# A Mini Project Report on

**AI – POWERED ACCIDENT DETECTION: TRAFFIC SURVEILLANCE IMAGES USING CNN WITH VGG16**

*Submitted to*

**JAWAHARLALNEHRUTECHNOLOGICALUNIVERSIT, HYDERABAD**

#### In Partial fulfillment of the requirement for the award of degree of the



**IN**



*By*

**M. AKHIL VARMA -22RA1A6701**

**J. VINEEL BHARATH -22RA1A6762**

**K. VENU -22RA1A6760**

Under the guidance of

**Mrs.P.Vyshali**

**(Asst. Professor)**

Department of Computer Science and

 Engineering (Data Science)

**Department of Computer Science and Engineering -(Data Science) KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY**

#### (Affiliated to JNTUH, Ghanpur(V), Ghatkesar(M), Medchal(D)-501301)

**2022-2026**

**KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY**

**(Affiliated to JNTUH, Ghanpur(V), Ghatkesar(M), Medchal(D)-501301)**

****

# CERTIFICATE

It is certified that the project work entitled “**AI – POWERED ACCIDENT DETECTION:TRAFFIC SURVEILLANCE USING CNN WITH VGG16”** carried out by **Mr.M.Akhil Varma, Mr.J.Vineel Bharath, Mr.K.Venu,** bonafide students of **Kommuri Pratap Reddy Institute of Technology** in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Technology (Data Science)** of the **Jawaharlal Nehru Technological University, Hyderabad** during the year 2024-2025. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

**Internal Guide HOD**

**Mrs.P.Vyshali Mr.B.Ramesh**

**External Examiner**

# DECLARATION

We hereby declare that this project work entitled **“AI – powered accident detection: traffic surveillance images using cnn with vgg16”** in partial fulfillment of requirements for the award of degree of **Computer Science and Engineering (Data Science)** is a bonafide work carried out by us during the academic year 2024-2025.

We further declare that this project is a result of our effort and has not been submitted for the award of any degree by us to any institute.

By

M. Akhil Varma (22RA1A6701)

J. Vineel Bharath (22RA1A6762)

K. Venu (22RA1A6760)

# ACKNOWLEDGEMENT

It gives us immense pleasure to acknowledge with gratitude, the help and support extended throughout the project report from the following:

We will be very much grateful to almighty our **Parents** who have made us capable of carrying out our job.

We express our profound gratitude to **Dr. E. RAVINDRA**, **Principal of Kommuri Pratap Reddy Institute of Technology**, who has encouraged in completing our project report successfully.

We are grateful to **Mr. B. Ramesh** who is our **Head of the Department, CSE(DS)** for his amiable in genius and adept suggestions and pioneering guidance during the project report.

We express our deep sense of gratitude and thanks to Internal Guide, **Mrs.P.Vyshali, Assistant Professor** for her guidance during the project report.

We are also very thankful to our **Management**, **Staff Members** and all **Our Friends** for their valuable suggestions and timely guidance without which we would not have been completed it.

By

**M. Akhil Varma (22RA1A6701)**

**J. Vineel Bharath (22RA1A6762)**

**K. Venu (22RA1A6760)**



**Vision of the Institute**

To emerge as a premier institute for high quality professional graduates who can contribute to economic and social developments of the Nation.

##### Mission of the Institute

|  |  |
| --- | --- |
| **Misson** | **Statement** |
| **IM1** | To have holistic approach in curriculum and pedagogy through industry interface to meet the needs of Global Competency. |
| **IM2** | To develop students with knowledge, attitude, employability skills entrepreneurship, research potential and professionally Ethical citizens. |
| **IM3** | To contribute to advancement of Engineering& Technology that would help to satisfy the societal needs. |
| **IM4** | To preserve, promote cultural heritage, humanistic values and Spiritual values thus helping in peace and harmony in the society. |



**Vision of the Department**

To produce excellent standard, quality education of professionals by imparting cognitive learning environment, ethical, research and industrial orientation to become pioneering Data Scientists.

**Mission of the Department**

|  |  |
| --- | --- |
| **Mission** | **Statement** |
| **DM1** | Data science is an [interdisciplinary](https://en.wikipedia.org/wiki/Interdisciplinarity) field that uses scientific methods, processes ,algorithms and systems to extract [knowledge](https://en.wikipedia.org/wiki/Knowledge) by using structured and [unstructured data](https://en.wikipedia.org/wiki/Unstructured_data). |
| **DM2** | Data science which depend on the application. More recently full-featured, end-to-end platforms have been developed and heavily used for data science and machine learning. |
| **DM3** | To facilitate the programming skills by imparting the qualitative technicality in theoretical and pragmatically approaches. |
| **DM4** | Enabling students to get expertise in critical skills with data science education and facilitate socially responsive research and innovation. |



**Program Educational Objectives (PEOs)**

|  |  |
| --- | --- |
| **PEOs** | **Statement** |
| **PEO1** | Data Science are becoming the hot topics of the IT industry as the entire world is relying on data. |
| **PEO2** | The data-driven operations adopted by almost every industry will increase the scopes of well-trained professionals. |
| **PEO3** | Data Science platform that provides open-source software and computing resources which Data Scientists can use and remain updated with the latest developments in the field. |



**Program Outcomes**

|  |  |
| --- | --- |
| PO1 | **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| PO2 | **Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| PO3 | **Design/development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. |
| PO4 | **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| PO5 | **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations. |
| PO6 | **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| PO7 | **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental context,and demonstrate the knowledge of, and need for sustainable development. |
| PO8 | **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |



|  |  |
| --- | --- |
| PO9 | **Individual and team network:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |
| PO10 | **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. |
| PO11 | **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. |
| PO12 | **Life-Long learning:** Recognize the need for, and have the preparation and able to engage in independent and life-long learning in the broadest context of technological change. |

**Program Specific Outcome’s**

|  |  |
| --- | --- |
| PSO1 | To prepare the students ready for industry usage by providing required training in cutting edge technologies |
| PSO2 | An Ability to use the core concepts of computing and optimization techniques to develop more efficient and effective computing mechanisms. |
| PSO3 | An Ability to use inculcates professional, social, ethical, effective communication skills and entrepreneurial practice among their holistic growth. |

# ABSTRACT

In India, road accidents remain a leading cause of death, with over 150,000 fatalities reported annually, according to the Ministry of Road Transport and Highways. Shockingly, more than 80% of these deaths result not from the accidents themselves but from delayed emergency response and lack of immediate aid. Currently, accident detection often relies on manual observation through CCTV feeds, eyewitness reports, or emergency calls—methods that are not only slow but also unreliable on less trafficked roads, especially at night or in remote areas. These manual systems are plagued by inefficiencies, human error, and delayed communication, resulting in critical loss of life-saving time. Motivated by the urgent need for faster, more reliable responses, this project proposes an AI-powered accident detection system utilizing deep learning and computer vision. By deploying CCTV cameras on roads and highways and integrating them with a real-time image classification system based on Convolutional Neural Networks (CNN), specifically the VGG16 architecture, the system can automatically analyze image frames to classify scenes as accident or non-accident with high accuracy, even on small datasets. CNNs have proven superior in image analysis tasks due to their ability to learn complex features directly from image data, requiring minimal preprocessing and yielding over 95.5% accuracy in similar domains. The proposed system will significantly reduce detection time and enable automated, real-time alerting to emergency services, improving the chances of survival for accident victims. It enhances highway safety through continuous monitoring and prompt detection, eliminating dependency on human intervention. This shift from manual to intelligent surveillance is essential for building smarter, safer cities and improving emergency response infrastructure. Thus, the integration of deep learning in traffic surveillance presents a transformative solution to a longstanding issue in public safety and emergency management

# TABLE OF CONTENT

**CONTENTS PAGENo.**

[Certificate i](#_TOC_250052)

[Declaration ii](#_TOC_250051)

[Acknowledgement iii](#_TOC_250050)

[Vision of the Institute iv](#_TOC_250049)

[Vision of the Department v](#_TOC_250048)

Program Educational Objectives vi

[Program Outcomes vii](#_TOC_250047)

[Program Specific Outcomes viii](#_TOC_250046)

[Abstract ix](#_TOC_250045)

[Table of Content x](#_TOC_250044)

[List of Figures xii](#_TOC_250043)

1. [Introduction](#_TOC_250042)

i. [Background and History………………………………………...1](#_TOC_250041)

ii. [Problem Definition……………………………………………...1](#_TOC_250040)

iii [Research Motivation 2](#_TOC_250039)

iv. [Objective](#_TOC_250038) 2

v. Applications of the project2

vi. [Module split](#_TOC_250038) 2

1. [Literature Survey 4](#_TOC_250037)
2. Existing System
   1. [Traditional System Overview](#_TOC_250036) 6
   2. [Limitation of the Traditional System](#_TOC_250032) 6
3. Proposed System
   1. [Overview](#_TOC_250031) 8
   2. Existing Algorithm………..……………………………………11
   3. [Proposed Algorithm](#_TOC_250030) 13
4. [UML Diagrams](#_TOC_250025) 15
5. System Requirements
   1. [Software Requirements 2](#_TOC_250030)1
   2. [Hardware Requirements](#_TOC_250030) 28

7. [Functional Requirements](#_TOC_250015)

* 1. [Dataset Upload………………………………………………](#_TOC_250014)30
  2. Image Processing………........................................................30
  3. [Train-Test Split……………………………………….……..](#_TOC_250008)30
  4. [Model Training……………………………………….……..](#_TOC_250007)30
  5. Evaluation Metrics…………………………………….…….31
  6. Performance Visualization……...……………………….……..31
  7. Single Image Prediction…..…………………………….……...31
  8. Logging and User Feedback……………………………..…….31
  9. Application Exit..…………………………………………..….31

1. [Source Code………………………………………………..…](#_TOC_250006)……32
2. [Results and Discussion](#_TOC_250005)
   1. [Implementation Description………………………………](#_TOC_250004)43
   2. [Dataset Description…………………………………….....](#_TOC_250003)44
   3. [Results and Description……………………………..……](#_TOC_250002)45
3. [Conclusion and Future Scope…………………………….…..](#_TOC_250001)48

