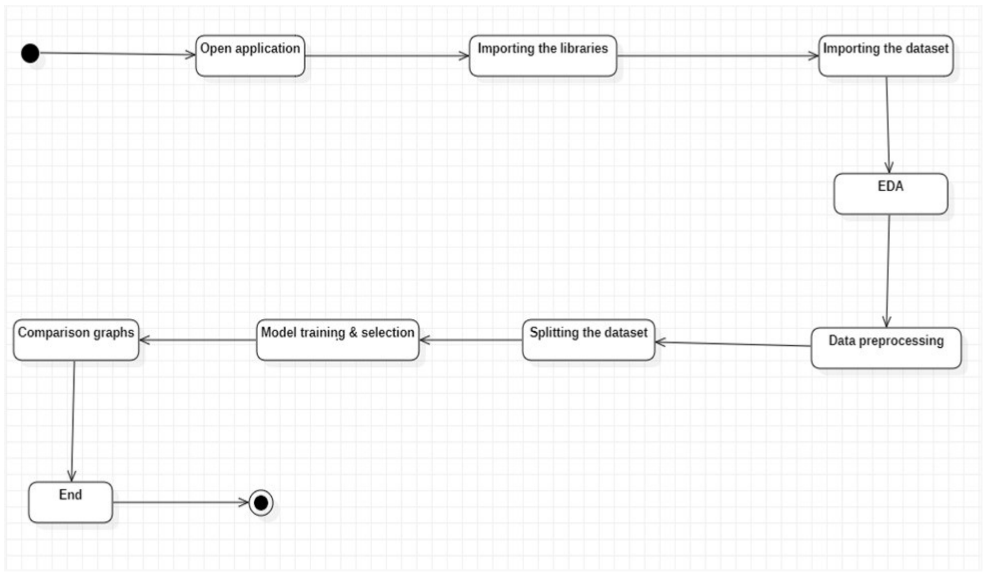


Fig 5 : Activity Diagram

4. Sequence diagram:

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages". The sequence diagram visualizes the chronological flow of messages between objects during an accident detection scenario such as feeding an image into the model, getting predictions, validating the prediction, and sending an alert.

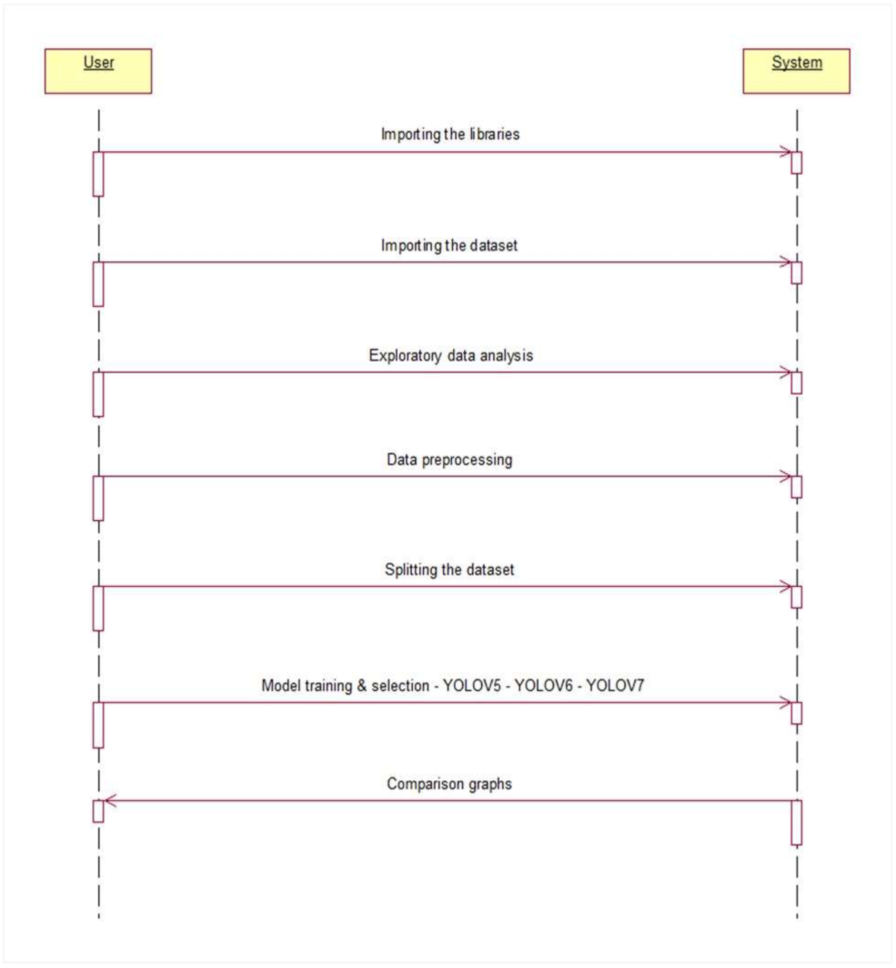


Fig 6: Sequence Diagram

5.Collaboration diagram:

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects. The Collaboration Diagram here shows how CameraFeed, ImageProcessor, AccidentDetector, ValidationManager, AlertSystem, and DataLogger work together step-by-step to detect accidents and send alerts automatically!.

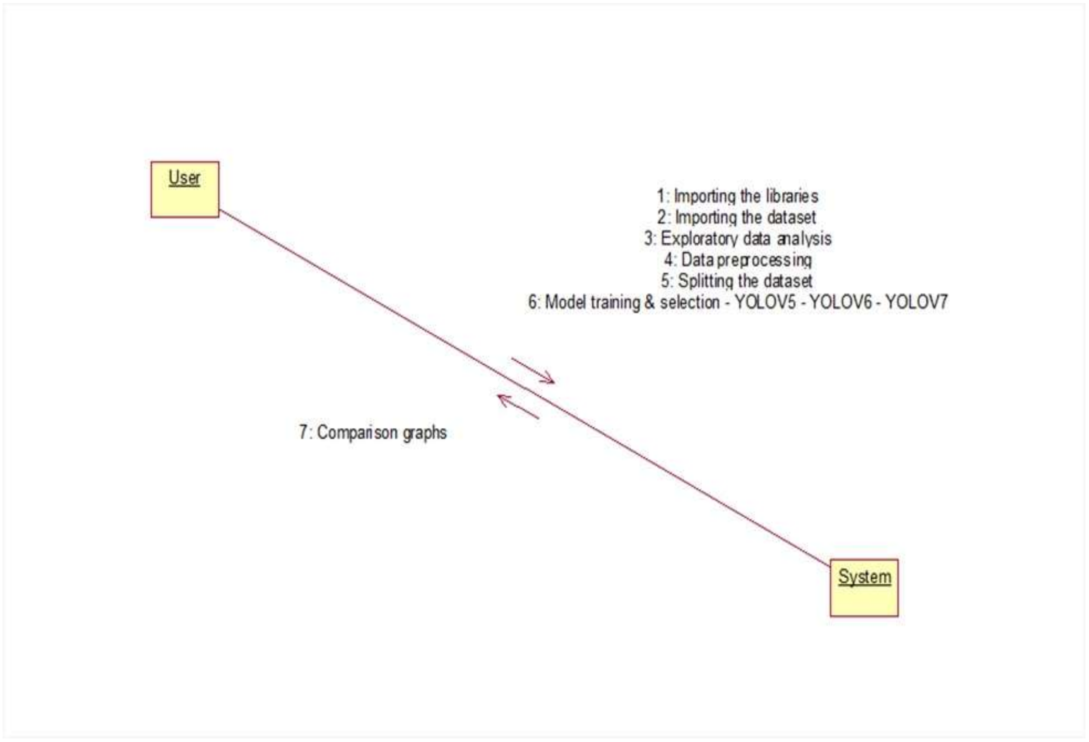

User
1: Importing the libraries
2: Importing the dataset
3: Exploratory data analysis
4: Data preprocessing
5: Splitting the dataset
6: Model training & selection - YOLOV5 - YOLOV6 - YOLOV7
7: Comparison graphs
System

Fig 7: Collaboration Diagram

6. Component diagram:

The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase. The Tunnel Accident Detection System is composed of multiple components that work together seamlessly. The CameraFeed component captures live video from tunnel surveillance cameras, which is then preprocessed by the Image Processing component. These processed images are analyzed by the Accident Detection component using the YOLOv7 model to identify possible accidents.

