COLLEGE OF APPLIED BUSINESS & TECHNOLOGY

Affiliated to Tribhuvan University Kathmandu, Nepal

Project Report on

“Vehicle Collision Detection and Reporting System”

In partial fulfillment of the requirement for the Bachelor Degree in Computer Science and Information Technology

Submitted to:

Department of Computer Science and Information Technology

College of Applied Business & Technology

Submitted By:

Arnav Sharma (26039)

Kushal Basnet (26049)

Lotus Kshetri (26050)

February 2025

ACKNOWLEDGEMENT

We would like to extend our heartfelt gratitude to all those who have provided invaluable support throughout the course of this project. First and foremost, we wish to express our deepest appreciation to our project supervisor, Mr. Indra Chaudhary, and coordinator Mr. Santosh Sharma for his constant guidance, insightful suggestions, and unwavering encouragement, which have been instrumental in the successful completion of this report.

We would also like to acknowledge the support and resources provided by the faculty and staff of CAB-Tech. Their permission to use the necessary materials and infrastructure has been essential to the progress of our work.

Our sincere thanks also go to the faculty members and seniors of the B.Sc. CSIT department for their continuous support, advice, and inspiration, which have significantly contributed to the successful development of this project.

Finally, we would like to express our appreciation to our colleagues and peers for their constant motivation and collaborative spirit throughout the project. Their encouragement has been a constant source of strength in overcoming challenges.

With sincere respect,
Arnav Sharma
Kushal Basnet
Lotus Kshetri

ABSTRACT

This project presents the development of an automated vehicle collision detection system using deep learning techniques to improve road safety and minimize the impact of traffic accidents. The system is based on the Faster R-CNN architecture with a ResNet-50 backbone for accurate vehicle and collision detection from recorded video footage. The model classifies vehicles into four categories—car, truck, bus, and motorcycle—and identifies collision events in real-time. The system leverages OpenCV for frame processing and Flask for API integration, enabling a user-friendly web interface for authorities to upload video footage, view detection results, and analyze incidents. By automating the collision detection process, the system reduces human error, enhances emergency response times, and supports more effective traffic management. Future enhancements include real-time video processing, environmental adaptation, and integration with IoT sensors to further optimize system performance in diverse scenarios.

Keywords: *Automated Systems, Collision Event, Deep Learning, Faster R-CNN, Flask API, Machine Learning, Object Detection, OpenCV, Real-time Monitoring, ResNet-50, Traffic Management, Traffic Safety, Vehicle Classification, Vehicle Collision Detection, Video Analysis, Web Application*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

R-CNN	Region-Based Convolutional Neural Network
ResNet	Residual Network
OpenCV	Open-Source Computer Vision Library
API	Application Programming Interface
IoT	Internet of Things
RPN	Region Proposal Network
YOLO	You Only Look Once
WHO	World Health Organization
CNN	Convolutional Neural Network
CCTV	Closed-Circuit Television
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
SSD	Solid-State Drive

CHAPTER 1 INTRODUCTION

1.1. Project Introduction

Vehicle collision detection plays a crucial role in enhancing road safety and minimizing accident-related damages. With the increasing number of vehicles on the road, timely and accurate detection of collisions has become essential for both preventive measures and post-incident analysis. Traditional methods rely heavily on manual video review, which is time-consuming, subjective, and prone to human error. To address these challenges, this project focuses on the development of an automated vehicle collision detection system using deep learning techniques.

The core of this system is a model built from scratch based on the Faster R-CNN architecture with a ResNet-50 backbone (fasterrcnn_resnet50_fpn). This two-stage object detection framework integrates a Region Proposal Network (RPN) to generate candidate bounding boxes and a Fast R-CNN head to classify these proposals and refine their coordinates. The model is designed to detect four types of vehicles—car, truck, bus, and motorcycle—while also identifying collision events. The training process involved two stages: initially training the model on a large vehicle dataset, followed by fine-tuning with a custom dataset specifically labeled for collision detection.

The implementation leverages OpenCV for frame processing and bounding box generation, enabling the system to process recorded video footage effectively. Additionally, a Flask API facilitates seamless integration with a front-end interface developed using HTML, CSS, JavaScript, and Bootstrap, providing an interactive platform for users to upload videos, view detection results, and analyze collision events efficiently.

1.2. Problem Statement

Vehicle collision detection is crucial for enhancing road safety and minimizing accident-related damages. Traditional methods, such as manual video analysis, are time-consuming and prone to human error, leading to delayed responses and increased risks. In Nepal, road accidents are a significant concern, with an average of 75 incidents daily, resulting in approximately seven fatalities per day [1].

According to the World Health Organization's Global Status Report on Road Safety 2023, road accidents remain one of the leading causes of death globally, with increasing traffic volume and inadequate safety measures contributing to the rising fatalities [2].

To address these challenges, an automated machine learning-based system is needed that can accurately detect and classify vehicle collisions in real time. This system should be trained on a custom dataset to differentiate between actual accidents and normal driving behavior, thus minimizing false positives and ensuring the efficient allocation of resources. Furthermore, a user-friendly interface is essential for authorities to monitor and manage detection results, enabling prompt emergency responses and better traffic management.

This project aims to develop a machine learning system that will accurately detect and classify vehicle collisions in real time. The system will also ensure that non-accident scenarios are not mistakenly identified as accidents, helping to reduce false alarms. A key aspect of the project will be the creation of an intuitive interface for authorities, allowing for effective monitoring, management, and reporting of detected collisions.

1.3. Objectives of the project

The objectives of the project are:

- To develop a deep learning model based on Faster R-CNN with a ResNet-50 backbone for the detection and classification of vehicle collisions from recorded video footage
- To create a web application using Flask and a front-end interface that enables users to upload videos, view collision detection results, and analyze incidents in real time.

1.4. Scope and Limitations of the project

The scope and limitations of the project are:

1.4.1. Scope

- The model detects vehicle collisions from recorded video footage and classifies them by identifying four types of vehicles: car, truck, bus, and motorcycle.
- The system is designed to assist authorities in real-time collision detection and post incident analysis by providing accurate and timely results.
- The web application allows users to upload videos, view detection results, and analyze collision events through an interactive interface.

1.4.2. Limitations

- The project does not account for environmental factors (such as weather conditions, road visibility, or lighting) that may affect the accuracy of collision detection in certain situations.
- The system is limited to detecting vehicle collisions based on visual data and may not perform well in cases of high-speed, blurry, or low-quality video footage.
- The system is intended to assist authorities in detecting collisions, not to replace manual review or expert judgment. Final accident analysis should be conducted by qualified personnel.

1.5. Development Methodology

The project employs an Incremental Model, which allows for systematic development of the vehicle collision detection system through a series of defined stages. Each stage focuses on specific tasks, ensuring focused development, testing, and integration.

Iteration 1:

The project began with the collection of data and the preparation of a robust dataset. Extensive research was conducted on various deep learning techniques, including one-shot neural networks, ResNet, and Faster R-CNN architectures, to determine the most suitable model for collision detection. A landing page wireframe was designed as part of the user interface, ensuring a seamless user experience for uploading and interacting with video data. Additionally, YOLO was initially utilized to evaluate the dataset for any inherent issues, ensuring the quality of the images and labels.

Iteration 2:

The second phase shifted focus to model training. Utilizing the findings from previous research, the Faster R-CNN architecture with a ResNet-50 backbone was employed to train the model for vehicle and collision detection. The model was initially trained for two epochs to verify the correctness of the architecture and assess its functionality. Key tests were performed to validate the model's capability to process video data, detect vehicles, and classify them. This phase was crucial for ensuring that the base model was functioning as expected.

Iteration 3:

In this phase, the model was further refined by training it for 10 epochs to enhance its performance and ensure stability. The system's architecture was optimized by adjusting hyperparameters based on earlier results. Additionally, a Flask API was developed to handle all tasks related to the model, such as video uploading, collision detection, and interaction with the trained deep learning model. The API integration allowed for seamless backend functionality, ensuring that the web interface could efficiently interact with the model for real-time collision detection.

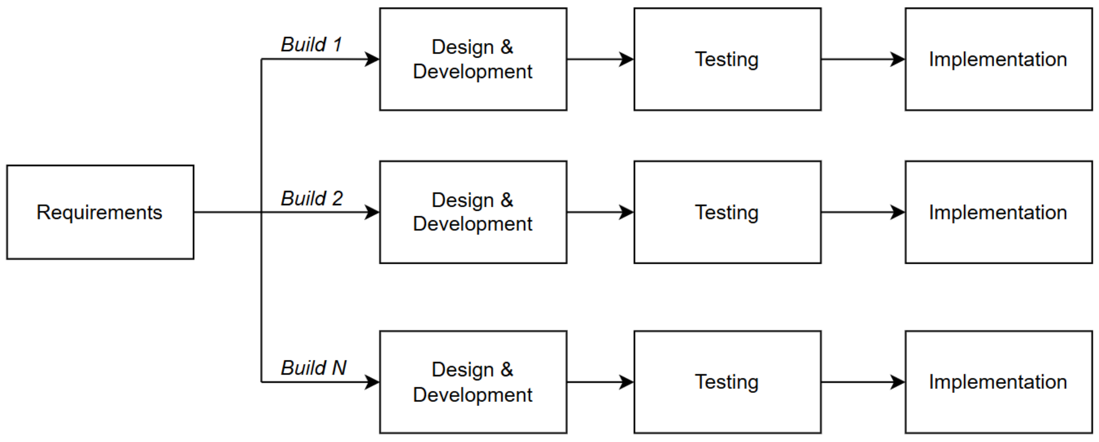


Figure 1.1 Development Methodology

1.6. Report Organization

The report is as follows:

The report is divided into five different chapters, each representing a different development phase of the project. The chapter can be discussed briefly as follow:

Chapter 1: INTRODUCTION - In the first section we present an outline of the recommended frameworks and their motivation. It describes the problem that the system needs to solve, and it gives insight into the objective, scope and limitations of the system.