[References……………………………………………….…….](#_TOC_250000)49

# LIST OF FIGURES

**FIGURES PAGE No.**

* 1. 4.1 Block Diagram………..……………………………..…….8
  2. 5.1 Class Diagram…………………………………………….16
  3. 5.2 Sequence Diagram………………………………………..16
  4. 5.3 Activity Diagram…………………………………………17
  5. 5.4 Deployment Diagram…………………………………….18
  6. 5.5 Use Case Diagram……………………………………….18
  7. 5.6 Component diagram……………………………………..19
  8. 5.7 Dataflow Diagram………………………………………20
  9. 6.1 Python URL link…………………………………………24
  10. 6.2 Click tick box…………………………………………….24
  11. 6.3 Install Python…………………………………………….25
  12. 6.4 Command Prompt………………………………………..26
  13. 6.5 Check Python version…………………………………....26
  14. 6.6 Install Packages………………………………………….26
  15. 9.1 Accident Images…………………………………………45
  16. Non Accident Images……………………………………45
  17. GUI………………………………………………………45
  18. Performance Evaluations………………………………..45
  19. Confusion matrix of CNN with VGG16………………...45
  20. Detected as no accident………………………………….47

9.7 Image Detected as accident……………………………..47

### INTRODUCTION

**1.1 Background**

About 1.3 million people die and another 25 to 65 million suffer minor injuries annually in motor accidents. Low- and middle-income or rising nations account for the bulk of traffic accident-related mortality, according a World Health Organisation (WHO) analysis on traffic accidents by country income. At 23.5 per 100,000, developing nations have a far greater road accident death rate than industrialized or high-income nations with 11.3 per 100,000 [1]. More than ninety-percent of the road Although they own just half of the world's cars, developing nations account for most traffic-related deaths. India typically has 12 to 13 fatalities in traffic accidents every hour. The truth might be far worse, though, given so many incidents go unreported. With an average of 13 deaths every hour, or more than 140,000 annually, India is on course to rank among the nation with the highest road accident mortality count. One hour following an accident, the second phase of the incident accounts for seventy-five percent of all deaths. Providing victims with quick assistance helps one to prevent this. The objective is to assist accident victims during this crucial period of need. With resources and medical attention, the third phase of an accident—which has a 15% death risk—can be avoided even when it starts days or weeks following the hit. Figure 1: comparison of population, income, and traffic accident frequency The major goal is to integrate a system that can identify an accident from video footage a camera provides. The system is built to rapidly identify an accident and thereafter alert the authorities thereby enabling aid for accident victims in need.

**1.2 Problem Definition**

Before the integration of machine learning, accident detection systems were dependent on manual video surveillance and human reporting, which often resulted in missed incidents. There was a significant delay in response due to the time taken to notice an accident and notify emergency responders. Remote roads and highways with minimal human presence made accident detection nearly impossible. Lack of real-time monitoring tools led to inefficient emergency management. These problems collectively contributed to increased fatalities and reduced chances of timely medical aid.

**1**

**1.3 Research Motivation**

The motivation behind this research stems from the growing need to reduce road accident-related fatalities through faster detection and response. Timely identification of accidents is essential to provide immediate medical assistance and save lives. Deep learning models, especially CNNs, have proven to be powerful in image classification tasks.

**1.4 Objective**

The objective of the accident detection system is to develop an AI-powered, real-time surveillance model that accurately identifies road accidents from CCTV footage. Using a Convolutional Neural Network (CNN) with a VGG16 architecture, the goal is to classify image frames as accident or non-accident with high precision. It aims to eliminate the delay caused by manual monitoring and instantly trigger alerts to emergency services. This model is designed to enhance road safety, minimize fatalities, and improve emergency response times.

**1.5 Applications**

1. Real-time accident detection from highway or urban CCTV feeds.
2. Automatic alert system to notify emergency services immediately.
3. Smart city surveillance systems for improved traffic management.
4. Integration with traffic control systems to manage congestion post-accident.
5. Road safety monitoring and data analytics for government use.
6. Enhanced emergency response planning and decision-making.
7. Implementation in autonomous vehicle systems for environmental understanding and safety monitoring.

**2**

**1.6 Module Split – Steps of CNN with VGG16 Algorithm**

1. **Upload Dataset** – Collect and prepare labeled images (accident and non-accident).
2. **Preprocessing** – Resize images to 224x224 pixels, normalize pixel values, and split into training and testing sets.
3. **Load VGG16 Model** – Use pre-trained VGG16 (without the top layer) as the base model.
4. **Add Custom Layers** – Add flatten, dense, and output layers suitable for binary classification.
5. **Compile Model** – Use binary crossentropy loss function and Adam optimizer.
6. **Train Model** – Fit the model on training data and validate using test data.
7. **Evaluate Model** – Assess model accuracy, precision, recall, and loss metrics.
8. **Test Prediction** – Use live or new CCTV frame inputs to predict accident or non-accident scenarios in real-time.

**3**

### LITERATURE SURVEY

S. Prabakar and colleagues [1] In our hectic surroundings, none of us are ready to pay close attention. People relax about mishaps. This challenge is meant to be creatively solved by a better accident detection system for victim status determination from the accident scene. Design and implementation of this system was done using mobile microcontrollers, biological smart sensors, and LabVIEW platform. Once the system detects an accident automatically, it will SMS the location and physiological data of the victims to the phone number of the emergency care center. SMS covered body temperature, heart rate, and coma recovery status. The suggested system thereby lowers accidental mortality. [2] Alwan Zainab S.; Ali Hamid M. Many people die globally from road accidents. Two great strategies to lower road mortality are an autonomous traffic accident detection system and shorter intervals between an accident and initial emergency replies. Current techniques apply automatic accident detection and notification. Though costly, complex, and incompatible with many cars, these techniques are rather effective. Improved smartphone sensors and computational capability enable phones to identify traffic accidents. Most smartphone-based accident detection systems use the G-Force value of the accelerometer sensor and the great speed of the GPS receiver to detect an accident. Many sources claim that sluggish drivers cause 90% of road accidents. This is hence incorporated in high-speed collision detection. [3], others Arif Shaik, Jennifer Bole, Gary Kunzi, Natalie Bowen IoT finds use both in the public and private sectors. IoT applications are sought for by automakers to satisfy consumer expectations, increase vehicle safety, and generate fresh products maximizing profitability. How the Internet of Things might improve communication speed and accuracy interests healthcare. The feasibility of arming a car with collision-detection sensors to notify emergency personnel is investigated in this paper. Call 911 for help following an automobile accident. Not one automated alert exists for friends, relatives, police, or ambulances. IoT can program replies and warnings. Once the accelerometer and GPS sensor send a signal to the cloud, the car subscriber will be warned. The signal will show accident degree and GPS position. The ambulance will come soon with GPS coordinates. To increase accident response times, the 2019 SOSmart automatic automobile crash detection app uses accelerometer and GPS data to locate car wrecks and immediately notify emergency contacts of their position. [5] To increase road safety and instantly identify collisions, Kongan (2008) investigates vehicle-to---vehicle (V2V) and vehicle-to---infrastructure (V2I) networks.

In intelligent transportation systems, this accelerates data flow. To increase situational awareness and traffic management, Zeng, Yuanyuan, et al. (2016)

offer an opportunistic fleet-based vehicle sensor network for real-time road incident identification. As mobile sensors, the cars in this network share traffic, accident, and road risk data. Inception-v3, a deep learning object identification and picture categorization model, is presented by Christian Szegedy and colleagues (2016). Batch normalisation, factorised convolutions, and auxiliary classifiers all help picture pattern identification. Inception-v1 (GoogLeNet), a fast image processing deep convolutional neural network, is presented by Christian Szegedy and colleagues (2015). In large-scale classification applications, 1x1 convolutions simplify processing complexity and increase accuracy, hence enhancing computer vision.

### EXISTING SYSTEM

**Traditional System Overview**

**Manual System : Human CCTV Monitoring**

In most urban areas, traffic control centers rely on human operators to monitor multiple live feeds from CCTV cameras installed across roads and highways. These operators visually observe traffic flow, detect unusual events such as collisions, and report them to emergency responders or traffic police. This method depends heavily on the attention span, skill, and availability of the personnel.It also becomes impractical to monitor hundreds of feeds simultaneously, especially in large cities where the number of cameras is high. Any delay in human reaction can result in a significant lag in dispatching help to the accident site.

**Manual System : Public Reporting via Phone or Emergency Hotlines**

Another commonly used manual method is relying on eyewitnesses or bystanders to report accidents via emergency helplines like 100 or 108 in India. The public plays a critical role in alerting authorities after witnessing an accident. Although this system works in populated areas, it is unreliable in remote locations, highways with low traffic, or during night hours when fewer people are present. Delays in calling or inaccurate location information often hinder timely emergency response.

**Manual System : Police Patrolling and Highway Surveillance**

Police or highway patrol units are deployed to monitor traffic and ensure safety. These teams are responsible for spotting accidents, helping victims, and maintaining road discipline. If an accident occurs outside their routine path or before their next visit to a particular area, it can go unnoticed for long periods. Additionally, resource limitations such as lack of vehicles, personnel, or communication tools can further delay response times. Manual logging and paperwork also slow down data processing and incident reporting.