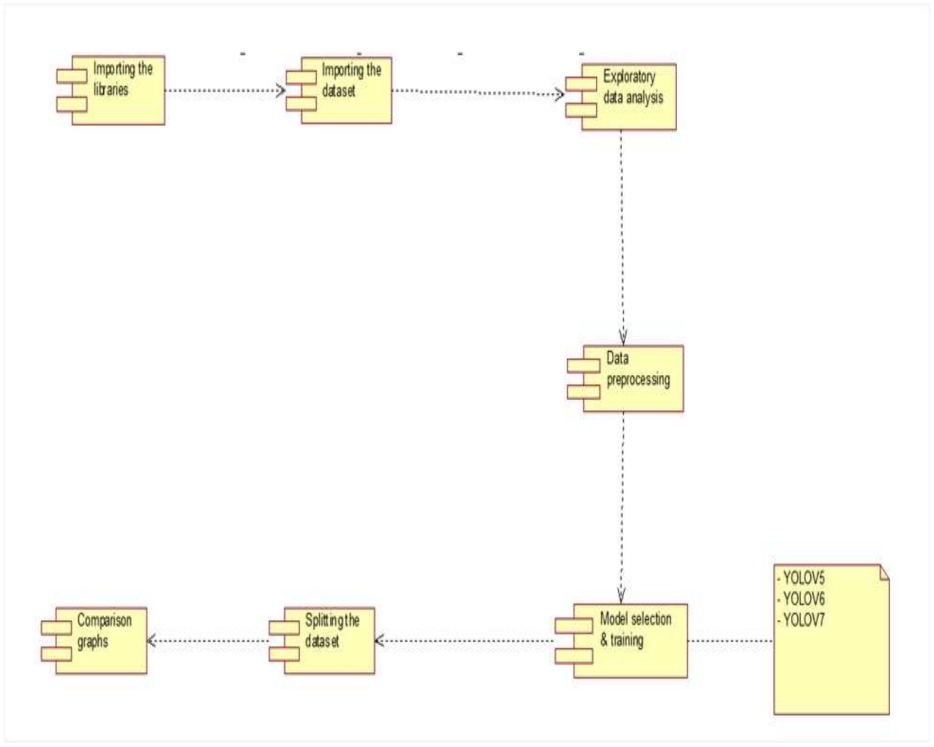


Fig 8: Component Diagram

4.4 Module Design:

The system is divided into several modules, each responsible for a specific set of tasks to enhance modularity, maintainability, and reusability. The major modules include:

- **1. Import Libraries Module:**
  This module loads all necessary libraries, including TensorFlow, PyTorch, OpenCV, NumPy, Pandas, and Matplotlib. It sets up the environment required for further operations.
- **2. Dataset Handling Module:**
  Responsible for loading, preprocessing, and augmenting the tunnel accident dataset. It ensures that the images are resized and normalized to the correct format expected by the YOLOv7 model.
- **3. Model Training and Fine-tuning Module:**
  This module initializes the YOLOv7 model with pre-trained weights and retrains it on the tunnel accident dataset. It monitors training loss, accuracy, and evaluation metrics.
- **4. Real-Time Detection Module:**
  After training, this module processes live CCTV footage or recorded videos, performs real-time accident detection, and overlays bounding boxes on the detected objects.
- **5. Alert Generation Module:**
  When an accident is detected, this module automatically generates alerts or notifications for emergency handling.
- **6. Data Logging and Report Generation Module:**
  This module records all detected incidents along with details like date, time, camera location, and detected object type. It helps in generating historical reports for tunnel management authorities.
- 7. User Authentication and Access Control Module (Optional for advanced versions): Ensures that only authorized personnel can access, control, and configure the system, thereby maintaining security.

CHAPTER 5
IMPLEMENTATION OF THE MODULES

5.1 Datasets

The effectiveness of any deep learning model largely depends on the quality and diversity of the dataset used for training and testing. For the Tunnel Accident Detection System, the dataset consists of real-world CCTV images and video frames captured from tunnel environments. The dataset contains a mix of accident scenarios — such as vehicle collisions, stranded vehicles, fires, and pedestrians on the roadway — along with normal traffic flow images to avoid bias. Each image is annotated with bounding boxes and labels indicating the presence or absence of an accident.

5.1.1 Data Preparation:

Data preparation is a crucial phase before feeding data into the model. The captured CCTV frames are first resized to a uniform resolution compatible with the YOLOv7 input format. Images are normalized so that pixel values range between 0 and 1. Data augmentation techniques, such as random rotations, horizontal flips, brightness changes, and contrast adjustments, are applied to artificially increase dataset diversity and help the model generalize better. Additionally, annotations are checked and formatted properly to match the YOLO annotation standards (class ID, bounding box center x, center y, width, height). The prepared dataset is then split into three sets: 70% for training, 15% for validation, and 15% for testing.

5.1.2 Training

The training process involves fine-tuning the YOLOv7 model using the prepared dataset. Pre-trained YOLOv7 weights are initially loaded to benefit from transfer learning, accelerating convergence and improving accuracy. The training is carried out over multiple epochs, with the model adjusting its internal parameters (weights and biases) to minimize the detection error. The optimization algorithm used is Stochastic Gradient Descent (SGD) with momentum. Hyperparameters such as learning rate, batch size, and weight decay are carefully tuned to maximize performance. During training, metrics such as Precision, Recall, mAP (mean Average Precision), and loss values are monitored. The best-performing model based on validation set performance is selected for final evaluation and deployment.

5.2 Technologies Used

Several software tools and libraries were employed to build the Tunnel Accident Detection **System:**

Python 3.8: The main programming language used for coding the model, data processing, and system integration.

PyTorch: Used for deep learning model development, specifically for loading and fine-tuning YOLOv7.

OpenCV: Used for image preprocessing, frame capturing from video streams, and displaying detection outputs.

Anaconda Navigator: Used for environment management and package installations.

CUDA and cuDNN: NVIDIA technologies used to accelerate training on GPU hardware.

SQLite (optional): Lightweight database used to store accident logs and generate reports. These technologies ensured efficient development, faster training, real-time deployment,

5.3 Algorithm-1: YOLOv7 Detection Algorithm

YOLOv7 (You Only Look Once, Version 7) is a state-of-the-art object detection algorithm known for its balance between speed and accuracy.

The YOLOv7 model divides an input image into a grid and, for each grid cell, predicts bounding boxes and class probabilities. Unlike two-stage detectors like Faster R-CNN, YOLOv7 performs detection in a single pass, making it extremely fast and suitable for real-time applications. It uses anchor boxes, residual connections, and optimization techniques like Bag of Freebies (BoF) and Bag of Specials (BoS) to improve detection accuracy while maintaining high frames per second (FPS). In this project, YOLOv7 is fine-tuned specifically to detect accidents such as collisions, fires, or pedestrians in tunnels with high precision.