Chapter 2: REQUIREMNT ANALYSIS AND FESIBILITY STUDY – The second chapter describes the pre-existing systems, system analysis, functional requirement and nonfunctional requirement, feasibility study (Technical requirement and non-functional requirements).

Chapter 3: SYSTEM ANALYSIS - The third chapter describes the system analysis, including the evaluation of existing systems, proposed system overview, system architecture, data flow analysis, use case analysis, and system constraints and assumptions.

Chapter 4: SYSTEM DESIGN - In the third section we give depiction and subtleties of the plan for our system. This chapter contains the diagram for system analysis and design. It also gives the basic information or details about the algorithm.

Chapter 5: IMPLEMENTATION - In the fourth section we show the execution of our system in various technologies. This chapter contains the implementation of the cosine similarity algorithm, unit and integration testing of the component is done and the result of the project is also summarized in this chapter.

Chapter 6: CONCLUSION AND FUTURE ENHANCEMENT – This chapter contains the conclusion of the project and further enhancements.

CHAPTER 2 REQUIREMENT ANALYSIS AND FEASIBILITY ANALYSIS

2.1. Background Study

Vehicle collisions are a significant cause of injury and death worldwide, with increasing road traffic contributing to a growing number of accidents. According to the World Health Organization (WHO), road traffic accidents cause over 1.35 million deaths annually, with millions more suffering from injuries. In Nepal, the rising number of vehicles and insufficient traffic management have exacerbated the problem, leading to higher accident rates. The need for early detection and timely response to collisions has never been more crucial, particularly in urban areas like Kathmandu, where road congestion and unsafe driving practices increase the likelihood of accidents.

Traditional methods of detecting vehicle collisions rely on manual inspection of video footage or surveillance cameras, which is time-consuming, prone to human error, and often impractical in real-time scenarios. As a result, there has been a significant push to develop automated systems that can detect and analyze vehicle collisions using deep learning techniques. One such approach is the use of deep neural networks, specifically Convolutional Neural Networks (CNNs), which have shown exceptional performance in various computer vision tasks.

Models like ResNet-50 and Faster R-CNN have gained prominence for object detection tasks. ResNet-50, a deep residual network, enables efficient training of deep neural networks by mitigating the vanishing gradient problem through the use of residual connections. Faster R-CNN, on the other hand, is an object detection framework that integrates a Region Proposal Network (RPN) to efficiently generate candidate bounding boxes for detected objects. By fine-tuning these models on custom datasets, such as vehicle collision images, it is possible to achieve high accuracy in detecting vehicle types and collision events.

The growing interest in automating vehicle collision detection, especially in the context of traffic monitoring systems in Nepal, has the potential to significantly enhance road safety. By reducing the reliance on manual labor and increasing the accuracy and speed of detection, such systems can provide timely information to authorities, potentially saving lives and reducing accident-related damages.

2.2. Machine Learning

Machine learning (ML) is a vital subfield of artificial intelligence (AI) that focuses on enabling computers to learn from data and improve their performance over time without being explicitly programmed. By utilizing algorithms and statistical models, ML systems can identify patterns and make predictions based on historical data. [3].

The process of machine learning involves several key steps: data collection, preparation, model selection, training, evaluation, and deployment. Initially, large datasets are gathered and cleaned to ensure quality input for the algorithms. Subsequently, models are trained to recognize relationships within the data, which enables them to make accurate predictions on unseen data. This iterative cycle of training and refinement is crucial for developing robust ML applications that can adapt to new information and changing environments [4].

2.3. Deep Learning

Deep learning (DL) is an advanced subset of machine learning (ML) that utilizes artificial neural networks to process vast amounts of data, mimicking the way the human brain operates. By employing multi-layered structures, deep learning algorithms can analyze complex patterns and make predictions without requiring explicit programming.

This hierarchical approach allows DL to excel in tasks involving unstructured data, such as images, audio, and text, making it a powerful tool for applications in speech recognition, natural language processing, and image classification [5].

2.4. Faster R-CNN

Faster R-CNN is an advanced object detection model that integrates Region Proposal Networks (RPN) with Convolutional Neural Networks (CNNs) to efficiently identify and localize objects. This integration allows the model to generate regional proposals and perform detection in a unified, end-to-end trainable framework, enhancing both speed and accuracy [6].

A key innovation is the use of anchor boxes, enabling the model to detect objects of various sizes [7]. Sharing convolutional features reduces redundancy, making it faster than R-CNN and Fast R-CNN, establishing it as a benchmark in object detection [8].

2.5. Related Works

Various research studies have explored the potential of deep learning techniques, particularly CNNs, in detecting and classifying objects in images, with applications ranging from medical imaging to autonomous driving. In the context of vehicle collision detection, several papers have utilized architectures like Faster R-CNN and ResNet-50 to improve detection accuracy and efficiency.

One notable study by Girshick et al. [9] introduced Faster R-CNN, which significantly improved the speed and accuracy of object detection by incorporating a Region Proposal Network (RPN). This approach reduces the reliance on external region proposal methods, enabling the model to generate bounding boxes directly from image data. Another study by He et al. [10] presented ResNet, which utilizes residual connections to prevent the degradation of performance as the depth of the network increases. ResNet-50, in particular, has been widely adopted for various image classification tasks, demonstrating its ability to extract complex features from images while maintaining computational efficiency.

In a study focusing on accident detection, Mitra et al. [11] employed the YOLOv3 algorithm for detecting accidents, achieving impressive accuracy in identifying vehicle rollover, rearend, and head-on collisions. However, this approach had limitations in terms of processing speed and the ability to handle diverse environmental conditions. On the other hand, Faster R-CNN has been shown to outperform YOLO in certain scenarios, particularly when finegrained object detection is needed [12]. In a similar vein, Zheng et al. employed a combination of Faster R-CNN and ResNet-50 to detect accidents, improving the model's ability to identify collisions in real-time video streams.

Further studies have demonstrated the effectiveness of integrating Faster R-CNN with ResNet-50 for detecting vehicles in challenging conditions, such as crowded roadways or low-visibility environments [13]. These advancements have paved the way for the development of collision detection systems that can operate in real-time, offering valuable support to law enforcement and traffic management authorities.

Additionally, some studies have explored the application of these models in Nepal's context, where the lack of sufficient surveillance infrastructure and the challenge of managing high traffic volumes contribute to higher accident rates. By utilizing deep learning models like Faster R-CNN and ResNet-50, it is possible to automate the detection and reporting of accidents, enabling more efficient responses from emergency services and reducing the burden on human resources.

Recent advancements in deep learning, particularly the use of transfer learning techniques, have also contributed to improving the performance of collision detection models. Transfer learning allows models like ResNet-50, pretrained on large datasets such as ImageNet, to be fine-tuned on specific datasets like vehicle collision images, significantly improving detection accuracy with limited labeled data [14].

In conclusion, the literature demonstrates that Faster R-CNN and ResNet-50 are highly effective models for vehicle collision detection, offering significant advantages in terms of accuracy, speed, and robustness. The integration of these models into automated traffic monitoring systems, particularly in regions like Nepal, has the potential to revolutionize road safety by providing timely and accurate collision detection and analysis.

CHAPTER 3 SYSTEM ANALYSIS

3.1. System Analysis

3.1.1. Requirement Analysis

a. Functional Requirement

- Use Case Diagram

  In figure no. 3.1, the emergency service personnel interact with the system by uploading CCTV footage through the system interface.

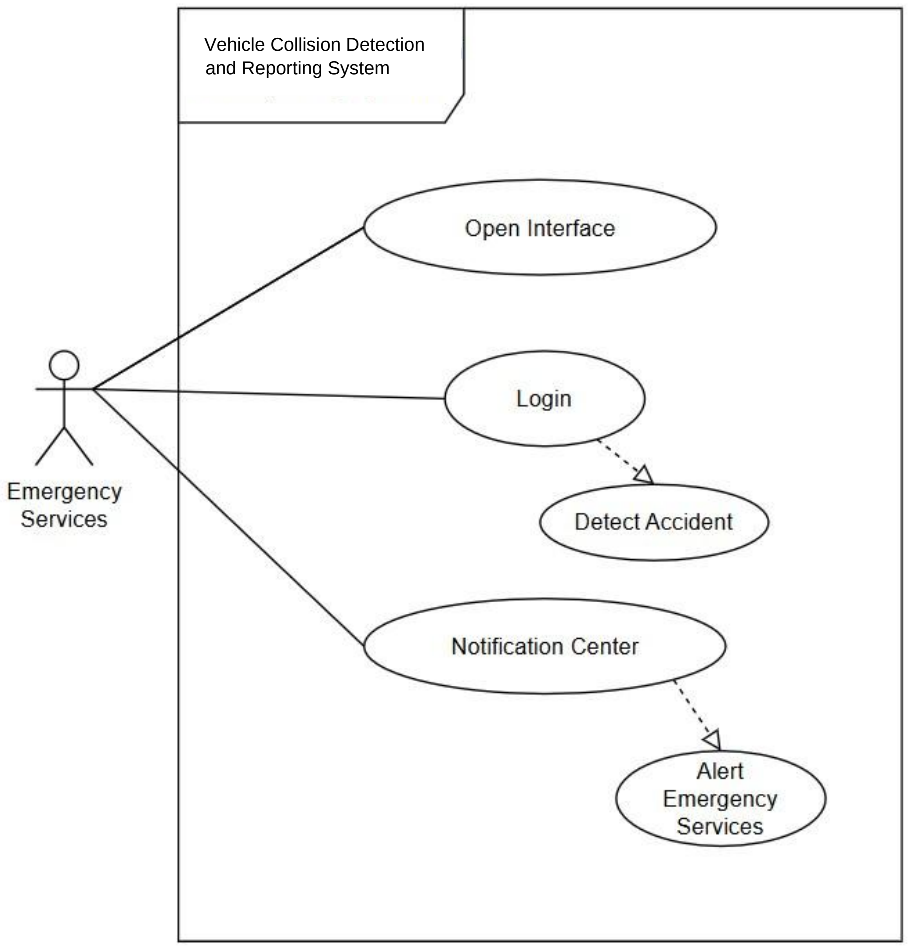


Figure 3.1 Use Case Diagram of Vehicle Collision Detection and Reporting System

Use case: Vehicle Collision Detection	
Primary Actor:	Emergency Service Administrator
Flow of Events	1. The administrator is responsible for monitoring large number of traffic CCTV footage. The admin schedules CCTV footage to feed into the system, to reduce the workload, and perform fast accident collision and notification. 2. Admin uploads video as input to the system. 3. The system processes the uploaded video. 4. The system extracts relevant features from the system. 5. The fine-tuned model classifies the video frame to detect any collisions. 6. If the system detects a collision, an alert is sent to the emergency services.
Pre-condition	The system must be operational, and the video must be in the mp4 format.
Post-condition	The system accurately classifies vehicles and collisions and provides a reliable classification.
Quality	The classification must be accurate and precise.

b. Non-Functional Requirements

- Performance:

  The system processes video frames and generates collision detection predictions within a defined time frame to ensure real-time monitoring. For example, video processing and collision prediction should be completed within 24 milliseconds per frame to support real-time applications.