**Limitation of the Traditional System**

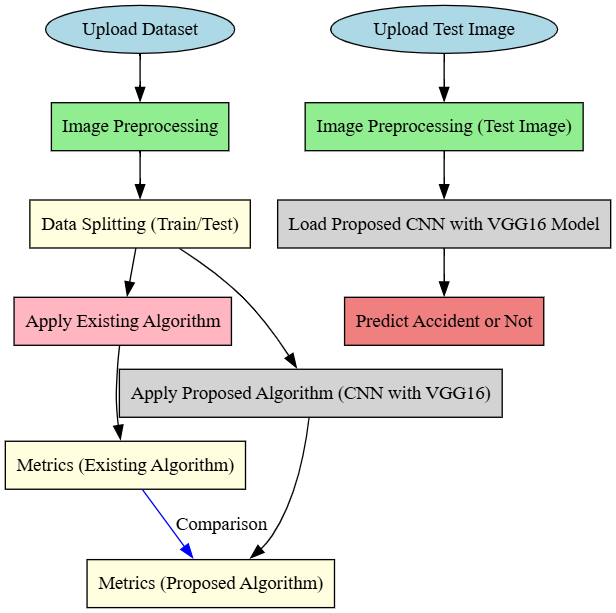
* **Human dependency**: Entire systems rely on individuals, leading to fatigue, errors, or missed detections.
* **Delayed response**: Manual identification and communication increase the time between accident and assistance.
* **Limited coverage**: Patrolling and observation are inefficient in covering large areas or during nighttime.
* **Unreliable public reporting**: Bystanders may not report accidents due to fear or indifference.
* **Lack of real-time alerts**: Manual systems lack the speed and automation required for immediate emergency action.

**4. PROPOSED SYSTEM**

**4.1 Overview**

Using a CNN with VGG16, which is intended to process images taken from traffic surveillance recordings, the accident detection model is constructed. The architecture uses VGG layers, Batch Normalization, max-pooling layers to extract spatial data while reducing dimensionality after several convolutional layers with ReLU activation. Dropout layers are used to ensure that the model generalizes properly by preventing overfitting. Using a softmax activation function, the last dense layers categorize images as either accident or non-accident. To improve the model's performance during training, data augmentation methods, batch processing, and validation on an independent dataset are used.

properly by preventing overfitting. Using a softmax activation function, the last dense layers categorize images as either accident or non-accident. To improve the model's performance during training, data augmentation methods, batch processing, and validation on an independent dataset are used.



4.1 Block Diagram

4

**Step 1: Images Pre-processing & Input Preparation**

Making sure that images are prepared correctly for the CNN with VGG16 model requires input preparation and image pre-processing as the initial stage in accident detection. Every image is taken from traffic surveillance footage and shrunk to a specific size (e.g., 128×128 pixels). In order to improve the efficiency and stability of the model, the images are then transformed into numerical arrays and normalized by scaling pixel values between 0 and 1. Techniques for data augmentation, like flipping, rotation, and contrast modifications, can be used to improve model generalization and diversify datasets. To ensure that the images are prepared for feature extraction and classification, they are then reshaped into a format that works with the deep learning model.

**Step 2: Model Selection & Loading Pre-trained Weights**

To guarantee effective accident detection, the model is chosen and pre-trained weights are loaded in the second stage. The ability of a CNN with CGG16 to recognize patterns and spatial information in traffic images is the reason it was selected. A saved JSON model file and matching pre-trained weights (.h5 file) are used to load any previously trained models in order to prevent having to start over. This method increases prediction speed and accuracy by utilizing prior training efforts. VGG convolutional layers, max-pooling, and fully connected layers make up the model architecture, which is optimized with category cross-entropy loss using the Adam optimizer.

**Step 3: Image Detection**

Using a CNN with VGG16 model that has been trained, the image detection system analyzes live images to identify accidents. A image file can be chosen by users and examined. To prepare each frame for model prediction, it is scaled to 128 x 128 pixels and reshaped. Using OpenCV's putText function, the trained classifier detects whether an accident is observed and shows the appropriate class. To notify users of an accident/.

When evaluating the effectiveness and dependability of an AI-powered accident occurrence notification system, model evaluation is a crucial step. To ascertain how well the trained model detects accidents from traffic surveillance footage, it is analyzed using important performance metrics like accuracy, precision,

recall, and F1-score. The model is usually tested on unseen images and its detection rate is compared against ground truth labels as part of the evaluation procedure.

**Detection Process:**

An object detection model based on deep learning is used in the accident detection process to examine images and spot possible collisions. Every image is captured by the algorithm, which then uses a TensorFlow detection model that has already been trained to extract important data, object classes. These components make it easier to detect cars, pedestrians, and other things in the scene. The function assesses the movement patterns and closeness of items it has identified in order to determine whether an accident has taken place. In order to ensure constant traffic scenario monitoring, the system continues processing images in real time if no accidents are identified. The detection model distinguishes between typical traffic flow and accident incidents based on object interactions, which aids in the creation of precise classifications

**4.2 Data Splitting & Preprocessing**

In this research, the dataset used for training and evaluating the AI-powered accident detection system is composed of traffic surveillance images categorized into two distinct classes: *Accident* and *No Accident*. The initial step in data handling involves **uploading the dataset**, wherein the system prompts the user to select a directory containing subfolders named according to their respective classes. Each subfolder holds images corresponding to its label. Upon selection, the system extracts and lists the available categories, ensuring a structured approach to classification.

Once the dataset is uploaded, the **image preprocessing phase** is triggered. During this stage, every image within the dataset is read using the OpenCV library and resized to a uniform dimension of **128x128 pixels with three color channels (RGB)** using the resize() function from the skimage.transform module. The primary objective of this resizing operation is to standardize the input dimensions, facilitating compatibility with convolutional neural networks (CNNs), particularly the VGG16 model. Each image is then **flattened into a one-dimensional array** to ease initial storage and later reshaped back during model training. Corresponding labels are assigned by mapping folder names to

their categorical indices. The preprocessed image arrays (X) and their labels (Y) are stored as .npy files to avoid redundant computation in future runs. Following preprocessing, the dataset is subjected to **data splitting** to segregate training

and testing samples. An 80:20 ratio is employed using train\_test\_split() from Scikit-learn, ensuring that 80% of the data is used for training while 20% is reserved for testing. This split is critical for model evaluation as it allows the system to assess its performance on previously unseen data. In the case of the CNN model, particularly VGG16, a slightly different split of 85:15 is applied post-shuffling to create training and validation sets. Furthermore, the target labels (Y) are converted into **one-hot encoded vectors** using Keras’s to\_categorical() function, which transforms categorical integers into binary class matrices — a format suitable for multi-class classification tasks. To preserve reproducibility and ensure consistent dataset ordering, the system checks for the presence of a shuffled\_indices.npy file, which holds randomly shuffled indices for the input data. If it exists, the same shuffled order is used; otherwise, new indices are generated and saved for future use. This strategy ensures that each run of the model starts with the same randomized distribution of training and testing data.

**4.3.1 Existing Algorithm: Extra Trees Classifier (ETC)**

**Definition & Information**

The Extra Trees Classifier (ETC), also known as Extremely Randomized Trees, is an ensemble machine learning method based on decision trees. It is designed to improve predictive performance by averaging the results of multiple randomized decision trees. Unlike traditional random forests, ETC increases variance reduction by selecting split points randomly for each feature, not just selecting random features. This randomness reduces overfitting and enhances generalization. It’s particularly efficient in high-dimensional datasets and is suitable for both classification and regression tasks.

**How It Works**

ETC builds multiple decision trees using the entire training dataset, but instead of optimizing split criteria at each node, it selects the split threshold randomly. For every split, a random subset of features is considered, and for each selected feature, a random split point is chosen. Among these random splits, the best one is chosen based on a scoring function like Gini index or entropy. This added

randomness makes ETC faster and less prone to overfitting compared to traditional ensembles, especially on noisy datasets.

**Algorithm Steps**

1. Input the training dataset.
2. Select K features randomly for each split.
3. For each selected feature, generate a random split value.
4. Compute impurity (e.g., Gini index) for each split.
5. Select the best among the random splits.
6. Grow the tree to its full depth without pruning.
7. Repeat the above for all trees in the ensemble.
8. Aggregate the predictions using majority voting (classification).
9. Return the final predicted class.

**Architecture (10 lines)**

* Input layer: Feature vector
* Random selection of features at each node
* Random split threshold generation
* Gini/entropy impurity computation
* Node split based on best score
* Fully grown tree (no pruning)
* Repeat tree generation for ensemble
* Voting across all trees
* Output: predicted class label
* Final classification result

**Disadvantages**  
Despite its efficiency, ETC suffers from several limitations. Its reliance on randomness can make it less interpretable than single decision trees. While ETC handles noise well, its performance might degrade with highly imbalanced

datasets. Moreover, ETC is not ideal for image data, as it lacks the spatial feature extraction capabilities of convolutional neural networks. It also requires extensive memory and processing for large datasets, limiting its scalability for deep learning-level tasks like image classification.

**4.3.2 Proposed Algorithm: CNN with VGG16**

**Definition & Information**

A Convolutional Neural Network (CNN) with VGG16 is a deep learning architecture tailored for image classification and recognition tasks. VGG16, developed by the Visual Geometry Group at Oxford, is a 16-layer CNN pre-trained on ImageNet. It employs small (3×3) convolutional filters and deep layers to extract hierarchical features from images. In this research, the VGG16 model is used as a feature extractor for classifying images into 'Accident' and 'No Accident' categories. The final layers are fine-tuned or replaced to suit the binary classification objective.

**What is CNN with VGG16?**

CNN with VGG16 is a fusion of convolutional operations for feature extraction and fully connected layers for classification. VGG16 uses 13 convolutional layers and 3 fully connected layers, with ReLU activation and max-pooling to reduce spatial dimensions. In this research, VGG16 acts as a backbone for extracting deep spatial features, followed by custom dense layers for binary classification. The pre-trained weights enable transfer learning, speeding up training and improving accuracy.