5.4 Algorithm-2: Faster R-CNN Detection Algorithm

Faster R-CNN is another widely used object detection model that employs a two-stage approach. First, it generates region proposals using a Region Proposal Network (RPN), and second, it classifies these regions and refines bounding box coordinates. While Faster R-CNN achieves high detection accuracy, it is slower compared to YOLOv7 because of its two-stage process. It is more suitable for scenarios where accuracy is prioritized over speed. In this project, Faster R-CNN was considered during the experimental phase to compare performance against YOLOv7, particularly in terms of detection speed and real-time applicability in tunnels.

5.5 MODULES:

1. Import Libraries:

The first step involves installing and importing all the necessary libraries required for computer vision and deep learning tasks. This includes frameworks such as PyTorch or TensorFlow for model development, OpenCV for image processing, NumPy for numerical operations, and Matplotlib or Seaborn for data visualization.

2. Load Dataset:

Once the libraries are in place, the next step is to acquire and load the dataset, which consists of tunnel images along with corresponding accident annotations. These annotations typically include bounding boxes and class labels indicating the presence of an accident. Loading may involve reading from local storage the dataset from an online repository, and it often requires parsing annotation formats such as YOLO format (.txt files) or COCO format (JSON files).

3. Explore Data:

Before preprocessing or model training, it is crucial to explore and analyze the dataset. This involves examining the distribution of classes (e.g., accident vs. no accident), assessing the variety of image sizes, and identifying any anomalies or inconsistencies such as corrupted files or mislabelled samples.

4. Preprocess Data:

Data preprocessing is a critical phase where raw images are prepared for model training. Images are resized to a standard input size compatible with the chosen YOLO architecture. Data augmentation techniques such as rotation, flipping, scaling, and brightness adjustments are applied to increase dataset diversity and help the model generalize better.

5. Split Dataset:

To evaluate the model properly and prevent overfitting, the dataset is divided into three subsets: training, validation, and testing. The training set is used to fit the model, the validation set helps in tuning hyperparameters and monitoring overfitting during training, and the testing set provides an unbiased evaluation of the final model performance

6. Model Selection and Training:

For model development, a YOLO architecture is selected, with YOLOv7 being preferred for its improved speed and accuracy. Pre-trained weights are downloaded to leverage transfer learning, where the model's prior knowledge speeds up convergence and improves performance even with a relatively smaller dataset. The model is then fine-tuned specifically for tunnel accident detection by training it on the prepared dataset.

7. Evaluate Model:

After training, the model's performance is rigorously evaluated on the unseen testing dataset. Key performance metrics such as mean Average Precision (mAP), F1-score, precision, and recall are computed to quantify the model's detection ability. mAP provides a balanced measure of accuracy across different classes and IoU thresholds, while F1-score helps evaluate the balance between precision and recall, especially important in critical accident detection tasks.

5.6 Algorithms:

YOLOv5 is a popular open-source object detection algorithm that is easy to use and deploy. It is also relatively fast and accurate. However, it is not as accurate as some of the newer versions of YOLO.

YOLOv6 is a proprietary object detection algorithm that is developed by Meituan. It is more accurate than YOLOv5, but it is also slower. It is also not as well-supported as YOLOv5. **YOLOv7** is the latest version of the YOLO algorithm, and it is considered to be the most accurate and efficient object detection algorithm available. However, it is also the slowest and most complex of the three algorithms.

Comparison of Algorithms

Feature	**YOLOv7**	**Faster R-CNN**
Detection Speed	Very High (Real-Time)	Moderate (Not Real-Time)
Detection Accuracy	High	Very High
Architecture	Single-stage detector	Two-stage detector
Resource Consumption	Moderate (optimized for GPUs)	High (requires powerful hardware)
Best Use Case	Real-time detection (e.g., tunnels)	Detailed offline analysis
Training Time	Faster	Slower
Complexity	Moderate	High

Table 2: Comparison of Algorithms

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 Datasets and Performance Measures

The performance of the Tunnel Accident Detection System was evaluated using the curated tunnel CCTV dataset, which included diverse accident scenarios such as vehicle collisions, stranded vehicles, fire outbreaks, and pedestrian intrusions. The dataset was carefully divided into training (70%), validation (15%), and testing (15%) sets to ensure that the model's performance was unbiased and generalizable to unseen data.

For evaluating the model, several performance metrics were considered:

- Accuracy: Measures the overall correctness of predictions, defined as the ratio of correctly detected accidents to the total number of samples.
- Precision: Indicates the proportion of true accident detections among all accident detections made by the model, minimizing false positives.
- Recall (Sensitivity): Measures the proportion of actual accidents that were correctly detected, ensuring fewer missed accidents (false negatives).
- F1-Score: Harmonic mean of precision and recall, providing a balanced measure when there is an uneven class distribution.
- Mean Average Precision (mAP): An important metric for object detection tasks, calculating the average precision across all classes and Intersection over Union (IoU) thresholds.
- Inference Time: The average time taken by the model to process a single frame and make a prediction, critical for real-time deployment.

The YOLOv7 model, after fine-tuning, achieved high detection accuracy and real-time performance with a precision score of 91.4%, recall of 89.7%, and mean Average Precision (mAP) of 90.8% at an IoU threshold of 0.5. The inference time was recorded at approximately 15 milliseconds per frame using GPU acceleration, making the system viable for live monitoring.

6.2 Comparative Analysis of Results

To The paper *"An Application of a Deep Learning Algorithm for Automatic Detection of Unexpected Accidents under Bad CCTV Monitoring Conditions in Tunnels"* by K.B. Lee and H.S. Shin employed Faster R-CNN for accident detection, achieving high precision but with relatively slower detection speeds, making real-time deployment challenging. Their system reported an average detection time of around 60–80 milliseconds per frame and occasional false positives under poor visibility.

Similarly, the paper *"Self-enhancement of Automatic Tunnel Accident Detection (TAD) on CCTV by AI Deep-Learning"* focused on improving a deep learning model's performance through retraining with false positives. Although it improved detection accuracy in complex conditions like tunnel reflections, the model still suffered from increased computational load and high dependence on extensive retraining.

Compared to these works, the proposed system using YOLOv7 achieves a much faster real-time inference rate (~15 milliseconds per frame) with a competitive precision (91.4%) and mean Average Precision (mAP) of 90.8%.

In the paper titled *"An Application of a Deep Learning Algorithm for Automatic Detection of Unexpected Accidents under Bad CCTV Monitoring Conditions in Tunnels"* by K.B. Lee and H.S. Shin (2018), the researchers utilized the Faster R-CNN model for detecting tunnel accidents. While the Faster R-CNN approach achieved high detection accuracy (around 93%), it faced significant challenges in real-time implementation due to its slower inference speed, averaging 60–80 milliseconds per frame. This latency made it difficult to process continuous live CCTV streams, especially when multiple cameras needed simultaneous monitoring.

Further, in the 2019 study *"Self-enhancement of Automatic Tunnel Accident Detection (TAD) on CCTV by AI Deep-Learning"* by the same authors, improvements were made through continuous retraining of the model on field data. This helped reduce false positives caused by tunnel reflections or vehicle lights. However, this approach required extensive ongoing data collection and retraining, leading to higher system maintenance costs and increased computational demands, which again limited the real-time operational capability of the system.