- Usability:

  The system provides a user-friendly interface for authorities and traffic management personnel to upload video feeds, view detection results, and notify other emergency services flawlessly.

- Compatibility:

  The system is compatible with different operating systems and devices to accommodate various user environments (e.g. systems useable from both CPU and GPU).

3.1.2. Feasibility Analysis

Technical Feasibility

The technical feasibility of the proposed vehicle collision detection system assesses whether the system can be developed and implemented using current technology and available resources.

- Hardware Feasibility
  From a hardware perspective, the system requires a high-performance CPU to manage model execution, real-time video processing, and system logic efficiently. A dedicated GPU with CUDA support, such as an NVIDIA RTX series, is essential to accelerate the performance of deep learning tasks, particularly for running the Faster R-CNN model used for collision detection. Additionally, a minimum of 16GB RAM is recommended to handle large datasets, process high-resolution video frames smoothly, and support complex computations. Storage requirements include at least 50GB of SSD to ensure fast data access for storing trained models, temporary video data, and logs at least during training.
- Software Feasibility
  In terms of software feasibility, the system is primarily developed using Python due to its extensive libraries and ease of integration with machine learning frameworks. PyTorch is utilized as the core deep learning library to train and deploy the Faster R-CNN model efficiently, while Scikit-learn supports model evaluation tasks. For handling numerical computations and data manipulation, NumPy is employed, and Matplotlib is used for data visualization to analyze the performance of the model. The system’s back end is developed using the Flask framework to create a

lightweight and efficient REST API, enabling smooth communication between the model and the front-end interface. On the front-end, the Bootstrap framework ensures a responsive and user-friendly interface for monitoring collision detection results. The development environment includes Kaggle, and Google Colab's Jupyter notebook as primary model training interface, Visual Studio Code as the primary IDE for writing and debugging code, with Git serving as the version control system to manage code changes and facilitate collaborative development.

This combination of hardware and software resources ensures that the system is technically feasible, scalable, and capable of delivering real-time, accurate collision detection.

Operational Feasibility

The vehicle collision detection system works by analyzing video footage from cameras. The video frames are first taken and converted to frames and passed to the system. Following preprocessing, the frames are passed through a Faster R-CNN model with a ResNet-50 backbone, which performs object detection to identify vehicles such as cars, trucks, buses, motorcycles, and vehicle collisions. These results are displayed on a user-friendly interface as a notification pop-up, accessible through a web application built with Flask and Bootstrap, and an alert is also sent to the relevant emergency service, allowing authorities or end-users to monitor collision events in real time and make informed decisions promptly.

Economic Feasibility

The economic feasibility of the vehicle collision detection system assesses the financial resources needed for its development and deployment. This includes costs for hardware, such as a capable CPU, at least 8 GB of RAM, and a dedicated GPU to handle real-time video processing efficiently. Expenses also cover storage solutions to manage large video datasets. On the software side, the project relies on open-source libraries like PyTorch, OpenCV, and Flask, reducing licensing costs.

Schedule Feasibility

The Gantt chart illustrates the work that is completed over a period of time in relation to the time planned for the work in the future.