**Algorithm Steps (Architecture)**

1. Input image (128x128x3)
2. Convolutional Layer 1 + ReLU
3. Convolutional Layer 2 + ReLU
4. Max Pooling
5. Repeat Conv + ReLU + Max Pooling layers (13 total conv layers)
6. Flatten the output
7. Fully Connected Layer 1 (Dense 4096) + ReLU
8. Fully Connected Layer 2 (Dense 4096) + ReLU
9. Final Dense Layer (2 units) + Softmax
10. Output: Accident / No Accident class

**How It Works**

VGG16 begins by accepting input images of fixed size and applies multiple layers of 3x3 convolutions, each followed by ReLU activation. After every few convolutions, a max-pooling operation reduces the spatial dimensions, retaining essential features. Once the features are extracted, the final pooled feature maps are flattened into a vector and passed through fully connected dense layers to make a final prediction. During training, the weights are adjusted to minimize classification error using backpropagation.

**Advantages**  
VGG16 offers robust performance in image classification due to its depth and architectural simplicity. It provides better accuracy than traditional classifiers like ETC when handling visual data, thanks to hierarchical feature extraction. Its pre-trained model allows for transfer learning, reducing training time and increasing generalization on smaller datasets. VGG16 captures both low-level (edges, textures) and high-level features (shapes, objects), making it highly effective in detecting traffic accidents from surveillance footage. Additionally, fine-tuning only the top layers allows customization without retraining the full network, saving computation and memory

1. **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 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 researchs.

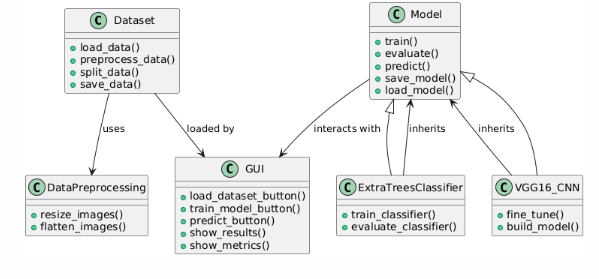
**GOALS:** The Primary goals in the design of the UML are as follows:

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**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

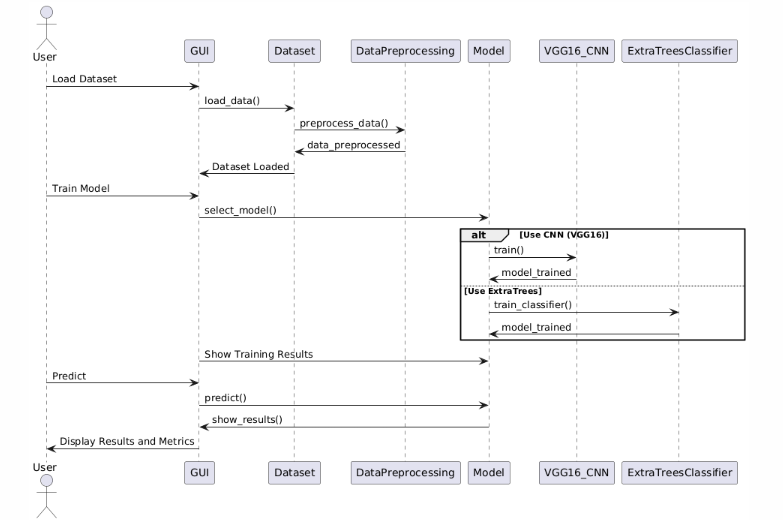
case diagram into a set of interrelated classes. The relationship or association between the classes can be either an “is-a” or “has-a” relationship. Each class in the class diagram be capable of providing certain functionalities. These functionalities provided by the class are termed “methods” of the class. Apart from this, each class have certain “attributes” that uniquely identify the class



5.1 Class Diagram

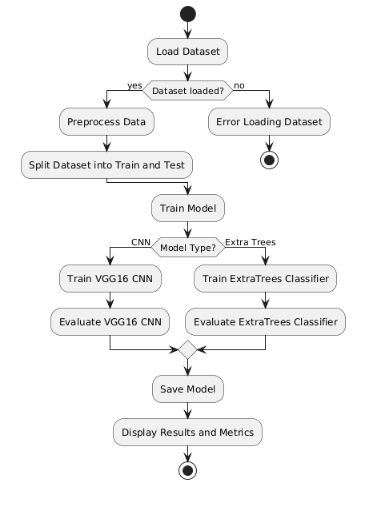
**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



5.2 Sequence Diagram

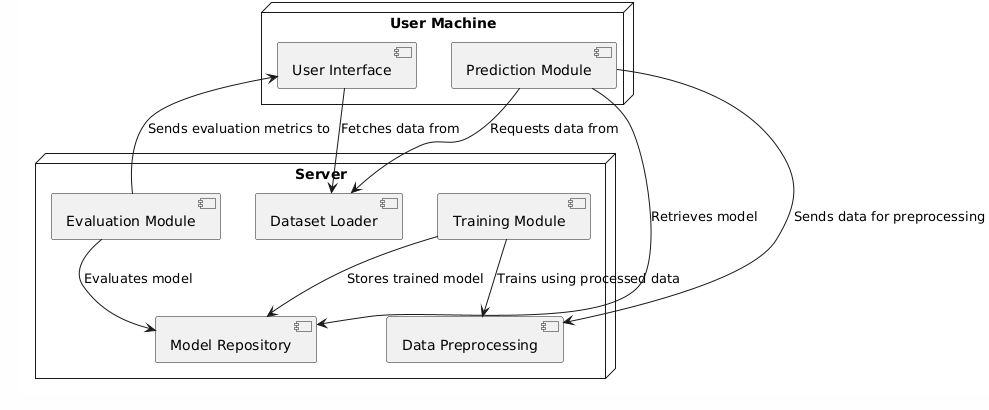
**Activity diagram:** Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.



5.3 Activity Diagram

**Deployment diagram**

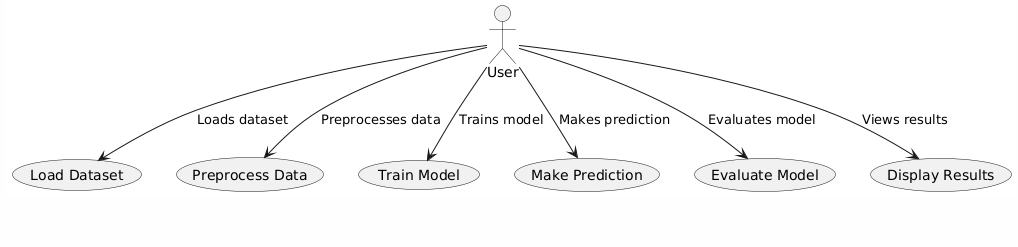
A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components (“nodes”) exist (e.g., a web server, an application server, and a database server), what software components (“artifacts”) run on each node (e.g., web application, database), and how the different pieces are connected (e.g., JDBC, REST, RMI). The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes have sub nodes, which appear as nested boxes. A single node in a deployment diagram conceptually represent multiple physical nodes, such as a cluster of database servers.



5.4 Deployment Diagram

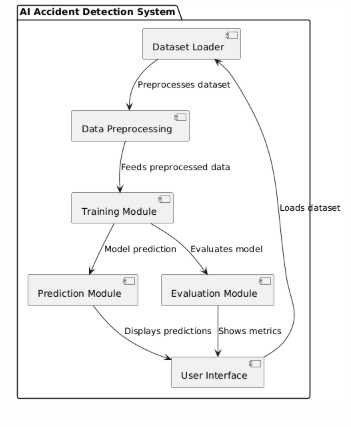
**Use case diagram**

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.



5.5 Use Case Diagram

**Component diagram:** Component diagram describes the organization and wiring of the physical components in a system.



5.6 Component diagram

**Data Flow Diagram (DFD):**

In UML (Unified Modeling Language) is a graphical representation of the flow of data through a system. It's particularly useful for understanding the data inputs, outputs, processes, and storage within a system or software application. Here's

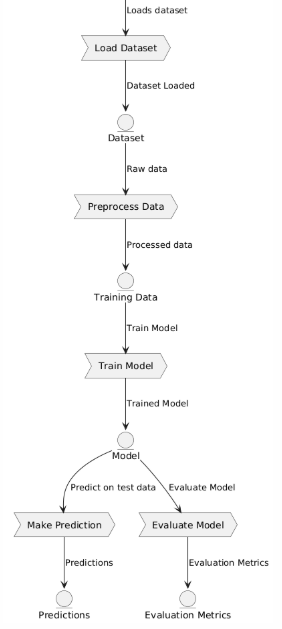
External Entities: These are entities outside the system that interact with it, such as users, other systems, or databases. They are represented as rectangles on the edges of the diagram.

Processes: Processes represent the actions or transformations that occur within the system. They are depicted as circles or ovals and typically have labels describing the action they perform on the data.

Data Stores: Data stores represent where data is stored within the system. They can be databases, files, or any other storage medium. Data stores are typically represented as rectangles.

Data Flows: Data flows represent the movement of data between external entities, processes, and data stores. They are depicted as arrows and indicate the direction of data flow.

Data Transformations: These represent the conversion or manipulation of data within a process. They can be depicted using labels on the data flow arrows or as separate process symbols.



5.7 Dataflow Diagram

**6. SYSTEM REQUIREMENTS**

**6.1 Software Requirements**

**Python 3.7.6**

Python 3.7.6 serves as a pivotal version for developers and researchers due to its robust features, backward compatibility, and widespread support across a variety of libraries and frameworks. Released during a time when machine learning and data science tools were rapidly evolving, Python 3.7.6 provided a stable and consistent platform. This version includes critical improvements like enhanced asyncio functionality for asynchronous programming, increased precision for floating-point numbers, and optimized data structures. It became the go-to version for compatibility with popular libraries like TensorFlow 2.0, PyTorch, and Pandas, ensuring seamless integration and efficient execution for both academic and industrial applications.

Compared to older Python versions, 3.7.6 introduced several features such as dataclasses, which simplified boilerplate code for object-oriented programming. The improved async and await syntax made concurrent programming more intuitive, while changes to the standard library enhanced usability and performance. Over newer versions, Python 3.7.6 remains a preferred choice for legacy systems and researchs requiring compatibility with libraries that may not yet support the latest Python updates. Its combination of stability and maturity ensures that it is reliable for long-term researchs, especsially in environments where upgrading the Python interpreter might disrupt existing workflows.