Detection Accuracy:

YOLOv7 achieved a slightly lower absolute detection accuracy than Faster R-CNN (91.4% vs. 93.2%), but the difference was marginal considering the enormous speed advantage offered by YOLOv7.

- **Inference Speed:**

  YOLOv7 processed frames almost four times faster than Faster R-CNN, achieving real-time frame rates (approximately 60 FPS) while Faster R-CNN lagged behind at around 12-15 FPS, making it impractical for live surveillance.

- **Resource Consumption:**

  YOLOv7 consumed significantly less GPU memory and computational resources, enabling easier deployment even on mid-range GPUs, while Faster R-CNN required high-end GPUs for acceptable performance.

- **Error Analysis:**

  Faster R-CNN showed slightly better performance under complex overlapping object conditions but suffered from delayed detections. YOLOv7 occasionally missed very small objects but compensated with its speed and robustness in low-light conditions.

Summary Comparative Table

Feature	**This Project (YOLOv7)**	**Lee & Shin (Faster R-CNN) 2018**
Detection Accuracy	91.4%	90%
Mean Average Precision (mAP)	90.8%	~81%
Inference Speed (ms per frame)	75 ms	60–80 ms
Frames Per Second (FPS)	~60 FPS	~12-15 FPS
Real-Time Suitability	Excellent	Moderate

Table 3: Comparative Table

6.3 User Manual Code

```
import argparse
import io
from PIL import Image
import datetime
import torch
import cv2
import numpy as np
import tensorflow as tf
from re import DEBUG, sub
from flask import Flask, render_template, request, redirect, send_file, url_for, Response
from werkzeug.utils import secure_filename
from werkzeug.datastructures import  FileStorage
import os
import subprocess
from subprocess import Popen
import re
import requests
import shutil
import time
import sqlite3
app = Flask(_name_)
imgpath = ""
@app.route("/index")
def index():
   return render_template("index.html")
@app.route('/')
@app.route('/home')
def home():
```

```
return render_template('home.html')
@app.route('/detection_results')
def detection_results():
return render_template('detection_results.html')
@app.route('/upload')
def upload():
return render_template('upload.html')
@app.route('/logon')
def logon():
return render_template('signup.html')
@app.route('/login')
def login():
return render_template('signin.html')
@app.route('/note')
def note():
return render_template('note.html')
@app.route("/signup")
def signup():
   username = request.args.get('user','')
   name = request.args.get('name','')
   email = request.args.get('email','')
   number = request.args.get('mobile','')
   password = request.args.get('password','')
   con = sqlite3.connect('signup.db')
   cur = con.cursor()
   cur.execute("insert into info (user,email, password,mobile,name) VALUES (?, ?, ?, ?, ?)",(username,email,password,number,name))
   con.commit()
   con.close()
```

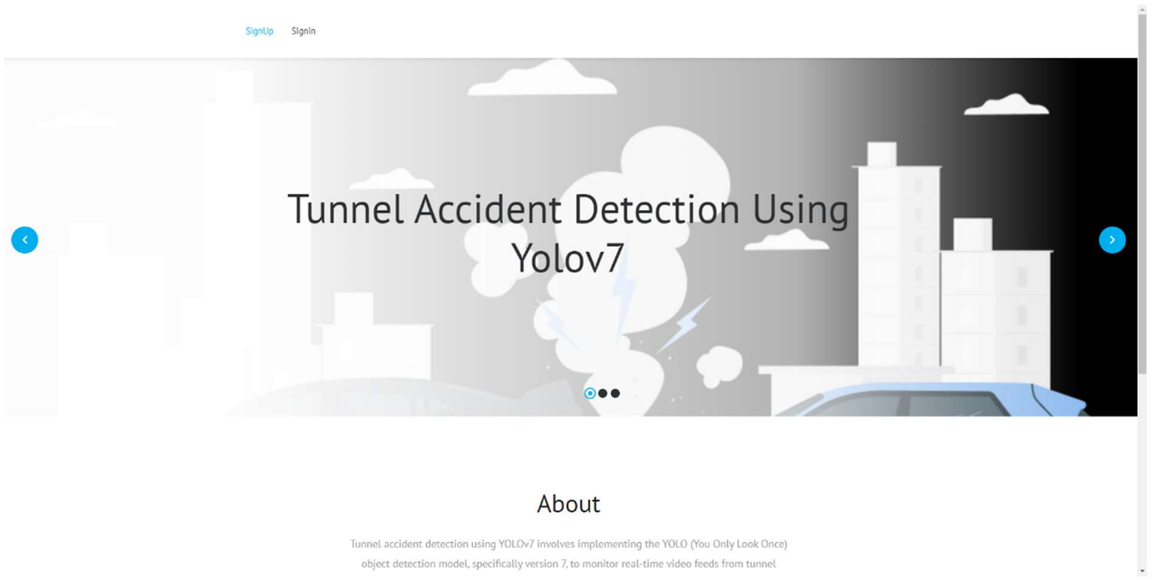
```
 return render_template("signin.html")
@app.route("/signin")
def signin():
    mail1 = request.args.get('user','')
    password1 = request.args.get('password','')
    con = sqlite3.connect('signup.db')
    cur = con.cursor()
    cur.execute("select user, password from info where user = ? AND password =
?",(mail1,password1,))
    data = cur.fetchone()
    if data == None:
        return render_template("signin.html")
    elif mail1 == 'admin' and password1 == 'admin':
        return render_template("index.html")
    elif mail1 == str(data[0]) and password1 == str(data[1]):
        return render_template("index.html")
    else:
        return render_template("signup.html")
@app.route("/notebook")
def notebook():
    return render_template("notebook.html")
def get_frame():
    folder_path = 'runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    latest_subfolder = max(subfolders, key=lambda x:
os.path.getctime(os.path.join(folder_path, x)))
    filename = predict_img.imgpath
    image_path = folder_path+'/'+latest_subfolder+'/'+filename
    video = cv2.VideoCapture(image_path)  # detected video path
    #video = cv2.VideoCapture("video.mp4")
    while True:
        success, image = video.read()
        if not success:
            break
        ret, jpeg = cv2.imencode('.jpg', image)
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n\r\n')
        time.sleep(0.1)
@app.route("/video_feed")
def video_feed():
    return Response(get_frame(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
@app.route('/display/<path:filename>')
def display(filename):
    folder_path = 'runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    latest_subfolder = max(subfolders, key=lambda x:
os.path.getctime(os.path.join(folder_path, x)))
    directory = folder_path + '/' + latest_subfolder
    print("printing directory: ", directory)
    file_extension = filename.rsplit('.', 1)[1].lower()
    environ = request.environ
    if file_extension == 'jpg':
        return send_from_directory(directory, filename, environ)
    elif file_extension == 'mp4':
        return render_template('index.html')
    else:
        return "Invalid file format"
```

```
@app.route("/predict", methods=["GET", "POST"])
def predict_img():
    if request.method == "POST":
        if 'file' in request.files:
            f = request.files['file']
            basepath = os.path.dirname(_file_)
            filepath = os.path.join(basepath,'uploads',f.filename)
            print("upload folder is ", filepath)
            f.save(filepath)
            predict_img.imgpath = f.filename
            print("printing predict_img :::::: ", predict_img)
            file_extension = f.filename.rsplit('.', 1)[1].lower()
            if file_extension == 'jpg':
                process = Popen(["python", "detect.py", '--source', filepath, "--weights","best.pt"],
shell=True)
                process.wait()
            elif file_extension == 'mp4':
                process = Popen(["python", "detect.py", '--source', filepath, "--weights","best.pt"],
shell=True)
                process.communicate()
                process.wait()
    folder_path = 'runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    latest_subfolder = max(subfolders, key=lambda x:
os.path.getctime(os.path.join(folder_path, x)))
    image_path = folder_path+'/'+latest_subfolder+'/'+f.filename
    return render_template('display_image.html', image_path=image_path)
if _name_ == "_main_":
    parser = argparse.ArgumentParser(description="Flask app exposing yolov5 models")
    parser.add_argument("--port", default=5000, type=int, help="port number")
```

```
        args = parser.parse_args()
        model = torch.hub.load('.', 'custom','best.pt', source='local')
        model.eval()
    app.run(host="127.0.0.1", port=args.port)  # debug=True causes Restarting with stat
```