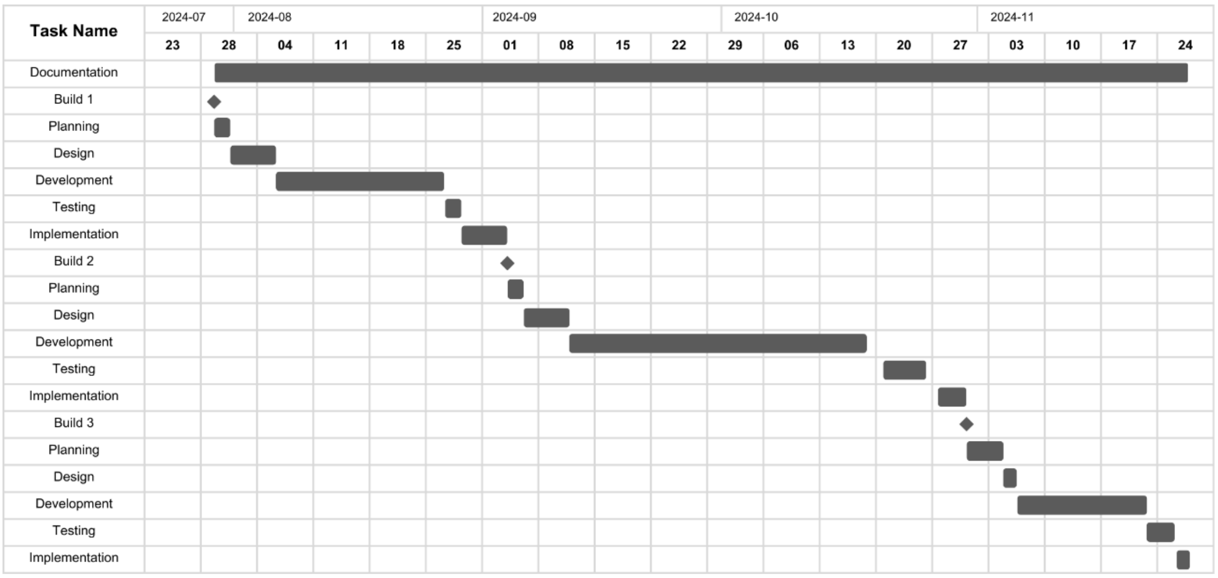


Figure 3.2 Gantt Chart

3.1.3. Analysis

- Dynamic modelling using Sequence Diagrams

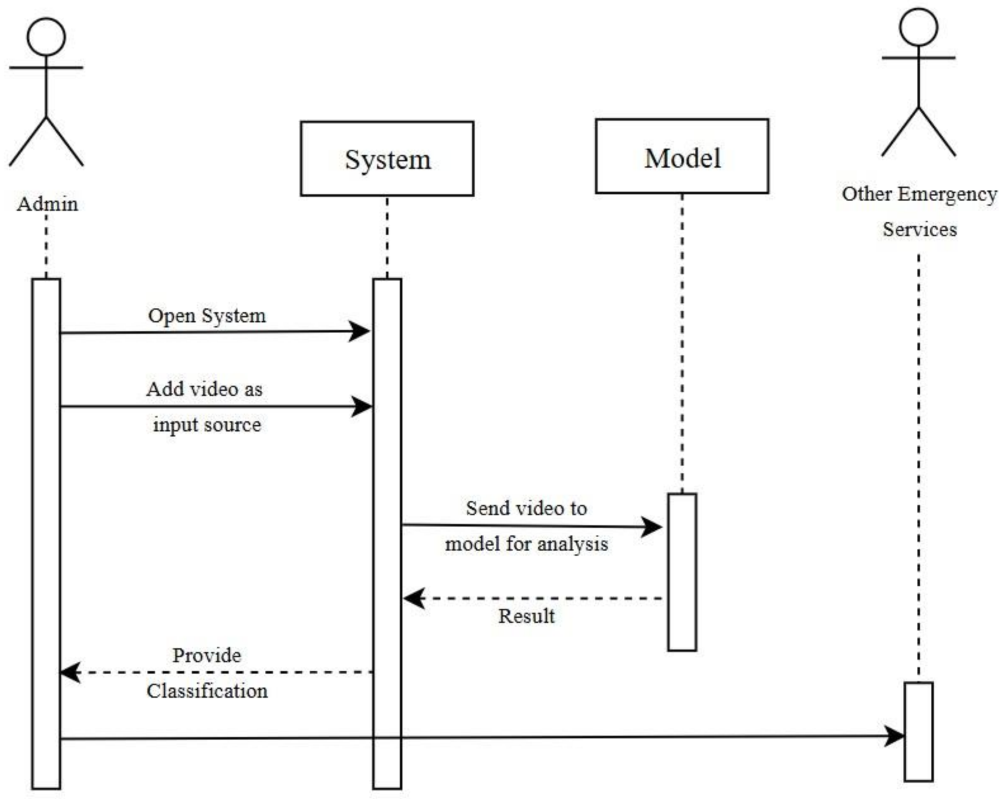


Figure 3.3 Sequence Diagram

- Process modelling using Activity Diagram

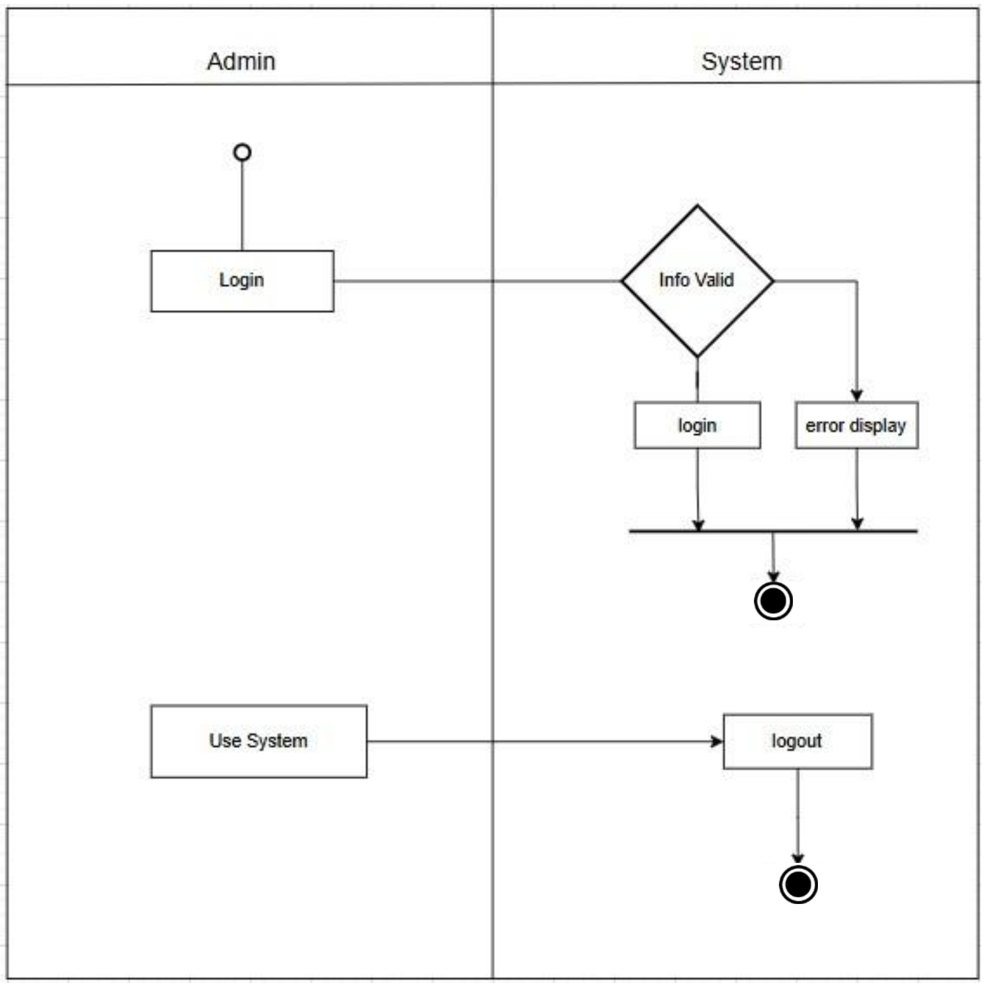


Figure 3.4 Activity Diagram

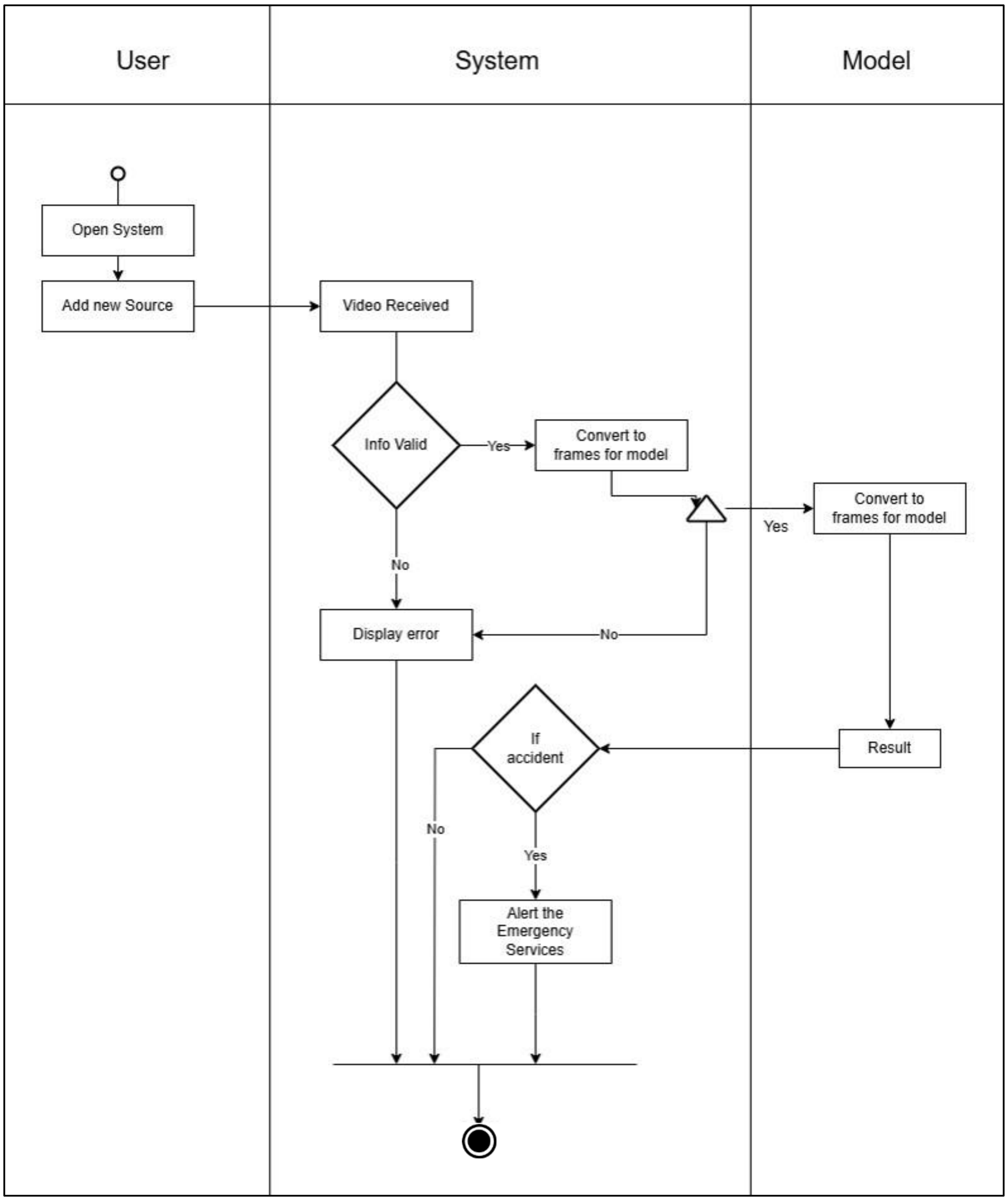


Figure 3.5 Activity for System Usage

CHAPTER 4 SYSTEM DESIGN

4.1. Design

System design is the process of developing architecture for the various parts, interfaces, and modules of the system as well as offering relevant information that is useful for putting such parts into systems.

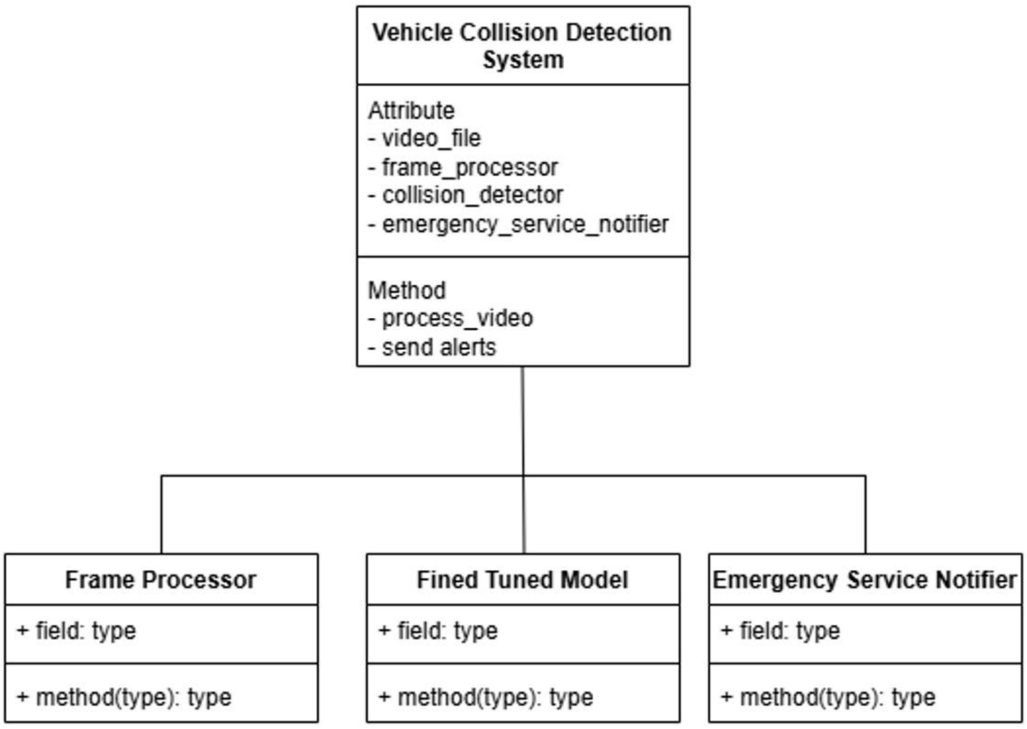


Figure 4.1 System Class Diagram

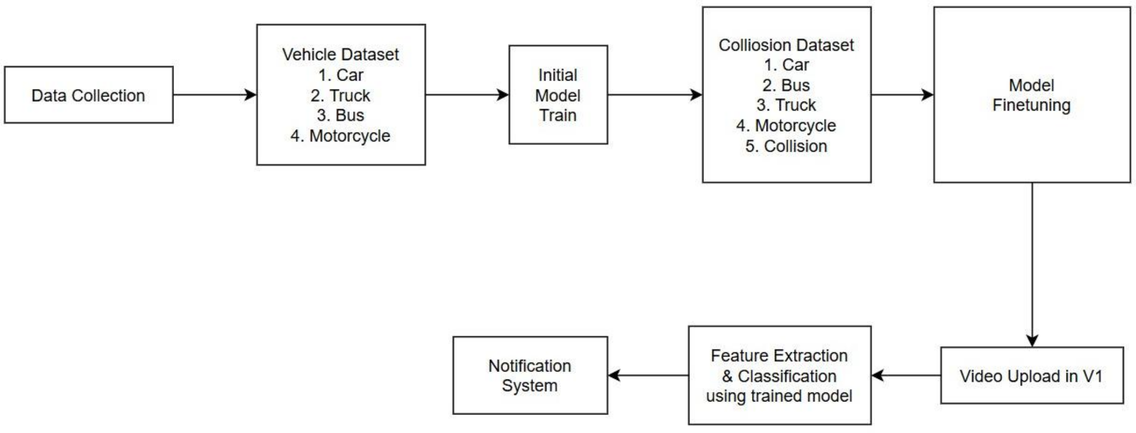


Figure 4.2 System Design

The system architecture in the above figure can be explained as:

- Data Collection

  Vehicle and collision data were collected for model training and prediction.