**TensorFlow Environment**

TensorFlow provides a comprehensive ecosystem for building, training, and deploying machine learning models. Its support for numerical computation and deep learning applications makes it a staple in AI research and development. By offering a flexible architecture, TensorFlow enables deployment across a variety of platforms, including desktops, mobile devices, and the cloud. The ability to scale across CPUs, GPUs, and TPUs ensures that TensorFlow is suitable for both small experiments and large-scale production systems.

TensorFlow’s transition from older versions, like 1.x, to 2.x brought significant improvements in ease of use, including the introduction of the tf.keras API for building models, eager execution for dynamic computation, and enhanced debugging capabilities. Compared to newer frameworks, TensorFlow retains a

strong advantage due to its mature community support, extensive documentation, and integration with TensorFlow Extended (TFX) for managing production pipelines. Its compatibility with other libraries and tools, such as Keras and TensorBoard, makes it a robust choice for end-to-end machine learning solutions.

**Packages Overview**

**Keras:** Older versions of Keras required extensive configuration for custom model creation. Version 2.3.1 unified the APIs with TensorFlow integration, reducing overhead and enabling direct use of TensorFlow backends, ensuring faster execution and easier debugging.While newer versions focus on performance and distributed training, version 2.3.1 is lightweight and stable, making it ideal for smaller researchs without the complexity introduced in later iterations, which are more suited for advanced workflows.

**NumPy:** Version 1.19.5 introduced critical bug fixes and performance enhancements over older versions, especially for operations involving large datasets. The improved random number generator and better handling of exceptions provide more reliable results for numerical computations.This version remains compatible with a wide range of dependent packages. While newer versions optimize speed further, 1.19.5 balances stability and compatibility, ensuring fewer compatibility issues with older software stacks.

**Pandas:** Version 0.25.3 brought significant speed improvements for large-scale data processing, particularly in operations like groupby. Enhancements in handling missing data and improved compatibility with external libraries made this version more robust for data analysis tasks.While newer versions does not add features like enhanced type checking, 0.25.3 remains lightweight and stable for researchs that do not require cutting-edge functionalities, making it a practical choice for legacy systems.

**Imbalanced-learn:** Version 0.7.0 introduced optimized algorithms for handling class imbalances, such as improved SMOTE implementations. This update also enhanced the ease of integrating with scikit-learn pipelines.While newer versions do not contain experimental features, 0.7.0 is reliable and well-documented, ensuring robust performance in addressing data imbalance issues without unnecessary complexity.

**Scikit-learn:** Version 0.23.1 included improved support for cross-validation and hyperparameter optimization. Updates to RandomForestClassifier and

GradientBoostingClassifier increased model accuracy and efficiency.0.23.1 is widely tested and compatible with older hardware, making it a dependable choice for environments where the latest versions may introduce compatibility issues.

**Imutils:** This package provides an easy-to-use interface for image processing tasks. Its functions for resizing, rotating, and translating images simplified workflows compared to writing custom code.Its lightweight nature and stable functionality make it suitable for researchs not requiring cutting-edge image manipulation techniques, balancing simplicity and capability.

**Matplotlib:** Version 3.x improved plot interactivity and introduced better 3D plotting capabilities. The tight\_layout function and compatibility with modern libraries streamlined visualization workflows.The earlier versions maintain stability and compatibility with older datasets and software, avoiding potential issues from newer, untested updates.

**Seaborn:** Improved APIs in newer versions simplified aesthetic customization of plots. The addition of new themes and color palettes in 0.11.x enhanced visual appeal for exploratory data analysis.Older versions remain computationally less demanding, suitable for lightweight applications without requiring extensive customizations.

**OpenCV-Python:** Recent updates enhanced compatibility with deep learning frameworks and accelerated image processing pipelines, especially for real-time applications.Older versions are stable and resource-efficient, making them ideal for systems with limited computational capacity or for legacy applications.

**H5Py:** Version 2.10.0 improved file handling efficiency for large datasets. It introduced better support for advanced indexing, which is crucial for working with high-dimensional data.While newer versions support more advanced features, 2.10.0 ensures compatibility with older machine learning frameworks and models.

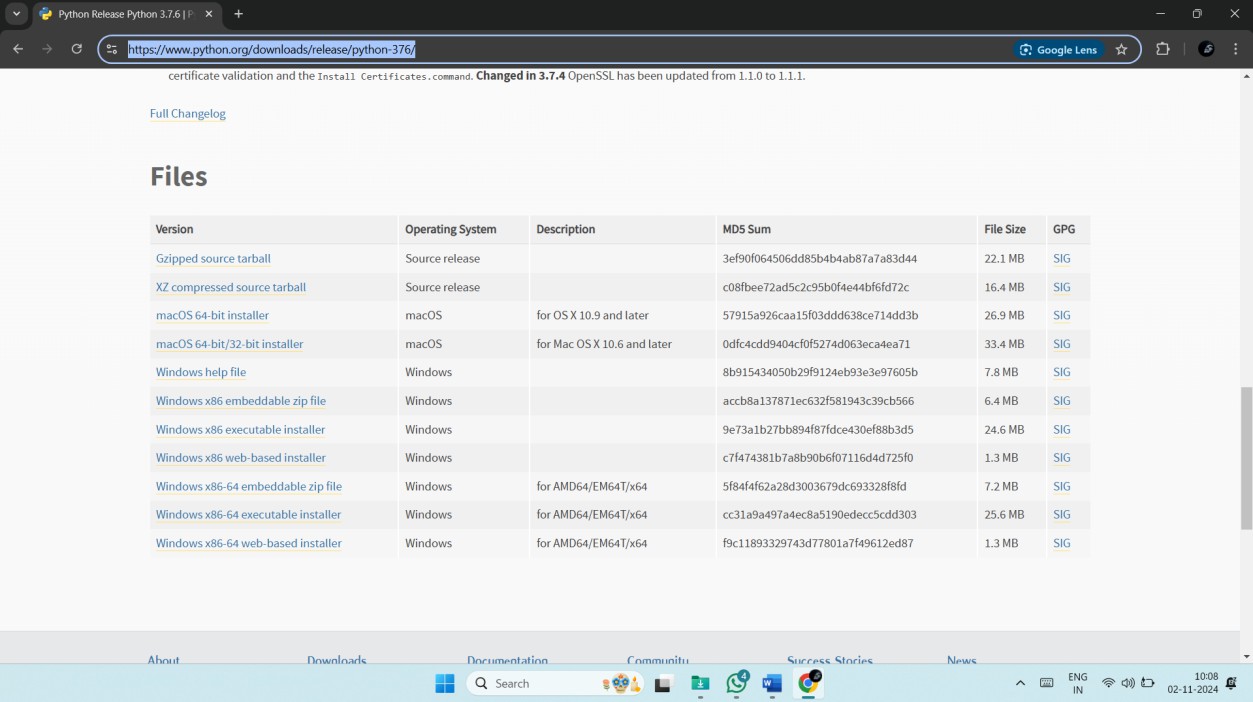
**Jupyter:** Jupyter improved the interactivity and scalability of notebooks for collaborative coding and visualization tasks. Integration with tools like Matplotlib made it a preferred environment for data exploration.Earlier versions are stable and lightweight, avoiding potential issues with dependencies introduced in newer releases.

**PYTHON INSTALLATION**

**INSTALLATION PROCESS OF PYTHON**

STEP 1: INSTALL PYTHON BY PRESSING THE PYTHON URL LINK.

<https://www.python.org/downloads/release/python-376/>

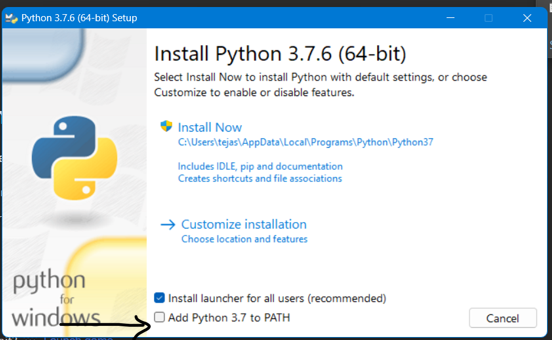


6.1 Python URL link

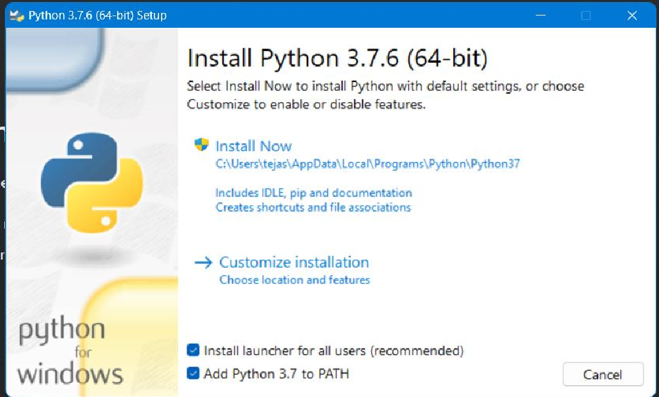
Then scroll down there you can see the files. After that click arrow located link to download.

STEP 2: The python with our required 3.7.6 version will download.

STEP 3: Click the below **tick box** to install your python in your path.