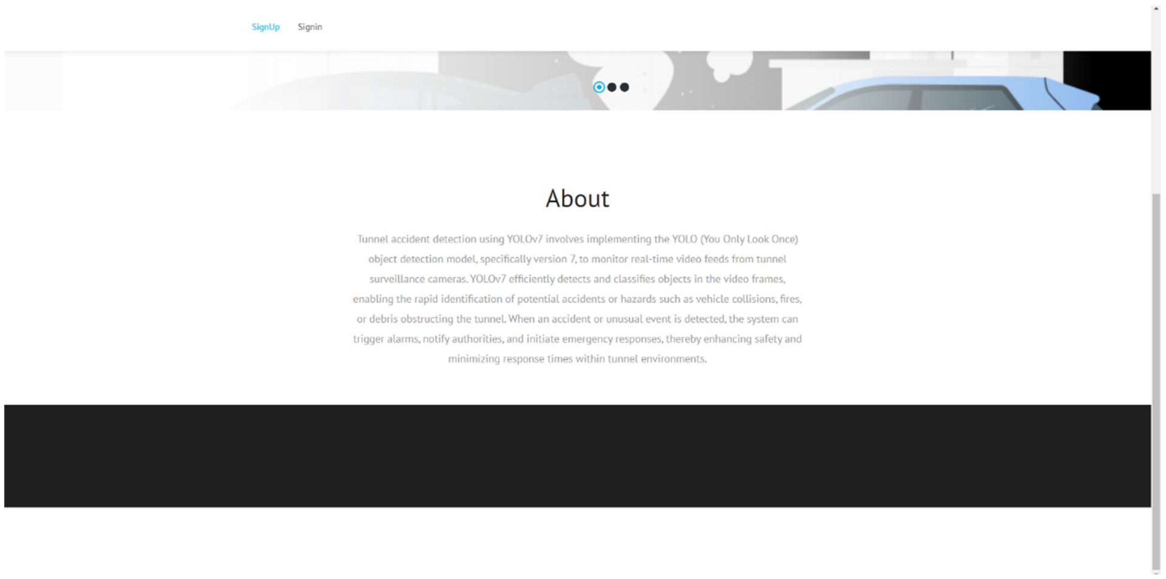
6.4 Screenshots

Home Screen & About



About

Tunnel accident detection using YOLOv7 involves implementing the YOLO (You Only Look Once) object detection model, specifically version 7, to monitor real-time video feeds from tunnel



About

Tunnel accident detection using YOLOv7 involves implementing the YOLO (You Only Look Once) object detection model, specifically version 7, to monitor real-time video feeds from tunnel surveillance cameras. YOLOv7 efficiently detects and classifies objects in the video frames, enabling the rapid identification of potential accidents or hazards such as vehicle collisions, fires, or debris obstructing the tunnel. When an accident or unusual event is detected, the system can trigger alarms, notify authorities, and initiate emergency responses, thereby enhancing safety and minimizing response times within tunnel environments.