  - Vehicle Dataset

    The image dataset selected for vehicle prediction consists of the following 4 classes:

    - Car

    - Truck

    - Bus

    - Motorcycle

• Initial Model Train

FasterRCNN with resnet50 backbone was trained on the dataset. Image preprocessing and augmentation were performed on the data before the training model. The models were trained for around 10 epoch, monitoring for overfitting and underfitting, and the final model was selected.

○ Collision Dataset

The image dataset for collision contained the following 5 classes:

- Car

- Bus

- Truck

- Motorcycle

- Collision

- Model Finetuning

  The previously trained model was finetuned on this dataset, by removing all the box predictors from the classification layer and adding a fifth class to the model architecture to predict. Similar image processing and augmentation were applied here as well before the training. Model was trained for 10 epoch while continually monitoring for over and underfitting.

- Video Upload in UI

  The administrator uploaded the CCTV footage from the UI. This acts as input for the model.

- Feature Extraction and Classification

  Features are extracted from the video and feed to the model for classification. The model classifies the input image into one of the 5 classes: bus, car, truck, motorbike and collision.

- Notification System

  If a collision was detected on the system during the classification, a notification is sent to the concerned emergency service agency.

4.2. Algorithm Details

Faster R-CNN (Faster Region-Convolutional Neural Network) is a state-of-the-art deep learning-based object detection algorithm introduced by Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun in 2015 as an improvement over previous R-CNN variant. Designed for high accuracy and efficiency, Faster R-CNN unifies object detection and localization by integrating deep convolutional neural networks (CNNs) with Region Proposal Networks (RPNs). This cohesive architecture significantly enhances both speed and precision, making it highly effective for various computer vision tasks such as vehicle detection, pedestrian recognition, and collision detection. Its ability to efficiently identify and precisely locate objects within an image makes it a suitable choice for detecting vehicle crashes in real-world scenarios.

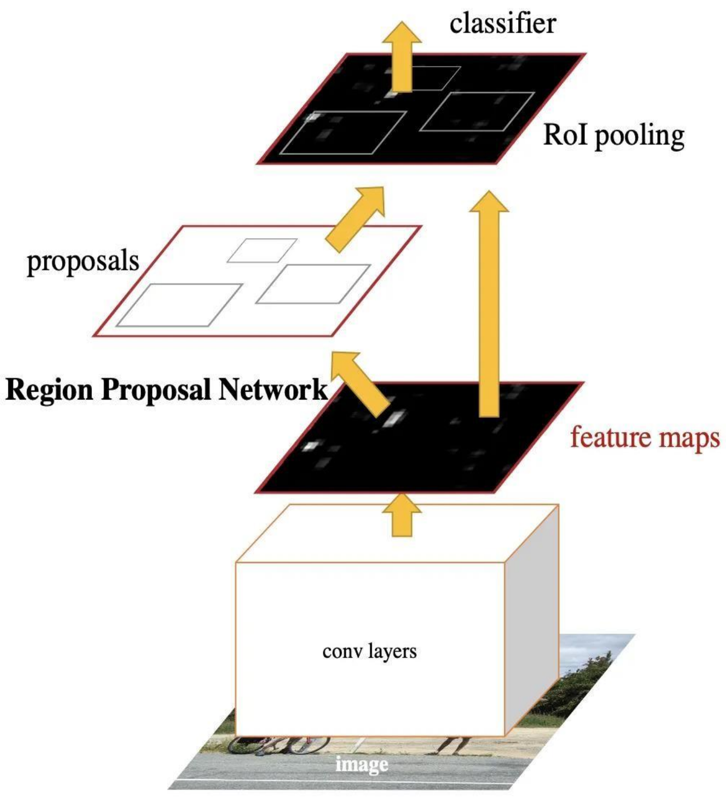


Figure 4.3 Faster Region-Convolutional Neural Network Architecture

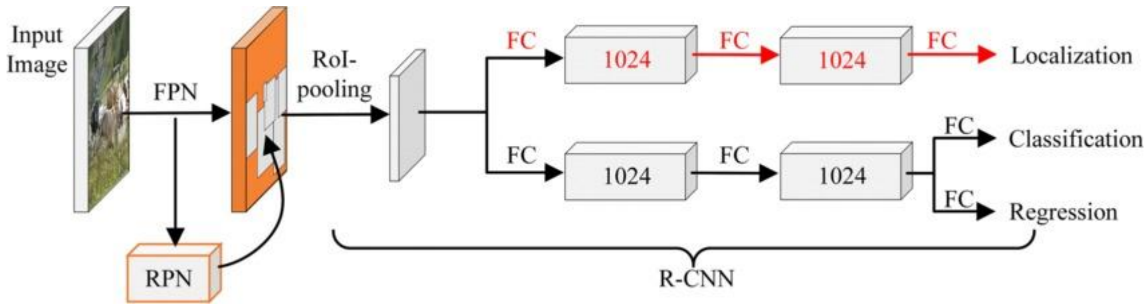


Figure 4.4 Faster Region-Convolutional Neural Network Architecture 2

Faster R-CNN Model Architecture

1. Input Layer

The Faster R-CNN model requires preprocessed input images to ensure compatibility and optimal performance. Preprocessing plays a crucial role in enhancing model efficiency by maintaining consistency in image dimensions and improving feature extraction.

- **Resizing:** All input images are resized to a fixed resolution, typically 600x600 or 1024x1024 pixels. This standardization helps maintain uniformity during model training and inference.
- **Normalization:** Pixel values are normalized, often scaling them between 0 and 1 or standardizing based on the dataset mean and standard deviation. This process enhances model stability and speeds up convergence during training.

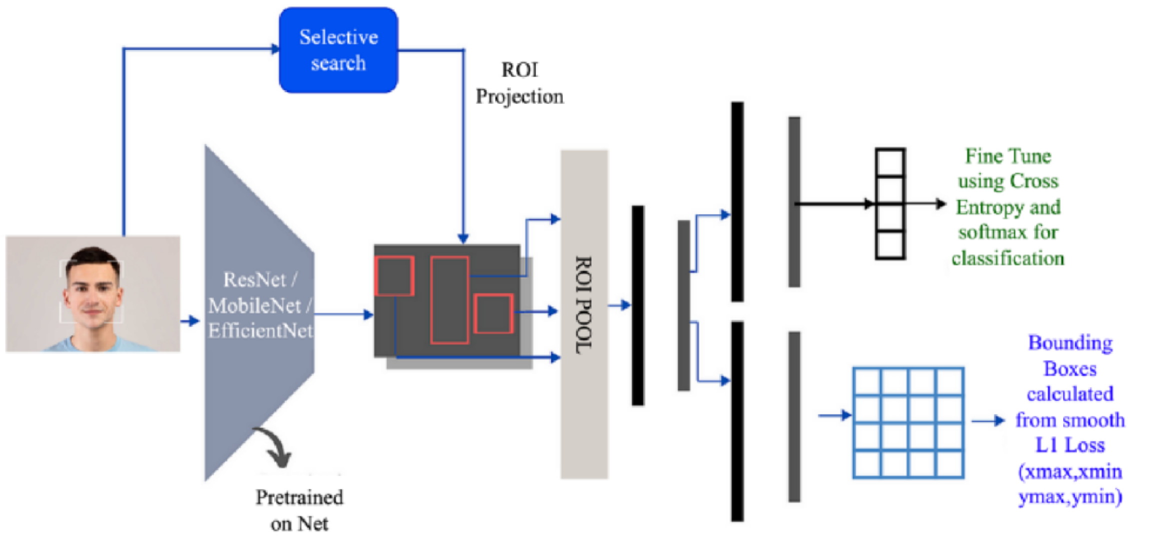

Selective search
ROI Projection
ResNet / MobileNet / EfficientNet
Pretrained on Net
ROI POOL
Fine Tune using Cross Entropy and softmax for classification
Bounding Boxes calculated from smooth L1 Loss (xmax,xmin ymax,ymin)

Figure 4.5 Faster R-CNN architecture

2. Feature Extraction with CNN Backbone

Faster R-CNN leverages a deep Convolutional Neural Network (CNN) as its backbone to extract key patterns from input images. Common backbone architecture includes **ResNet50**, **ResNet-101**, and **VGG16**. Here ResNet-50 backbone architecture was utilized.

- **Convolutional Layers:** The input image is passed through multiple convolutional layers, which extract spatial and texture-based features. These layers detect edges, textures, shapes, and more complex patterns as the network deepens.
- **Feature Maps:** The output of these layers is a feature map, representing the key characteristics of the input image, which is then passed to the Region Proposal Network (RPN) for object detection.