6.2.1 Click tick box



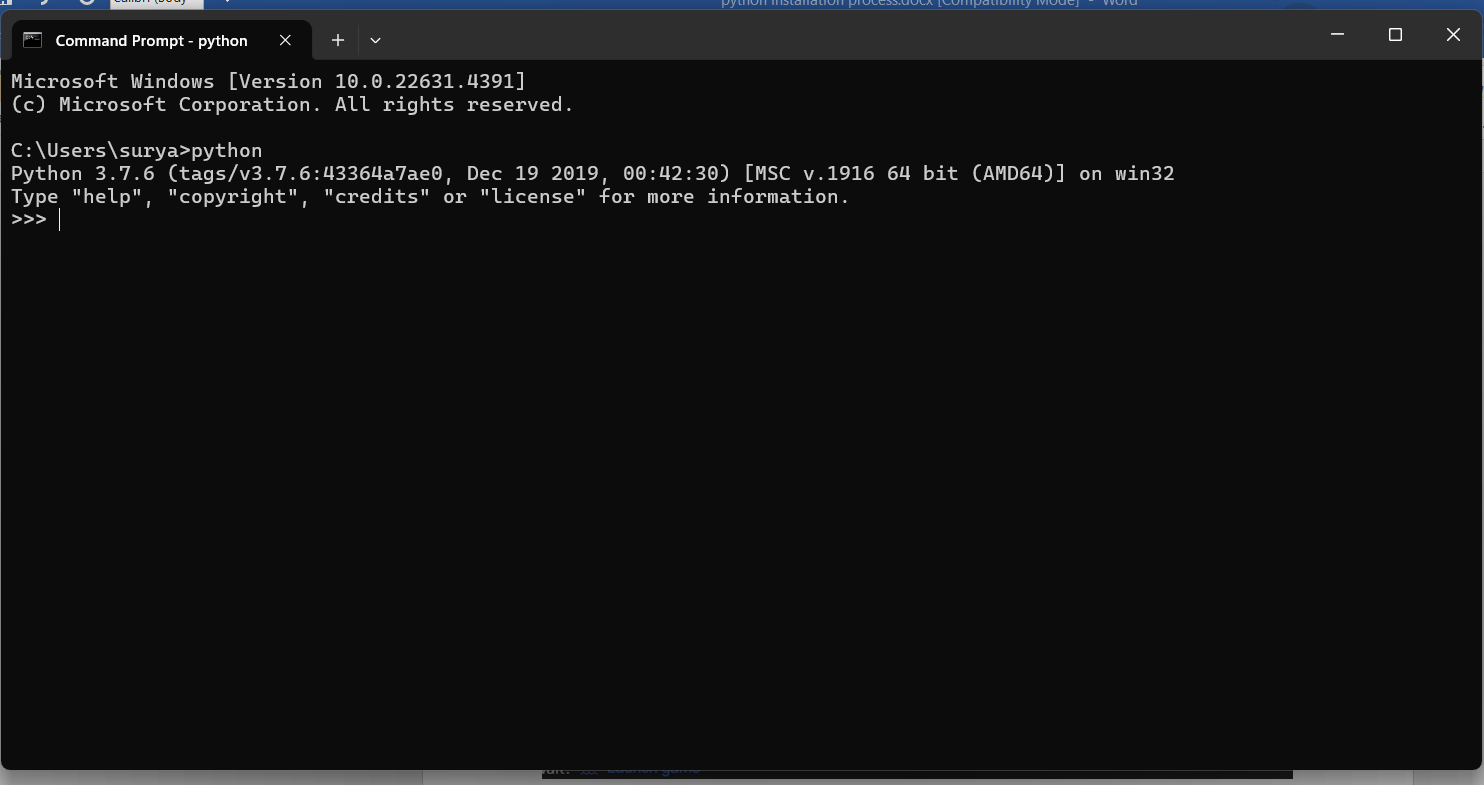
6.2.2 Click tick box

STEP 4: Then enter **install now**.

6.3 Install Python

After completion of installation.

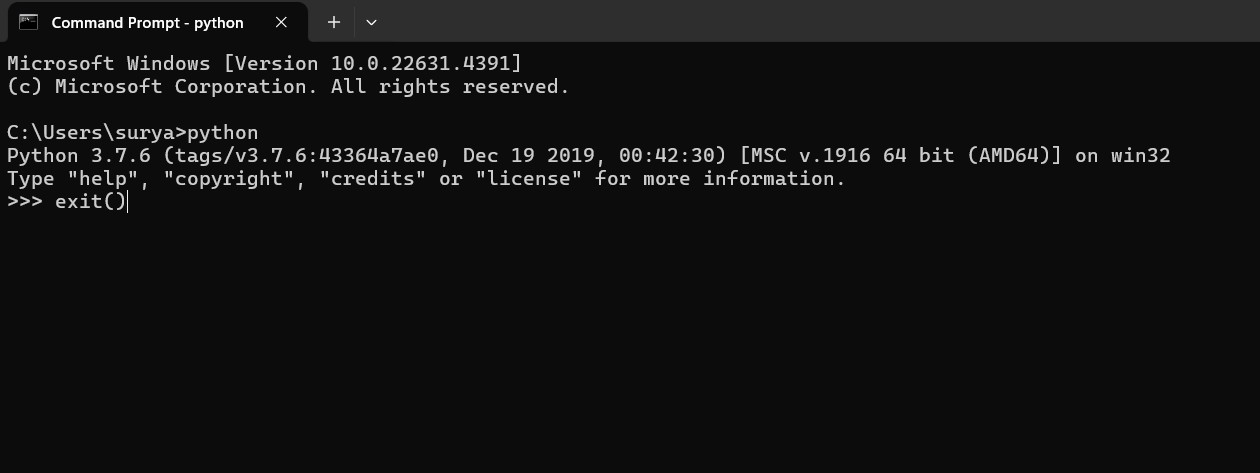
Step 5: open Command Prompt and enter python.



6.4 Command Prompt

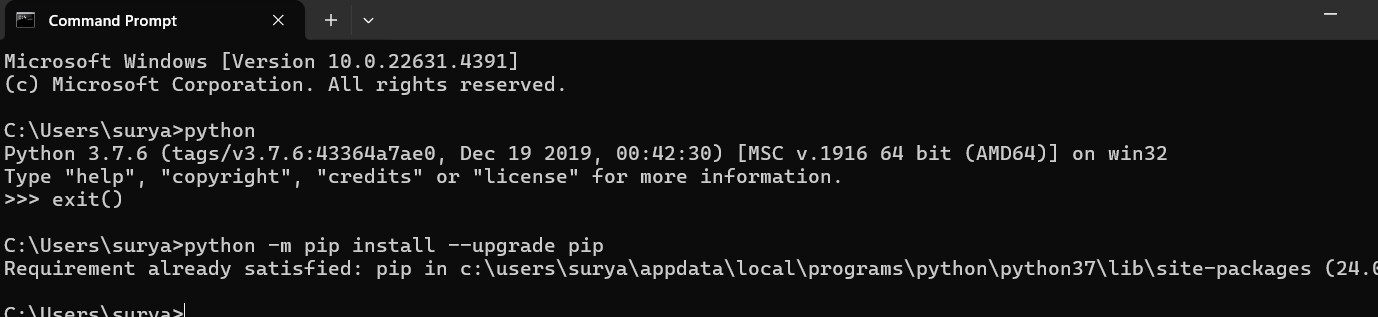
If it shows the python version as python 3.7.6. means it is perfectly installed.

Step 6: enter exit. By entering **exit().**



6.5 Check Python version

Step: install below packages.



6.6 Install Packages

**install the below packages.** python -m pip install --upgrade pip pip install tensorflow==1.14.0

pip install keras==2.3.1 pip install pandas==1.3.5

pip install scikit-learn==1.0.2 pip install imutils

pip install matplotlib==3.2.2 pip install seaborn==0.12.2

pip install opencv-python== 4.1.1.26 pip install h5py==2.10.0

pip install numpy==1.19.2

pip install imbalanced-learn==0.7.0 pip install jupyter

pip install protobuf==3.20.\* pip install scikit-image==0.16.2

**Complete Explanation of installation**

To install Python 3.7.6, first, visit the official Python website at [Python Downloads](https://www.python.org/downloads/release/python-376/). Scroll down to the "Files" section and select the appropriate installer based on your operating system. For Windows users, choose either the "Windows x86-64 executable installer" for 64-bit or the "Windows x86 executable installer" for 32-bit. macOS users should download the macOS 64-bit installer. Linux users can install Python using their respective package managers such as apt, dnf, or yum.

After downloading, run the installer and check the box that says **"Add Python to PATH"** before proceeding with the installation. Click on **"Install Now"** and wait for the process to complete. Once installed, open **Command Prompt** and type python --version to verify that Python 3.7.6 has been successfully installed. If the version appears correctly, it means the installation was successful. You can exit Python by typing exit().

Once Python is installed, the next step is to upgrade pip, the package installer for Python. Open **Command Prompt** and run the command python -m pip install --upgrade pip. This ensures that you have the latest version of pip, which is required for installing other dependencies.

Now, install the necessary machine learning and deep learning libraries. Begin by installing TensorFlow and Keras using the commands pip install tensorflow==1.14.0 and pip install keras==2.3.1. These libraries are crucial for building and training deep learning models. Additionally, install Pandas (pip install pandas==1.3.5), NumPy (pip install numpy==1.19.2), and Scikit-learn (pip install scikit-learn==1.0.2) to handle data processing and machine learning tasks.

For image processing tasks, install OpenCV (pip install opencv-python==4.1.1.26), scikit-image (pip install scikit-image==0.16.2), and imutils (pip install imutils). These libraries help with tasks such as image transformations, object detection, and feature extraction. Additionally, visualization libraries like Matplotlib (pip install matplotlib==3.2.2) and

Seaborn (pip install seaborn==0.12.2) will be useful for plotting data and analyzing results.

Other essential packages include H5py (pip install h5py==2.10.0) for handling HDF5 file formats, imbalanced-learn (pip install imbalanced-learn==0.7.0) for dealing with imbalanced datasets, and Jupyter Notebook (pip install jupyter) for interactive coding and visualization. Also, install Protobuf (pip install protobuf==3.20.\*), which is required for serializing structured data in machine learning applications.

To ensure that all packages are installed correctly, open **Python** in **Command Prompt** by typing python, then try importing the installed libraries using import tensorflow as tf, import keras, import pandas as pd, import numpy as np, import matplotlib.pyplot as plt, import seaborn as sns, import cv2, import imutils, and from sklearn.model\_selection import train\_test\_split. If there are no errors, the installation has been completed successfully.