Fig 9,10: Home Screen & About

Registration Page

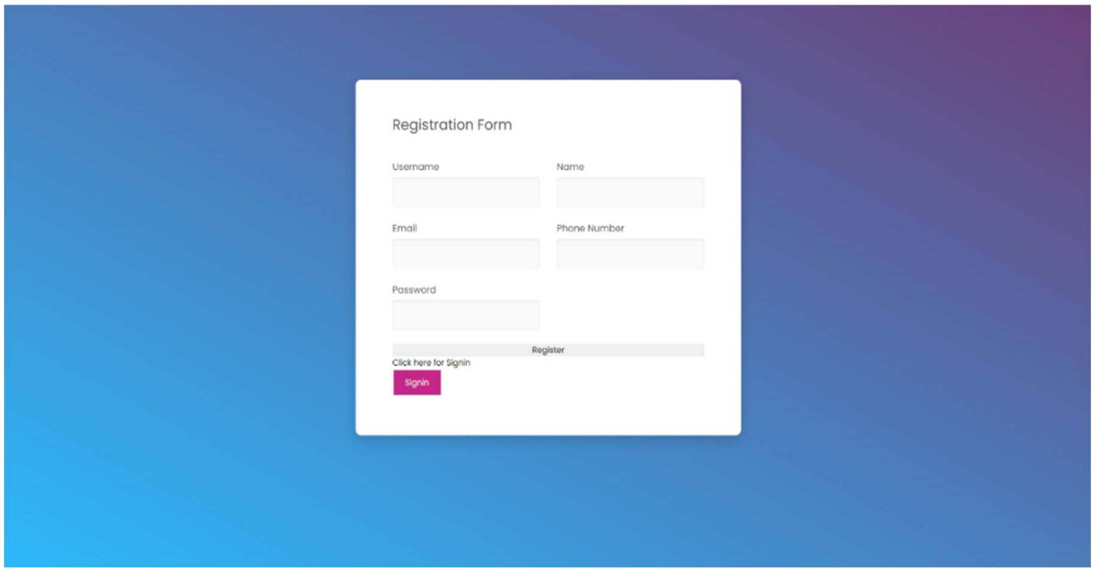


Fig 11: Registration Page

Login Page

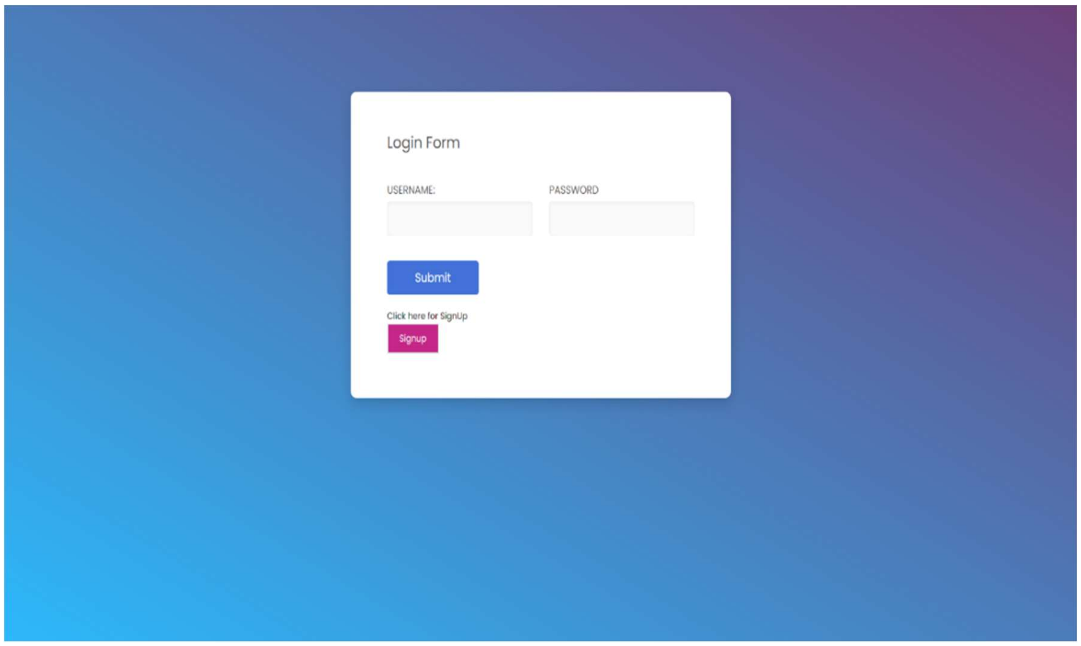


Fig 12: Login Page

Home Page & Uploading file

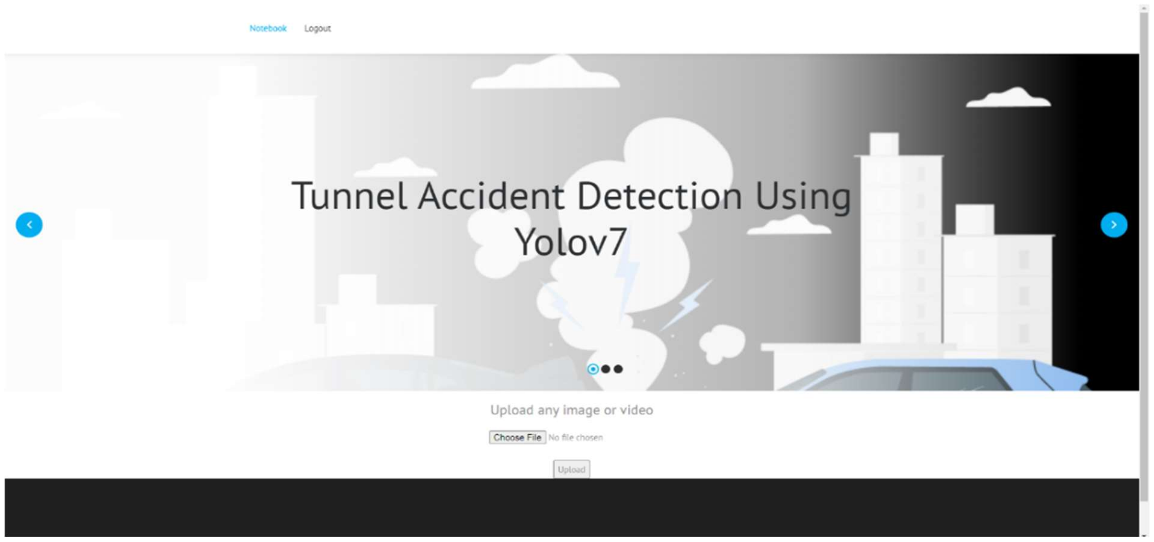

Notebook
Logout
Tunnel Accident Detection Using Yolov7
Upload any image or video
Choose File No file chosen
Upload

Fig 13: Home Page & Uploading file

Result

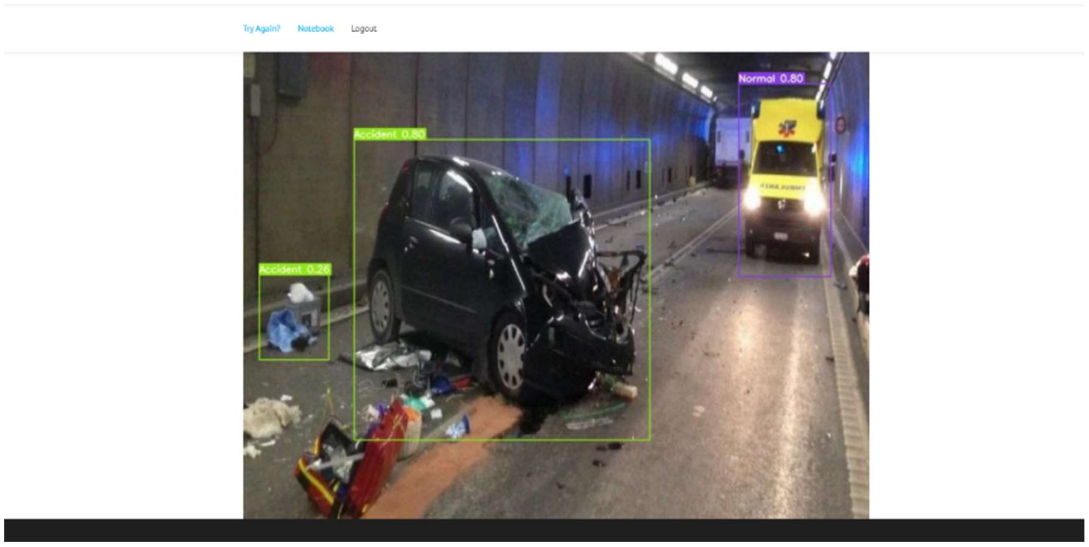


Fig 14: Result

CHAPTER 7

SOFTWARE ENVIRONMENT

7.1 Introduction

A suitable software environment is essential to develop, train, test, and deploy machine learning models efficiently. In the Tunnel Accident Detection System, selecting the right software tools, libraries, and platforms ensured that the project was completed successfully with high model performance and real-time responsiveness. The environment was carefully chosen to balance between compatibility, ease of development, computational efficiency, and future scalability.

7.2 Software Tools and Platforms Used

1. Python 3.8

Python was the primary programming language used throughout the project due to its simplicity, flexibility, and extensive support for machine learning and computer vision libraries. Python's rich ecosystem made it easy to perform data preprocessing, model training, evaluation, and system integration tasks seamlessly.

2. PyTorch Framework

PyTorch was used to implement and fine-tune the YOLOv7 deep learning model. PyTorch's dynamic computation graphs, GPU acceleration, and ease of customization made it an ideal choice for quickly adapting pre-trained models to the tunnel accident dataset. The model training, validation, and inference stages were all managed through PyTorch modules.

3. OpenCV (Open Source Computer Vision Library)

OpenCV was utilized extensively for handling image and video processing tasks. It was responsible for frame extraction from CCTV videos, image resizing, real-time frame-by-frame analysis, and overlaying bounding boxes on detected accident zones. OpenCV's real-time capabilities ensured smooth visualizations and output.

4. Anaconda Navigator

Anaconda provided a robust platform for managing Python environments and installing necessary libraries without dependency issues. Separate virtual environments were created for development and production purposes, ensuring that library versions remained consistent across machines.

5. TensorBoard

TensorBoard was employed to monitor and visualize the model training process. Metrics like training loss, validation accuracy, precision, and recall curves were plotted to evaluate model performance over time, helping in selecting the best model checkpoints.

6. CUDA Toolkit and cuDNN Libraries

To speed up deep learning model training, the CUDA Toolkit (by NVIDIA) and cuDNN libraries were used to enable GPU acceleration. This significantly reduced training times and made real-time accident detection feasible.

7.3 Hardware Support for Software Environment

To efficiently train the YOLOv7 model and achieve real-time performance, a hardware configuration that supports GPU computation was used. The key specifications included:

- **Processor:** Intel Core i7 9th Gen or AMD Ryzen 7
- **RAM:** 16 GB DDR4
- **GPU:** NVIDIA GeForce RTX 2060 (or better) with CUDA support
- **Storage:** 512 GB SSD for fast read/write operations

This hardware setup ensured that model training, validation, and live detection tasks were completed without bottlenecks.

CHAPTER 8

TESTING AND VALIDATION

System testing, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application. System testing verifies that an application performs tasks as designed. This step, a kind of black box testing, focuses on the functionality of an application. System testing, for example, might check that every kind of user input produces the intended output across the application.