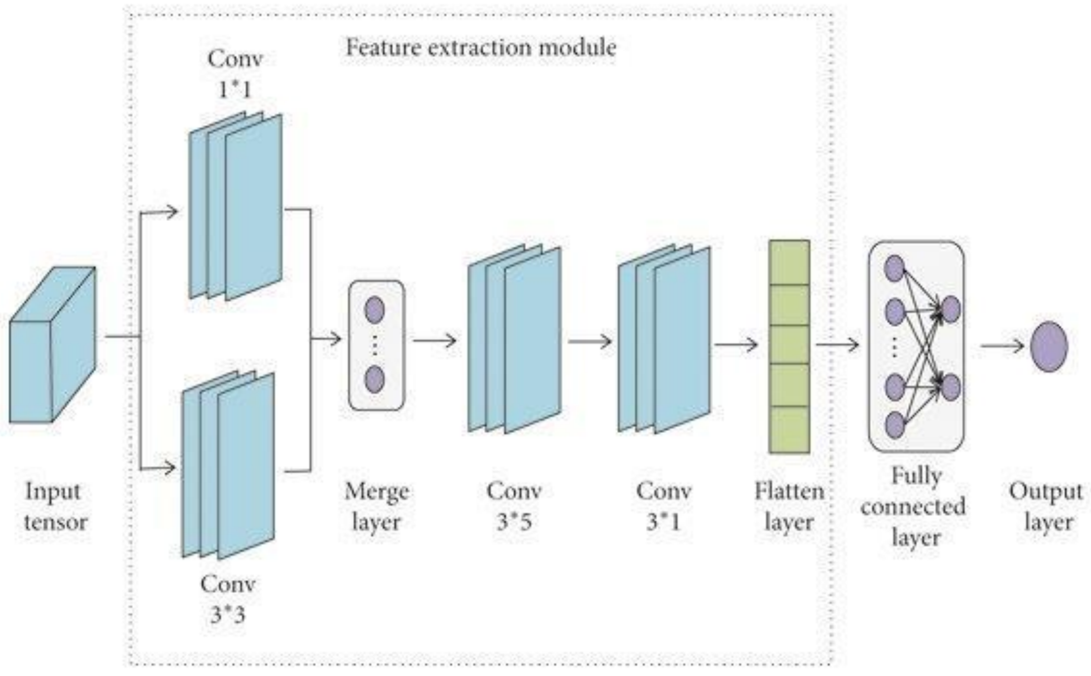


Figure 4.6 Structure of CNN feature extraction module

3. Region Proposal Network (RPN)

The **Region Proposal Network (RPN)** is a key component of Faster R-CNN, responsible for generating potential object locations, known as Regions of Interest (RoIs).

- **Input:** Takes the feature maps from the CNN backbone.

- **3x3 Convolutional Layer:** Slides over the feature map to capture spatial features and generate dense object proposals.
- **Anchor Boxes:** The RPN uses predefined bounding boxes, known as anchor boxes, with various scales and aspect ratios to detect objects of different sizes.
- **Output:** Generates RoIs, classifying them as foreground (potential object) or background (no object).

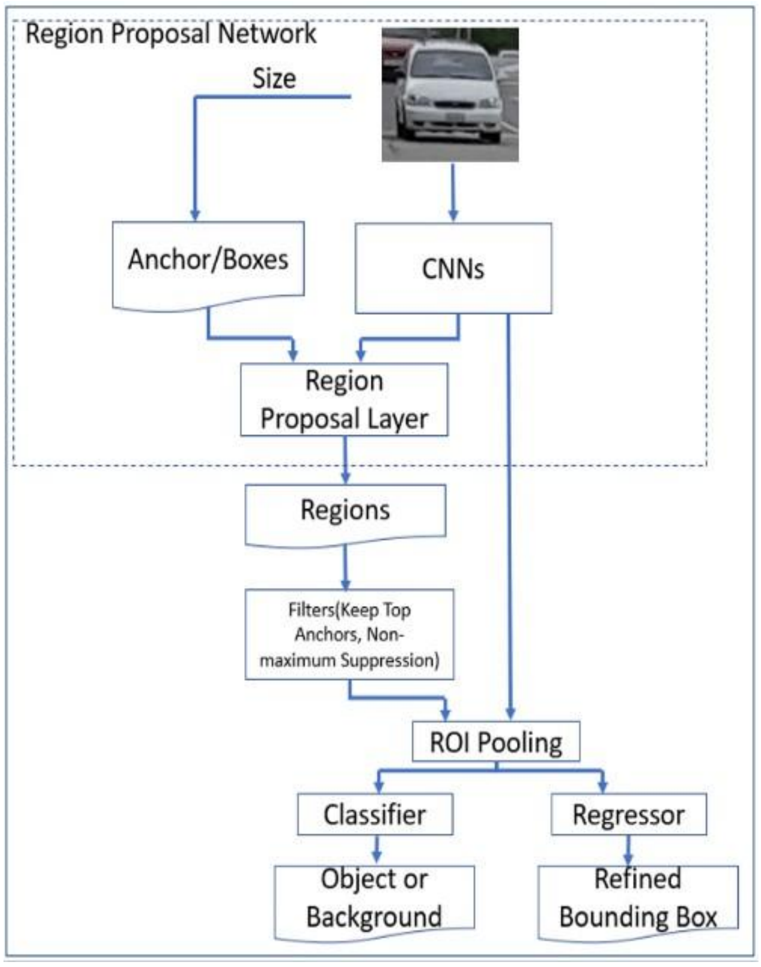


Figure 4.7 RPN Architecture with Anchor Boxes

4. ROI Pooling & Feature Refinement

The RoIs generated by the RPN vary in size, which presents a challenge when passing them through fully connected layers that require fixed-size inputs. To address this:

- **ROI Pooling:** This operation converts RoIs into a fixed size (e.g., 7x7 pixels) by dividing each region into a grid and applying max-pooling within each grid cell.

- **Feature Refinement:** The pooled features are then refined to improve detection accuracy before classification and bounding box regression.

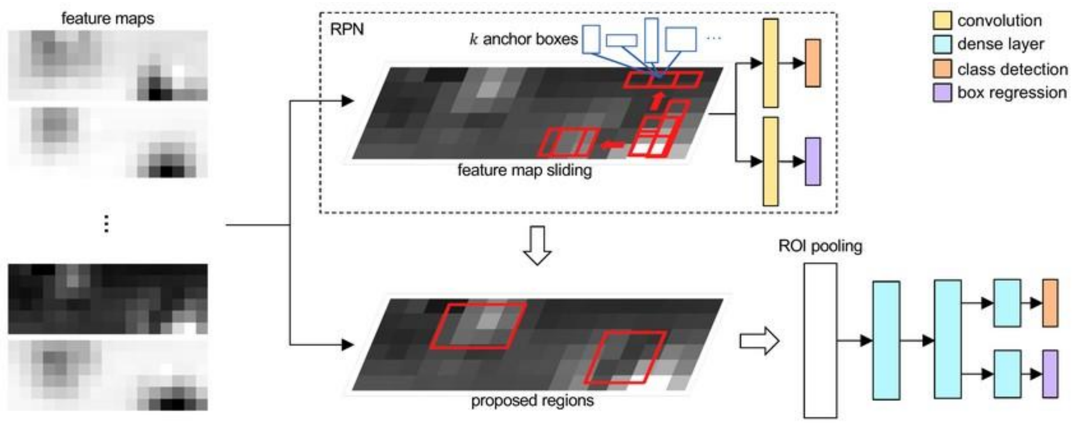


Figure 4.8 ROI Pooling Process

5. Classification & Bounding Box Regression

After ROI pooling, each fixed-size feature map passes through fully connected layers, leading to two parallel outputs:

1. **Classification Head:**
   - Determines the object class (e.g., car, truck, motorcycle, collision event).
   - Uses a softmax activation function for multi-class classification.
2. **Bounding Box Regression Head:**
   - Refines the bounding box coordinates for precise localization. ○ Applies a regression function to adjust the box’s position, width, and height to better fit the detected object.

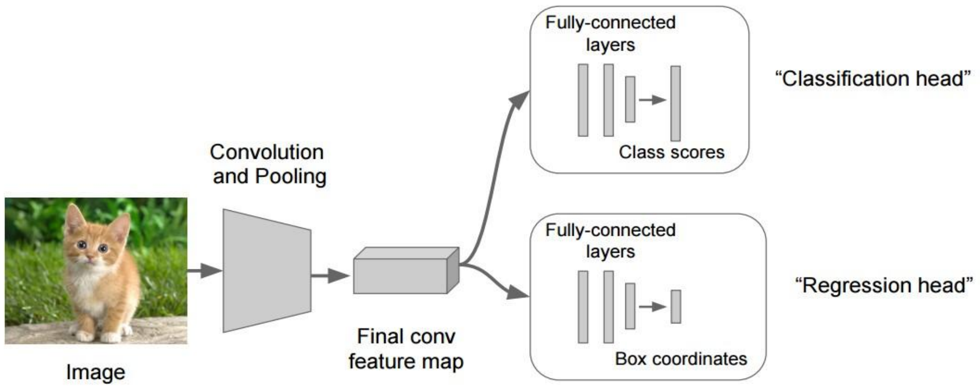


Figure 4.9 Modified Faster R-CNN Architecture Diagram

6. Final Output

The final output of the Faster R-CNN model consists of:

- **Detected Objects & Labels:** Identifies objects in the image, such as vehicles or collision events.
- **Bounding Box Coordinates:** Specifies the location of each detected object.
- **Confidence Scores:** Provides a probability score indicating the model's confidence in each detection.

This comprehensive architecture enables Faster R-CNN to perform real-time object detection with high accuracy, making it suitable for applications like vehicle collision detection.