For users who prefer using Jupyter Notebook, launch it by typing jupyter notebook in **Command Prompt**. This will open a browser interface where you can run Python code interactively. With all these steps completed, your Python environment is now fully set up for machine learning and deep learning researchs.

**6.2 Hardware Requirements**

Python 3.7.6 can run efficiently on most modern systems with minimal hardware requirements. However, meeting the recommended specifications ensures better performance, especially for developers handling large-scale applications or computationally intensive tasks. By ensuring compatibility with hardware and operating system, can leverage the full potential of Python 3.7.6.

**Processor (CPU) Requirements:** Python 3.7.6 is a lightweight programming language that can run on various processors, making it highly versatile. However, for optimal performance, the following processor specifications are recommended:

* **Minimum Requirement**: 1 GHz single-core processor.
* **Recommended**: Dual-core or quad-core processors with a clock speed of 2 GHz or higher. Using a multi-core processor allows Python

applications, particularly those involving multithreading or multiprocessing, to execute more efficiently.

**Memory (RAM) Requirements:** Python 3.7.6 does not demand excessive memory but requires adequate RAM for smooth performance, particularly for running resource-intensive applications such as data processing, machine learning, or web development.

* **Minimum Requirement**: 512 MB of RAM.
* **Recommended**: 4 GB or higher for general usage. For data-intensive operations, 8 GB or more is advisable.

Insufficient RAM can cause delays or crashes when handling large datasets or executing computationally heavy programs.

**Storage Requirements:** Python 3.7.6 itself does not occupy significant disk space, but additional storage required for Python libraries, modules, and researchs.

* **Minimum Requirement**: 200 MB of free disk space for installation.
* **Recommended**: At least 1 GB of free disk space to accommodate libraries and dependencies.
* Developers using Python for large-scale researchs or data science should allocate more storage to manage virtual environments, datasets, and frameworks like TensorFlow or PyTorch.

**Compatibility with Operating Systems:** Python 3.7.6 is compatible with most operating systems but requires hardware that supports the respective OS. Below are general requirements for supported operating systems:

* **Windows**: 32-bit and 64-bit systems, Windows 7 or later.
* **macOS**: macOS 10.9 or later.
* **Linux**: Supports a wide range of distributions, including Ubuntu, CentOS, and Fedora.

**7.** **FUNCTIONAL REQUIREMENTS**

1. **Dataset Upload:** The application must provide a button that opens a directory‑selection dialog, allowing the user to point to a root folder containing subfolders for each class (“Accident” and “no Accident”). Once the user selects the folder, the system shall scan its subdirectories, identify class labels, and display a confirmation message along with the list of detected classes in the on‑screen log area.
2. **Image Preprocessing:** Upon invoking the preprocessing function, the system shall traverse the chosen dataset directory, read each image file, resize it to a fixed dimension of 128×128×3, flatten it into a one‑dimensional array, and append it to an input array (X). Simultaneously, it will record the corresponding class index in an output array (Y). To optimize future runs, these arrays must be saved as .npy files and loaded directly if they already exist.
3. **Train–Test Split:** The application shall split the preprocessed data into training and testing subsets according to a predefined ratio (e.g., 80/20 ) It must use a fixed random seed to ensure reproducible splits and then report the shapes of the resulting training and testing sets in the log window.
4. **Model Training – Extra Trees Classifier:** When the user opts to train the traditional machine‑learning model, the system shall check for an existing ExtraTreesClassifier saved on disk. If found, it loads and reuses it; if not, it trains a new Extra Trees model with specified hyperparameters (e.g., 10 trees, max depth 3) on the training set, saves the trained model, and then proceeds to make predictions on the test set.
5. **Model Training – CNN with VGG16:** For deep‑learning training, the application must first shuffle and optionally cache shuffled indices to ensure data randomness. It then reshapes and normalizes the data, one‑hot encodes the labels, and either loads a previously saved VGG16‑based model and its weights or builds a new model by:
   * Loading the ImageNet‑pretrained VGG16 base (without its top layers) and freezing its weights.
   * Adding a sequence of custom layers: flatten, dense (512 units), batch normalization, dropout, dense (256 units), batch normalization, dropout, and a final softmax classification layer.
   * Compiling with the Adam optimizer and categorical crossentropy loss.
   * Training for a fixed number of epochs with a ReduceLROnPlateau scheduler.
   * Saving the model architecture, weights, and training history for future reuse.
6. **Evaluation Metrics:** After any model generates predictions on the test set, the system shall calculate and display in the log area: accuracy, precision, recall, F1‑score (macro‑averaged), sensitivity, specificity, and the full classification report. It must also render and display a confusion‑matrix heatmap using seaborn and matplotlib.
7. **Performance Visualization:** The application shall offer a “Graph” button that, when clicked, produces two separate plots: (1) training vs. validation accuracy over each epoch, and (2) training vs. validation loss over each epoch. These plots must be displayed in a standalone window.
8. **Single‑Image Prediction:** The user can select an individual test image via a file dialog. The system will resize and normalize the image, feed it through the trained CNN model, determine the class label, and open a window displaying the image with overlaid text indicating “Classified as: Accident” or “Classified as: no Accident.”
9. **Logging and User Feedback:** All key operations—including dataset loading, preprocessing completion, split summaries, training progress, metric outputs, and error messages—must be appended in real time to a scrollable text widget within the GUI to keep the user informed.
10. **Application Exit:** An “Exit” button must be provided that, when pressed, cleanly destroys the main window and terminates the application without errors.

**8. SOURCE CODE**

The

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import ttk

from tkinter import filedialog

from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.callbacks import EarlyStopping

import seaborn as sns

import os

import cv2

import joblib

import pickle

import warnings

warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import xgboost as xgb

from PIL import Image

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,classification\_report,confusion\_matrix

from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

from keras.applications import VGG16

from keras.utils.np\_utils import to\_categorical

from keras.models import Sequential

from keras.layers.core import Dense,Activation,Dropout, Flatten

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from keras.models import model\_from\_json

from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, InputLayer, BatchNormalization, Dropout

from keras.callbacks import ModelCheckpoint

from skimage.transform import resize

from skimage.io import imread

from skimage import io, transform

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import LogisticRegression

main = tkinter.Tk()

main.title('AI-POWERED ACCIDENT DETECTION: TRAFFIC SURVEILLANCE IMAGES USING CNN WITH VGG16')

screen\_width = main.winfo\_screenwidth()

screen\_height = main.winfo\_screenheight()

main.geometry(f"{screen\_width}x{screen\_height}")

global filename

global X, Y

global model

global categories,model\_folder

model\_folder = "model"

label = ['Accident','no Accident']

def uploadDataset():

    global filename,categories,categories,path

    text.delete('1.0', END)

    filename = filedialog.askdirectory(initialdir=".")

    categories = [d for d in os.listdir(filename) if os.path.isdir(os.path.join(filename, d))]

    text.insert(END,'Dataset loaded\n')

    text.insert(END,"Classes found in dataset: "+str(categories)+"\n")

def imageProcessing():

    text.delete('1.0', END)

    global X, Y, model\_folder, filename,X\_file,Y\_file

    path = r"Dataset"

    model\_folder = "model"

    categories = [d for d in os.listdir(path) if os.path.isdir(os.path.join(path, d))]

    X\_file = os.path.join(model\_folder, "X.txt.npy")

    Y\_file = os.path.join(model\_folder, "Y.txt.npy")

    if os.path.exists(X\_file) and os.path.exists(Y\_file):

        X = np.load(X\_file)

        Y = np.load(Y\_file)

        print("X and Y arrays loaded successfully.")

    else:

        X = [] # input array

        Y = [] # output array

        for root, dirs, directory in os.walk(path):

            for j in range(len(directory)):

                name = os.path.basename(root)

                print(f'Loading category: {dirs}')

                print(name+" "+root+"/"+directory[j])

                if 'Thumbs.db' not in directory[j]:

                    img\_array = cv2.imread(root+"/"+directory[j])

                    img\_resized = resize(img\_array, (128, 128, 3))

                    # Append the input image array to X

                    X.append(img\_resized.flatten())

                    # Append the index of the category in categories list to Y

                    Y.append(categories.index(name))

        X = np.array(X)

        Y = np.array(Y)

        np.save(X\_file, X)

        np.save(Y\_file, Y)

    text.insert(END, "Dataset Preprocessing completed\n")

def Train\_Test\_split():

    global X,Y,x\_train,x\_test,y\_train,y\_test

x\_train,x\_test,y\_train,y\_test=train\_test\_split(X,Y,test\_size=0.20,random\_state=42)

    text.insert(END,"Total samples found in training dataset: "+str(x\_train.shape)+"\n")

    text.insert(END,"Total samples found in testing dataset: "+str(x\_test.shape)+"\n")

def calculateMetrics(algorithm, predict, y\_test):

    global categories

    a = accuracy\_score(y\_test,predict)\*100

    p = precision\_score(y\_test, predict,average='macro') \* 100

    r = recall\_score(y\_test, predict,average='macro') \* 100

    f = f1\_score(y\_test, predict,average='macro') \* 100

    text.insert(END,algorithm+" Accuracy  :  "+str(a)+"\n")

    text.insert(END,algorithm+" Precision : "+str(p)+"\n")

    text.insert(END,algorithm+" Recall    : "+str(r)+"\n")

    text.insert(END,algorithm+" FScore    : "+str(f)+"\n")

    conf\_matrix = confusion\_matrix(y\_test, predict)

    total = sum(sum(conf\_matrix))

    se = conf\_matrix[0,0]/(conf\_matrix[0,0]+conf\_matrix[0,1])

    se = se\* 100

    text.insert(END,algorithm+' Sensitivity : '+str(se)+"\n")

    sp = conf\_matrix[1,1]/(conf\_matrix[1,0]+conf\_matrix[1,1])

    sp = sp\* 100

    text.insert(END,algorithm+' Specificity : '+str(sp)+"\n\n")