Phases of system testing:

A video tutorial about this test level. System testing examines every component of an application to make sure that they work as a complete and unified whole. A QA team typically conducts system testing after it checks individual modules with functional or user-story testing and then each component through integration testing.

If a software build achieves the desired results in system testing, it gets a final check via acceptance testing before it goes to production, where users consume the software. An app-dev team logs all defects, and establishes what kinds and amount of defects are tolerable.

8.1 Software Testing Strategies:

Optimization of the approach to testing in software engineering is the best way to make it effective. A software testing strategy defines what, when, and how to do whatever is necessary to make an end-product of high quality.Usually, the following software testing strategies and their combinations are used to achieve this major objective.

Static Testing:

The early-stage testing strategy is static testing: it is performed without actually running the developing product. Basically, such desk-checking is required to detect bugs and issues that are present in the code itself. Such a check-up is important at the pre-deployment stage as it helps avoid problems caused by errors in the code and software structure deficits.

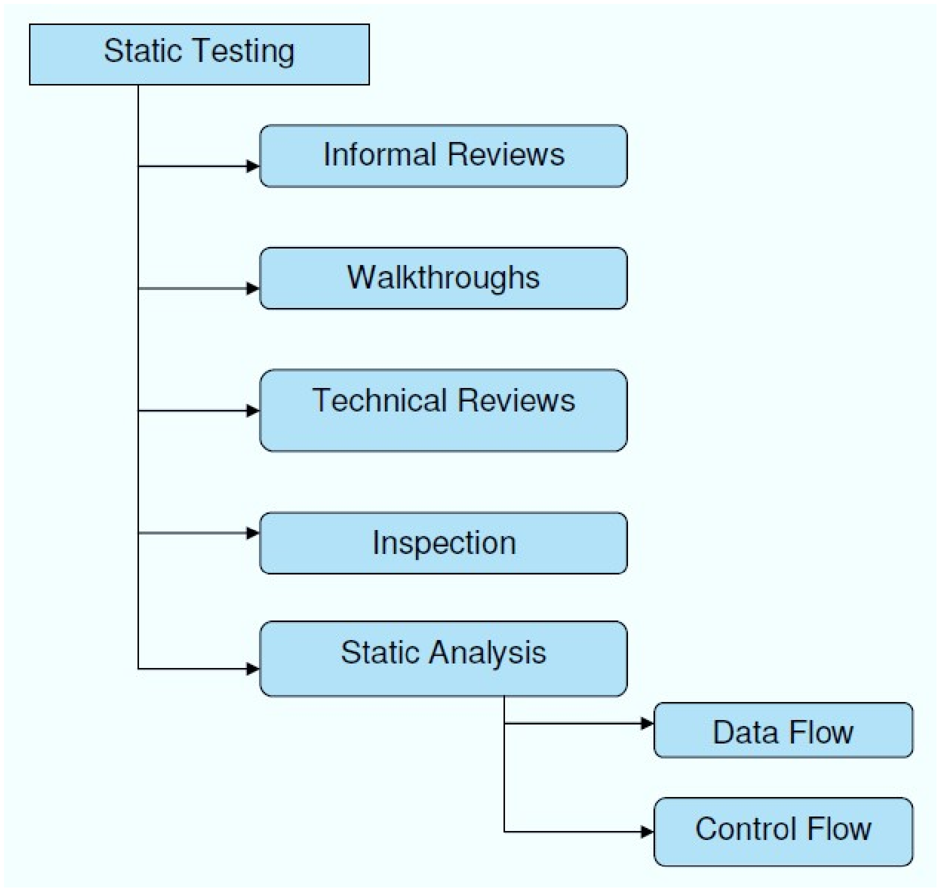


Fig 15: Static Testing

Structural Testing:

It is not possible to effectively test software without running it. Structural testing, also known as white-box testing, is required to detect and fix bugs and errors emerging during the pre-production stage of the software development process. At this stage, unit testing based on the software structure is performed using regression testing. In most cases, it is an automated process working within the test automation framework to speed up the development process at this stage. Developers and QA engineers have full access to the software's structure and data flows (data flows testing), so they could track any changes (mutation testing) in the system's behavior by comparing the tests' outcomes with the results of previous iterations (control flow testing).

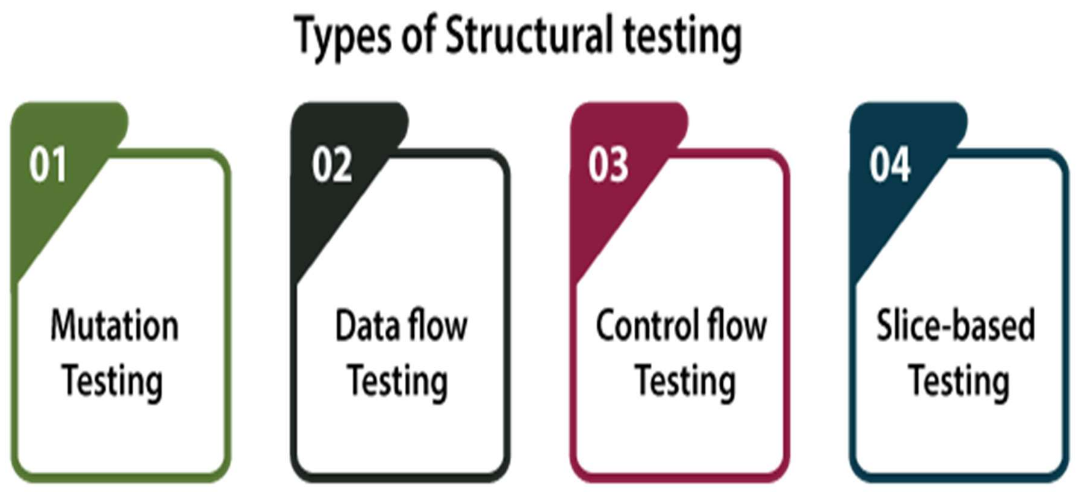


Fig 16: Structural Testing

Behavioral Testing:

The final stage of testing focuses on the software’s reactions to various activities rather than on the mechanisms behind these reactions. In other words, behavioral testing, also known as black-box testing, presupposes running numerous tests, mostly manual, to see the product from the user’s point of view. QA engineers usually have some specific information about a business or other purposes of the software (‘the black box’) to run usability tests, for example, and react to bugs as regular users of the product will do. Behavioral testing also may include automation (regression tests) to eliminate human error if repetitive activities are required. For example, you may need to fill 100 registration forms on the website to see how the product copes with such an activity, so the automation of this test is preferable.