CHAPTER 5 IMPLEMENTATION AND TESTING

5.1. Implementation

5.1.1. Tools used

- Software Development Tools

  The primary programming language used for developing the vehicle collision detection system is **Python**, owing to its versatility and extensive support for machine learning and deep learning tasks. The project leverages several powerful libraries, including **PyTorch** for building, training, and fine-tuning the Faster R-CNN model, **NumPy** for efficient numerical computations, **Scikit-learn** for evaluation metrics and data preprocessing, and **OpenCV** for image processing and manipulation tasks.

  For the API layer, **FastAPI** is employed to create a lightweight yet robust RESTful API, enabling smooth interaction between the frontend and backend systems. The frontend interface is designed using **Streamlit, HTML**, **CSS**, **JavaScript**, and the **Bootstrap** framework to ensure responsiveness, ease of use, and an intuitive user experience.

- Hardware Development Tools

  The model training and experimentation are conducted on **Kaggle's cloud infrastructure**, utilizing **T4 GPUs** within the Kaggle **Jupyter Notebook** environment. This setup provides the computational power necessary for training deep learning models efficiently. Additionally, a **dedicated 16 GB GPU** is used for intensive training tasks, significantly reducing model training time and improving performance during inference.

- Diagram Tools

  For visual representation and system design documentation, **Draw.io** is used to create various **Unified Modeling Language (UML) diagrams**, including class diagrams, sequence diagrams, and use case diagrams. This tool also supports the development of additional diagrams such as architectural diagrams and workflow diagrams, aiding in better understanding and communication of system processes.

- Dataset

  The system is trained on two datasets. The primary **vehicle dataset** is sourced from **Kaggle**, consisting of **17,000 images** covering four vehicle classes: **car, truck, bus, and motorcycle**. To enhance the system's capabilities for collision detection, a secondary **self-annotated collision dataset** containing **4,000 images** was created using **Roboflow**. The model was initially trained on the vehicle dataset and later fine-tuned to include collision detection, expanding the classification categories to encompass both vehicle types and collision events.

5.1.2. Implementation Detail and Modules

- Dataset Preparation Module

  This module is responsible for organizing and preparing the dataset for training the object detection model. It loads images and their corresponding label files, ensuring proper mapping. If an image lacks a corresponding label file, an empty one is created for consistency. Label conversion is performed from the YOLO format (center coordinates and dimensions) to the Faster R-CNN format (bounding box coordinates). Data augmentation techniques, including random flipping, rotation, and color jittering, are applied to improve model robustness.

- Data Loading and Augmentation Module

  This module manages the efficient loading of training, validation, and testing data using PyTorch's Dataset and DataLoader classes. It standardizes preprocessing by resizing images to a resolution of 416x416, converting them into PyTorch tensors, and normalizing pixel values. A custom collate_fn is implemented to handle batch processing, ensuring that images and targets are correctly padded for optimal training efficiency.

- Object Detection Model Module

  This module defines the Faster R-CNN model architecture. A pre-trained fasterrcnn_resnet50_fpn model is used, with its classification head modified to match the dataset's specific number of classes (vehicles, accidents, and background). The model is adapted for multi-task learning, enabling both vehicle detection and accident classification. Additionally, it ensures efficient computation by assigning the model to the appropriate hardware (GPU or CPU) and integrates logic for both training and inference.

- Training and Validation Module
  This module is responsible for training the Faster R-CNN model. It defines loss functions for both vehicle classification and accident detection, computing the total loss for optimization. The training loop processes images and targets in batches, optimizing the model using the SGD optimizer with momentum and weight decay. A learning rate scheduler dynamically adjusts the learning rate to enhance convergence. The validation phase assesses model performance on unseen data, tracking loss values to monitor overfitting and generalization.

5.2. Testing

Testing is a phase of the software development lifecycle that deals with verifying that the product or the application that has been developed is free of errors or problems. Such a step is also important in ensuring that the software is sufficient to fulfill the specified functionalities as well as working correctly under different situations.

5.2.1. Test Cases for Unit Testing

Table 5.1 Test Cases for Dataset Integration

S.N.	**Test Case ID**	**Test Description**	**Input Test Data**	**Expected Results**	**Actual Results**	**Remarks**
1	TC- 1	Load training dataset	Path to training dataset	Training dataset loaded successfully	Training dataset loaded successfully	Pass
2	TC- 2	Load testing dataset	Path to testing dataset	Testing dataset loaded successfully	Testing dataset loaded successfully	Pass

Table 5.2 Test Cases for Image Preprocessing

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC-3	Resize images to 416 x 416	Image of various sizes	Images resized to 416 x 416	Images resized to 416 x 416	Pass
2	TC-4	Convert annotation to ResNET format	Annotation of all images	Annotations converted to RESnet format	Annotations converted to RESnet format	Pass

Table5.3 Test Cases for Data Augmentation

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC-5	Apply horizontal flipping, random rotation, and brightness adjustment	Images of various sizes	Augmented images with horizontal flip, random rotation, and brightness adjustment applied	Augmented images generated	Pass

5.2.2. Test cases for System Testing Table

Table 5.4 Test Cases for Vehicle Collision Detection Classification Model

S.N.	Test Case ID	Test Description	Input Test Data	Expected Results	Actual Results	Remarks
1	TC-6	Classify Vehicle Bus	Vehicle labeled as Bus	Predicted label: Bus	Predicted label: Bus	Pass
2	TC-7	Classify Vehicle Car	Vehicle labeled as Car	Predicted label: Car	Predicted label: Car	Pass
3	TC-8	Classify Vehicle Truck	Vehicle labeled as Truck	Predicted label: Truck	Predicted label: Truck	Pass
4	TC-9	Classify Vehicle Motorcycle	Vehicle labeled as Motorcycle	Predicted label: Motorcycle	Predicted label: Motorcycle	Pass
5	TC-10	Handle invalid login attempts	User enters incorrect credentials	Error message indicating invalid credentials	Error message indicating invalid credentials	Pass

5.2.3. Confusion Matrix of Testing Data for Training

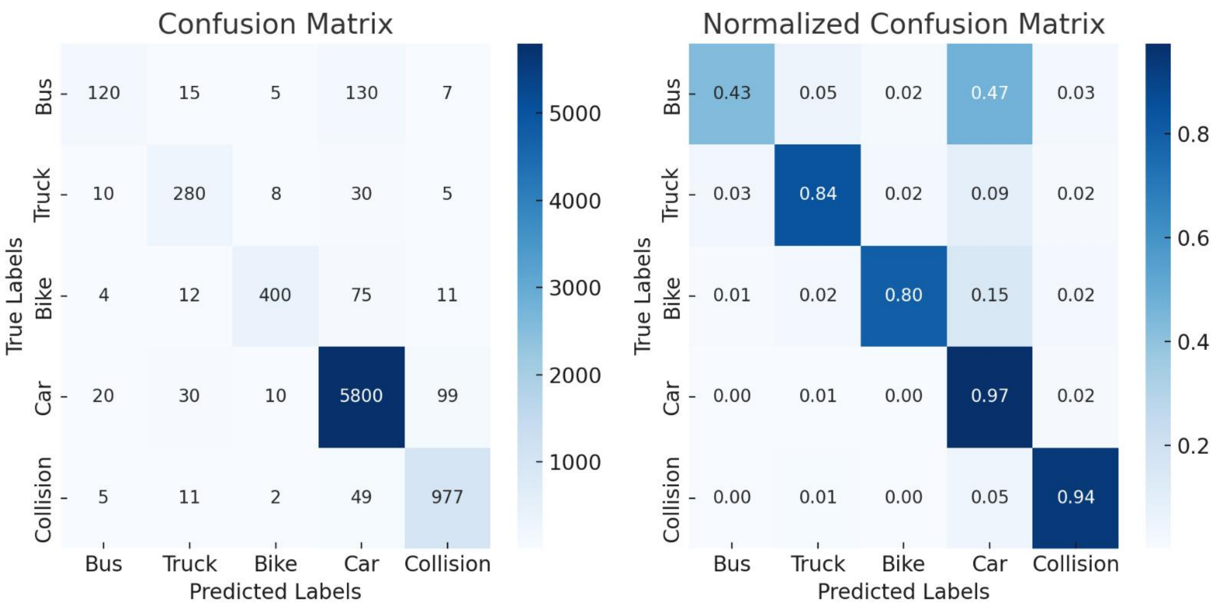


Figure 5.1 Confusion Matrix of Testing Data for Training

CHAPTER 6 CONCLUSION AND FUTURE ENHANCEMENT

6.1. Conclusion:

In conclusion, this project successfully developed an automated vehicle collision detection system leveraging deep learning techniques, specifically Faster R-CNN with a ResNet-50 backbone, to accurately detect and classify vehicle collisions from video footage. The system provides timely and reliable collision detection, aiding authorities in real-time monitoring and decision-making. By integrating machine learning with a user-friendly web interface, the system enhances road safety and contributes to more efficient traffic management. Future work could focus on improving the system's adaptability to various environmental conditions and optimizing real-time processing capabilities.

6.2. Future Enhancement:

The future recommendations for the vehicle collision detection system are listed as follows:

- **Real-time Video Processing**: Optimizing the model and system to handle live video streams in real-time, reducing latency and enabling immediate collision detection and alerts, which would further improve emergency response times.
- **Integration with IoT Sensors**: Integrating the system with IoT devices (e.g., vehicle sensors, traffic cameras, or road sensors) could provide additional data points, improving the accuracy and robustness of collision detection, especially in complex traffic scenarios.
- **Environmental Adaptation**: Enhancing the model to handle different weather conditions (e.g., rain, fog, snow) and varying lighting conditions (e.g., night-time) to ensure more accurate detection in diverse environments.
- **Extended Collision Classification**: Expanding the system’s ability to classify a wider range of accident types, such as pedestrian accidents, vehicle rollovers, or rear-end collisions, to provide more comprehensive accident analysis.
- **Automatic Incident Reporting**: Developing automatic incident reporting capabilities that can send detailed collision reports to authorities, along with video footage and vehicle information, to streamline emergency response coordination.