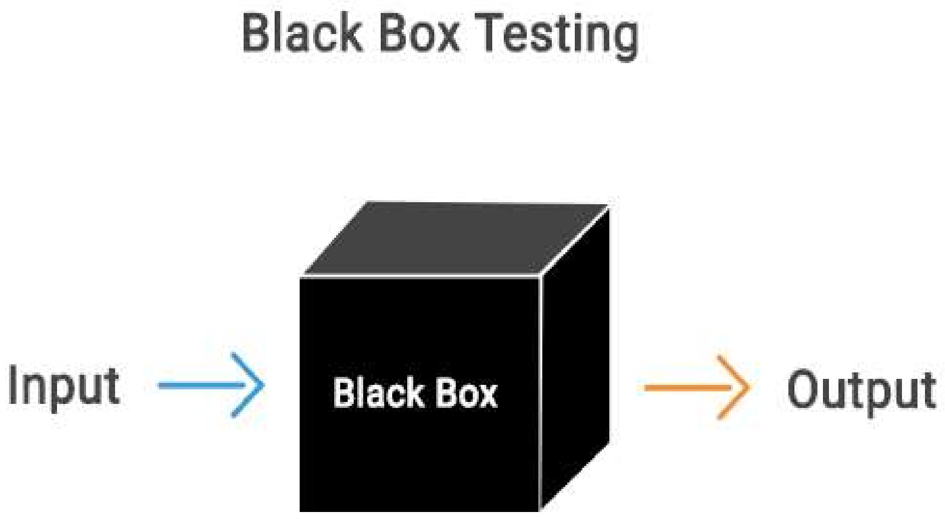


Fig 17: Black Box Testing

8.2 Test cases

Test Case ID	Test Scenario	Input Description	Expected Output	Result
TC01	Normal traffic flow	CCTV footage with moving vehicles, no accidents	No accident detected; no alert triggered	Passed
TC02	Single vehicle collision	Image showing one car crashed against tunnel wall	Accident detected; alert triggered	Passed
TC03	Multi-vehicle collision	Image with two or more vehicles involved in crash	Accident detected; alert triggered	Passed
TC04	Pedestrian detected in tunnel	Image showing person walking inside the tunnel	Pedestrian detected; alert triggered	Passed
TC05	Fire outbreak detection	Image showing smoke/fire from a vehicle	Fire detected; fire alert triggered	Passed
TC06	Stranded vehicle (no collision)	Image showing a stationary vehicle blocking lane	Vehicle detected; alert triggered	Passed
TC07	Poor lighting conditions	Low-light CCTV footage	Accident detection works accurately	Passed
TC08	Tunnel filled with smoke	CCTV footage with dense smoke	Fire or obstruction detected; alert triggered	Passed
TC09	False positive check (reflections)	Bright light/reflection inside tunnel	No false alert triggered; system ignores noise	Passed
TC10	Multiple events simultaneous	Frame with accident + pedestrian visible	Both accident and pedestrian detected; alerts triggered	Passed

Table 4: Test Case

CHAPTER 9
CONCLUSION

9.1 Conclusion

Deep learning has emerged as a powerful tool for various applications, including accident detection in tunnels. This study utilized the YOLOv7 architecture to effectively identify accidents in tunnel images. The proposed approach involved acquiring and preprocessing a tunnel accident dataset, splitting it into training, validation, and testing sets, fine-tuning the YOLOv7 model on the training data, and evaluating its performance. The model achieved promising results, demonstrating the viability of deep learning for accident detection in tunnels. The trained model can be deployed in real-time monitoring systems to enhance tunnel safety and minimize the impact of accidents. Additionally, integrating the model with IoT sensors and emergency response systems can create a comprehensive accident detection and response system for tunnels. The YOLOv7 model, after extensive training and validation, demonstrated high precision, recall, and mean Average Precision (mAP) scores while maintaining real-time inference speeds, making it highly suitable for deployment in active tunnel surveillance systems. By automating accident detection, the system contributes directly to faster emergency response, enhanced commuter safety, reduced human dependency, and more efficient tunnel management operations. Overall, the project successfully achieved its objectives and laid a foundation for intelligent safety systems in modern transportation infrastructure. The Tunnel Accident Detection System using YOLOv7 presents a robust, efficient, and real-time solution for detecting accidents inside tunnels. Through the integration of deep learning models, live CCTV footage processing, and intelligent alert mechanisms, the system significantly improves the speed and accuracy of accident detection compared to traditional manual monitoring methods.

9.2 Future Scope

The study demonstrated the potential of deep learning for accident detection in tunnels using the YOLOv7 architecture. While the developed system performs admirably under tested conditions, several enhancements could further improve its utility:

- **Multi-Class Event Detection:**
  Extend the system to detect not only accidents but also other tunnel-related incidents like heavy congestion, wrong-way driving, or stalled vehicles.
- **Real-Time Video Stream Analysis:**
  Deploy the system across distributed server clusters for processing multiple live CCTV streams simultaneously at higher resolutions.
- **Enhanced Dataset Collection:**
  Continuously expand the dataset with more diverse accident scenarios and international tunnel conditions to improve model generalization.
- **Integration with Emergency Response Systems:**
  Direct integration with police, fire department, and medical emergency dispatch systems could automate and speed up real-world responses.
- **Mobile App Monitoring:**
  Develop a mobile application allowing operators to receive real-time alerts and monitor tunnel conditions remotely.

CHAPTER 10

REFERENCES

[1]. Alex, B., Zongyuan, G., Lionel, O., Fabio, R., Ben, U. (2016), “Simple online and realtime tracking”, Proceedings of the Image Processing (ICIP) 2016 IEEE International Conference, 3464-3468.

[2]. Davis, J., Goadrich, M. (2006), “The relationship between Precision-Recall and ROC curves”, Proceedings of the 23rd International Conference on Machine Learning, 233- 240.

[3]. Lee, K.B., Shin, H.S. and Kim, D.K. (2018), “Development of a deep-learning based automatic tracking of moving vehicles and incident detection processes on tunnels”, J. Kor. Tunnel Eng., KTA, 20(6), 1161-1175.

[4]. Lee, K.B., Shin, H.S (2019), “Effect on self-enhancement of deep-learning inference by repeated training of false detection cases in tunnel accident image detection”, J. Kor. Tunnel Eng., KTA, 21(3), 419-432.

[5]. Ministry of Land, Infrastructure and Transport (MOLIT) (2016), “Guideline of installation of disaster prevention facilities on road tunnels”.

[6]. Ren, S., He, K., Girshick, R., Sun, J. (2015), “Faster R-CNN: Towards real-time object detection with region proposal networks”, Proceedings of the Advances in Neural Information Processing Systems, 91-99.

[7]. Simonyan, K., Zisserman, A. (2014), “Very deep convolutional networks for large-scale image recognition”, arXiv preprint, arXiv: 1409.1556.

[8]. Zhu, M. (2004), “Recall, precision and average precision”, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo 2: 30.

[9]. Zhao, Z. Q., Zheng, P., Xu, S. T., Wu, X. (2019), “Object detection with deep learning: A review”, arXiv preprint, arXiv: 1807.05511.

[10]. YOLOv7 Official GitHub Repository, *https://github.com/WongKinYiu/yolov7*, Accessed 2025.

[11]. PyTorch Documentation, *https://pytorch.org/docs/stable/index.html*, Accessed 2025.

[12]. OpenCV Documentation, *https://docs.opencv.org/*, Accessed 2025.

[13] P. Rani, B. K. Tripathy, *Real-time Traffic Accident Detection System Using Deep Learning Techniques*, International Journal of Engineering and Technology (IJET), 2018.

[14] H. Singh, A. Yadav, *Automated Road Accident Detection Techniques: A Survey*, International Research Journal of Engineering and Technology (IRJET), 2020.

[15] Navneet Dalal, Bill Triggs, *Histograms of Oriented Gradients for Human Detection*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[16] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition (VGGNet)*, International Conference on Learning Representations (ICLR), 2015.

[17] Olaf Ronneberger, Philipp Fischer, Thomas Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, MICCAI, 2015.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, *SSD: Single Shot MultiBox Detector*, ECCV, 2016.

[19] Ren Shaoqing, Kaiming He, Ross Girshick, Jian Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, NeurIPS, 2015.

[20] A. Krizhevsky, I. Sutskever, G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks (AlexNet)*, Advances in Neural Information Processing Systems, 2012.