- **Model Optimization for Low-Resolution Footage**: Enhancing the model's performance with low-quality or low-resolution video feeds, ensuring effective detection even from older or low-cost surveillance systems.

REFERENCES

[1] "Nepal Sees at Least 75 Road Accidents on Average Daily, Report Shows," The Kathmandu Post, Jan. 26, 2025. [Online]. Available: https://kathmandupost.com/national/2025/01/26/nepal-sees-at-least-75-roadaccidents-on-average-daily-report-shows [Accessed: Jan. 31, 2025].

[2] World Health Organization, "Global Status Report on Road Safety 2023," 2023. [Online]. Available: https://www.who.int/teams/social-determinants-of-health/safety-andmobility/global-status-report-on-road-safety-2023 [Accessed: Feb. 2, 2025].

[3] “Machine learning explained | MIT Sloan,” *MIT Sloan*, Apr. 21, 2021. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (accessed Dec. 08, 2024).

[4] L. Craig and L. Tucci, “What is machine learning? Guide, definition and examples,” *Search Enterprise AI*, Aug. 16, 2024. https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML (accessed Dec. 24, 2024).

[5] J. Kriese, T. Hoeser, S. Asam, P. Kacic, E. Da Da Ponte, and U. Gessner, “Deep learning on synthetic data enables the automatic identification of deficient forested windbreaks in the Paraguayan Chaco,” *Remote Sensing*, vol. 14, no. 17, p. 4327, Sep. 2022, doi: 10.3390/rs14174327.

[6] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv.org*, Jun. 04, 2015. https://arxiv.org/abs/1506.01497 (accessed Dec. 28, 2024).

[7] GeeksforGeeks, “Faster RCNN | ML,” *GeeksforGeeks*, Aug. 23, 2023. https://www.geeksforgeeks.org/faster-r-cnn-ml/ (accessed Jan. 06, 2025).

[8] G. Boesch, “The Fundamental Guide to Faster R-CNN [2025],” *viso.ai*, Dec. 06, 2024. https://viso.ai/deep-learning/faster-r-cnn-2/ (accessed Jan. 06, 2025).

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142-158, Jan. 2016.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[11] R. Mitra, A. S. Bedi, and G. Gupta, "Accident detection using YOLO," IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2023, pp. 215-221.

[12] Z. Zheng, Y. Li, W. Xu, and J. Wang, "Vehicle collision detection using Faster RCNN with ResNet-50," IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 5, pp. 1495-1503, May 2018.

[13] J. Liu, Y. Chen, and X. Li, "Real-time vehicle accident detection using Faster RCNN with ResNet-50 backbone," Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019, pp. 1349-1357.

[14] H. Liu, L. Sun, and J. Xu, "Vehicle detection and collision recognition with Faster R-CNN in real-time traffic monitoring," IEEE Access, vol. 8, pp. 99834 99843, 2020.

APPENDICES

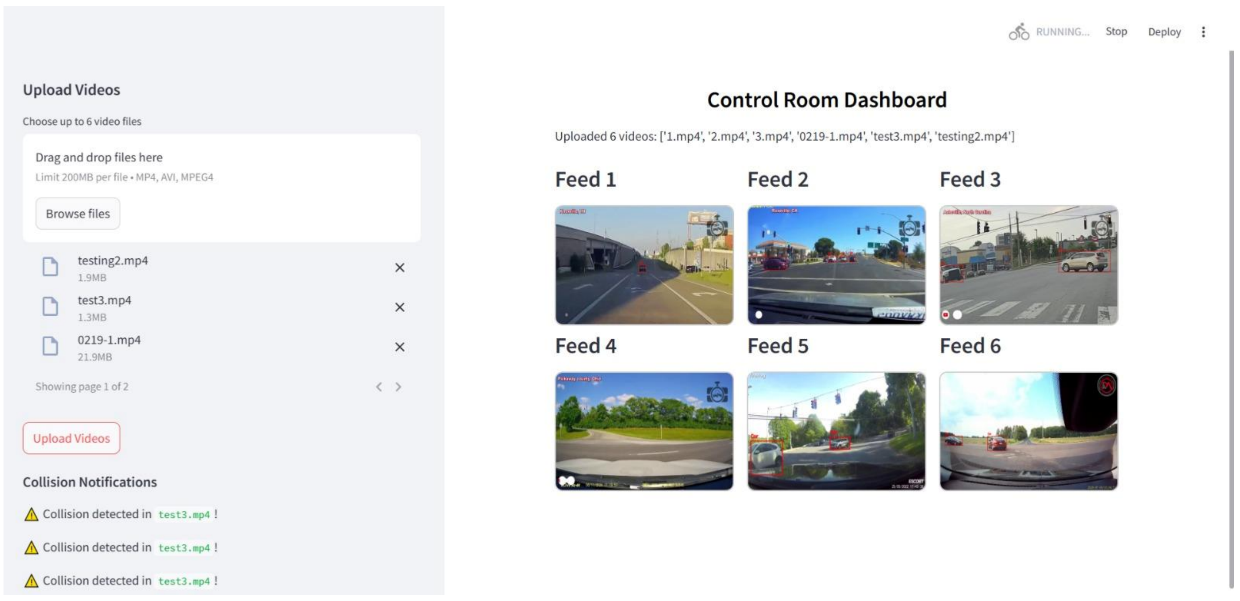
Backend Code

```
from fastapi import FastAPI, File, UploadFile, HTTPException

from fastapi.responses import StreamingResponse, JSONResponse

import os

import cv2

import torch

import torchvision

from torchvision.transforms import functional as F


# Initialize FastAPI

app = FastAPI()


# Configuration

UPLOAD_FOLDER = "uploads"

os.makedirs(UPLOAD_FOLDER, exist_ok=True)


# Load the trained model

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model_path = "fasterrcnn_finetuned_epoch10.pth"


# Define the model architecture

num_classes = 5  # Bus, Truck, Motorcycle, Car, Collision

model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=None)
```

```
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)

# Load the model state dictionary
model.load_state_dict(torch.load(model_path, map_location=DEVICE, weights_only=False))
model.to(DEVICE)
model.eval()

# Define class labels and colors
class_labels = ["Bus", "Truck", "Car", "Collision", "Motorcycle"]
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0, 255, 255)]

# Dictionaries to track collision counters and status
collision_counters = {}  # Tracks collision counters for each video (0 or 1)

# Process video and yield frames with bounding boxes at 1 FPS
def process_video(input_path, filename):
    cap = cv2.VideoCapture(input_path)
    if not cap.isOpened():
        raise ValueError(f"Could not open video file: {input_path}")
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_count = 0
    while cap.isOpened():
```

```
    ret, frame = cap.read()
    if not ret:
        break
    # Perform inference only once per second (1 FPS)
    if frame_count % int(fps) == 0:
        img_tensor = F.to_tensor(frame).unsqueeze(0).to(DEVICE)
        with torch.no_grad():
            predictions = model(img_tensor)
        pred_boxes = predictions[0]["boxes"].cpu().numpy()
        pred_scores = predictions[0]["scores"].cpu().numpy()
        pred_labels = predictions[0]["labels"].cpu().numpy()
        # Draw bounding boxes on the current frame
        for i in range(len(pred_scores)):
            if pred_scores[i] > 0.5:  # Confidence threshold
                x1, y1, x2, y2 = map(int, pred_boxes[i])
                label_index = pred_labels[i] - 1
                label = class_labels[label_index]
                color = colors[label_index]
                cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
                text = f"{label}"
                cv2.putText(frame, text, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, color, 2)
                if label == "Collision":
                    # Update collision counter only if it's currently 0
                    if collision_counters.get(filename, 0) == 0:
                        collision_counters[filename] = 1  # Set counter to 1
                        print(f"Collision Detected in video `{filename}`!")
        # Encode frame as JPEG and yield it for streaming
        _, buffer = cv2.imencode('.jpg', frame)
        if not _:
            continue
        frame_bytes = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')
    frame_count += 1
cap.release()


# Endpoint to upload video
@app.post("/upload/")
async def upload_video(file: UploadFile = File(...)):
file_location = f"{UPLOAD_FOLDER}/{file.filename}"
with open(file_location, "wb+") as file_object:
    file_object.write(file.file.read())
# Initialize collision counter for the uploaded video
collision_counters[file.filename] = 0
return {"message": "Video uploaded successfully", "filename": file.filename}


# Stream processed video
```

```
@app.get("/video_feed")
async def video_feed(filename: str):
    input_path = f"{UPLOAD_FOLDER}/{filename}"  # Use the uploaded video's filename
    if not os.path.exists(input_path):
        raise HTTPException(status_code=404, detail="Video not found.")
    return StreamingResponse(
        process_video(input_path, filename),
        media_type="multipart/x-mixed-replace; boundary=frame"
    )

# Endpoint to check collision status
@app.get("/check_collision/{filename}")
async def check_collision(filename: str):
    if filename not in collision_counters:
        return JSONResponse(content={"collision_detected": False}, status_code=404)

    # Check if the collision counter is 1 (collision detected but not yet reported)
    if collision_counters[filename] == 1:
        collision_counters[filename] = 2  # Mark as "already notified"
        print(f"Collision reported for `{filename}`.")
        return JSONResponse(content={"collision_detected": True})

    # If the collision has already been reported, return False
    return JSONResponse(content={"collision_detected": False})
```



Upload Videos

Choose up to 6 video files

Drag and drop files here

Limit 200MB per file • MP4, AVI, MPEG4

testing2.mp4 1.9MB

test3.mp4 1.3MB

0219-1.mp4 21.9MB

Collision Notifications

⚠ Collision detected in `test3.mp4` !

⚠ Collision detected in `test3.mp4` !

⚠ Collision detected in `test3.mp4` !

Control Room Dashboard

Uploaded 6 videos: ['1.mp4', '2.mp4', '3.mp4', '0219-1.mp4', 'test3.mp4', 'testing2.mp4']

Figure A.1 Application Interface