    CR = classification\_report(y\_test, predict,target\_names=categories)

    text.insert(END,algorithm+' Classification Report \n')

    text.insert(END,algorithm+ str(CR) +"\n\n")

    plt.figure(figsize =(6, 6))

    ax = sns.heatmap(conf\_matrix, xticklabels = categories, yticklabels = categories, annot = True, cmap="Pastel1" ,fmt ="g");

    ax.set\_ylim([0,len(categories)])

    plt.title(algorithm+" Confusion matrix")

    plt.ylabel('True class')

    plt.xlabel('Predicted class')

    plt.show()

def Existing\_ETC():

    global x\_train,x\_test,y\_train,y\_test,model\_folder

    text.delete('1.0', END)

    model\_filename = os.path.join(model\_folder, "ETC\_model.pkl")

    if os.path.exists(model\_filename):

        mlmodel = joblib.load(model\_filename)

    else:

        mlmodel = ExtraTreesClassifier(n\_estimators=10,max\_depth=3)

        mlmodel.fit(x\_train, y\_train)

        joblib.dump(mlmodel, model\_filename)

        print(f'Model saved to {model\_filename}')

    y\_pred = mlmodel.predict(x\_test)

    calculateMetrics("Existing Extra Trees Classifier", y\_pred, y\_test)

def cnnModel():

    global X, Y, x\_train, x\_test, y\_train, y\_test, model\_folder, categories, model, history, base\_model,lr\_scheduler,opt

    text.delete('1.0', END)

    indices\_file = os.path.join(model\_folder, "shuffled\_indices.npy")

    if os.path.exists(indices\_file):

        indices = np.load(indices\_file)

        X = X[indices]

        Y = Y[indices]

    else:

        indices = np.arange(X.shape[0])

        np.random.shuffle(indices)

        np.save(indices\_file, indices)

        X = X[indices]

        Y = Y[indices]

    x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.15, random\_state=42)

    x\_train = x\_train.reshape((-1, 128, 128, 3))  # VGG16 requires 224x224 input size

    x\_test = x\_test.reshape((-1, 128, 128, 3))

    y\_train = to\_categorical(y\_train, num\_classes=len(categories))

    y\_test = to\_categorical(y\_test, num\_classes=len(categories))

    Model\_file = os.path.join(model\_folder, "VGG16\_model.json")

    Model\_weights = os.path.join(model\_folder, "VGG16\_model\_weights.h5")

    Model\_history = os.path.join(model\_folder, "history.pckl")

    num\_classes = len(categories)

    if os.path.exists(Model\_file):

        with open(Model\_file, "r") as json\_file:

            loaded\_model\_json = json\_file.read()

            model = model\_from\_json(loaded\_model\_json)

        json\_file.close()

        model.load\_weights(Model\_weights)

        model.\_make\_predict\_function()

        print(model.summary())

        with open(Model\_history, 'rb') as f:

            history = pickle.load(f)

            acc = history['accuracy']

    else:

        base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(128, 128, 3))

        lr\_scheduler = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, verbose=1)

        for layer in base\_model.layers:

            layer.trainable = False  # Freeze pretrained layers

        model = Sequential([

            base\_model,

            Flatten(),

            Dense(512, activation='relu'),

            BatchNormalization(),

            Dropout(0.4),

            Dense(256, activation='relu'),

            BatchNormalization(),

            Dropout(0.2),

            Dense(num\_classes, activation='softmax')

        ])

        model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

        print(model.summary())

        hist = model.fit(x\_train, y\_train, batch\_size=16, epochs=20, validation\_data=(x\_test, y\_test), shuffle=True, verbose=2,callbacks=[lr\_scheduler])

        model.save\_weights(Model\_weights)

        model\_json = model.to\_json()

        with open(Model\_file, "w") as json\_file:

            json\_file.write(model\_json)

        json\_file.close()

        with open(Model\_history, 'wb') as f:

            pickle.dump(hist.history, f)

        with open(Model\_history, 'rb') as f:

            accuracy = pickle.load(f)

            acc = accuracy['accuracy']

            acc = acc[9] \* 100

            print("VGG16 Model Prediction Accuracy = "+str(acc))

    Y\_pred = model.predict(x\_test)

    Y\_pred\_classes = np.argmax(Y\_pred, axis=1)

    y\_test = np.argmax(y\_test, axis=1)

    calculateMetrics("VGG16 Model", Y\_pred\_classes, y\_test)

def graph():

    global history

    fig, axs = plt.subplots(2, 1, figsize=(10, 10))

    # Plot training & validation accuracy

    axs[0].plot(history['accuracy'])

    axs[0].plot(history['val\_accuracy'])

    axs[0].set\_title('Model Accuracy')

    axs[0].set\_ylabel('Accuracy')

    axs[0].set\_xlabel('Epoch')

    axs[0].legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss

    axs[1].plot(history['loss'])

    axs[1].plot(history['val\_loss'])

    axs[1].set\_title('Model Loss')

    axs[1].set\_ylabel('Loss')

    axs[1].set\_xlabel('Epoch')

    axs[1].legend(['Train', 'Validation'], loc='upper left')

    plt.tight\_layout()

    plt.show()

def predict():

    global model

    categories= ['Accident','no Accident']

    filename = filedialog.askopenfilename(initialdir="testImages")

    img = cv2.imread(filename)

    img = cv2.resize(img, (128,128))

    im2arr = np.array(img)

    im2arr = im2arr.reshape(-1,128,128,3)

    test = np.asarray(im2arr)

    test = test.astype('float32')

    test = test/255

    X\_test\_features = model.predict(test)

    predict = np.argmax(X\_test\_features)

    img = cv2.imread(filename)

    img = cv2.resize(img, (500,500))

    cv2.putText(img, 'Classified as : '+categories[predict], (10, 25),  cv2.FONT\_HERSHEY\_SIMPLEX,0.7, (0, 255, 255), 2)

    cv2.imshow('Classified as : '+categories[predict], img)

    cv2.waitKey(0)

def close():

    main.destroy()

font = ('times', 16, 'bold')

title = Label(main, text='AI-POWERED ACCIDENT DETECTION: TRAFFIC SURVEILLANCE IMAGES USING CNN WITH VGG16')

title.config(bg='SlateBlue1', fg='black')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 13, 'bold')

ff = ('times', 12, 'bold')

uploadButton = Button(main, text="Upload Dataset", command=uploadDataset)

uploadButton.place(x=20,y=100)

uploadButton.config(font=ff)

processButton = Button(main, text="Image Processing", command=imageProcessing)

processButton.place(x=20,y=150)

processButton.config(font=ff)

mlpButton = Button(main, text="Dataset Splitting", command=Train\_Test\_split)

mlpButton.place(x=20,y=200)

mlpButton.config(font=ff)

mlpButton = Button(main, text="Train Extra Trees Classifier", command=Existing\_ETC)

mlpButton.place(x=20,y=250)

mlpButton.config(font=ff)

modelButton = Button(main, text="Train VGG16 Model", command=cnnModel)

modelButton.place(x=20,y=300)

modelButton.config(font=ff)

graphButton = Button(main, text="Accuracy & Loss", command=graph)

graphButton.place(x=20,y=350)

graphButton.config(font=ff)

predictButton = Button(main, text="Prediction from Test Image", command=predict)

predictButton.place(x=20,y=400)

predictButton.config(font=ff)

exitButton = Button(main, text="Exit", command=close)

exitButton.place(x=20,y=450)

exitButton.config(font=ff)

font1 = ('times', 12, 'bold')

text=Text(main,height=30,width=85)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=450,y=100)

text.config(font=font1)

main.config(bg = 'azure')

main.mainloop()

**9. RESULTS AND DISCUSSION**

**9.1 Implementation and Description**

**Key Functionalities:**

1. **Dataset Upload & Preprocessing**:
   * The system allows users to select a dataset organized into folders for each class (e.g., "Accident" and "No Accident").
   * Images are loaded, resized to 128x128 pixels, flattened, and converted into numerical arrays for model training.
   * The data is saved for reuse, so preprocessing doesn't need to be repeated every time.
2. **Train-Test Split**:
   * The dataset is split into training and testing sets to evaluate the model's performance objectively.
3. **Model Training**:
   * Two types of models are supported:
     + **Extra Trees Classifier (ETC)**: A traditional machine learning ensemble method.
     + **VGG16-based CNN**: A deep learning model using pretrained VGG16 (without the top layers), fine-tuned with new layers for binary classification.
   * The CNN uses callbacks like ReduceLROnPlateau to adapt learning rate and avoid overfitting.
4. **Model Saving and Loading**:
   * Trained models are saved (both architecture and weights) and can be reloaded to avoid retraining.
   * The training history (like accuracy over epochs) is also saved.
5. **Prediction and Evaluation**:
   * Once trained, the model predicts classes on test images.
   * Multiple performance metrics are calculated: Accuracy, Precision, Recall, F1-score, Sensitivity, Specificity.
   * A confusion matrix and classification report are displayed using heatmaps.
6. **Graph Visualization**:
   * Graphs show how accuracy and loss evolve during training, helping understand model learning behavior.

**9.2 Dataset Description**

The goal of the dataset, which consists of 2400 images from traffic surveillance film captured , is to enhance accident detection by utilizing CNNs and other deep learning models. The CCTV footage images and dashboard cameras used to capture these pictures show a range of traffic accidents on city and interstate roads. In addition to 1200 images of non-accident traffic, the dataset is evenly distributed and contains 1200 images of accidents, including near-miss incidents, car crashes, skidding, and quick braking. To improve model evaluation and training, each frame is tagged with the relevant metadata. Weather intensity, fog density classification (light mist, moderate fog, or dense fog), vehicle speed (which gives an estimate of motion dynamics), accident severity (which categorizes incidents as minor, moderate, or severe), and timestamp (which shows the time of the incident) are some examples of these metadata. Through the use of CNN with VGG16 based models, traffic monitoring systems can be automated, improving road safety and intelligent surveillance during bad weather by facilitating speedier accident detection and response

. 

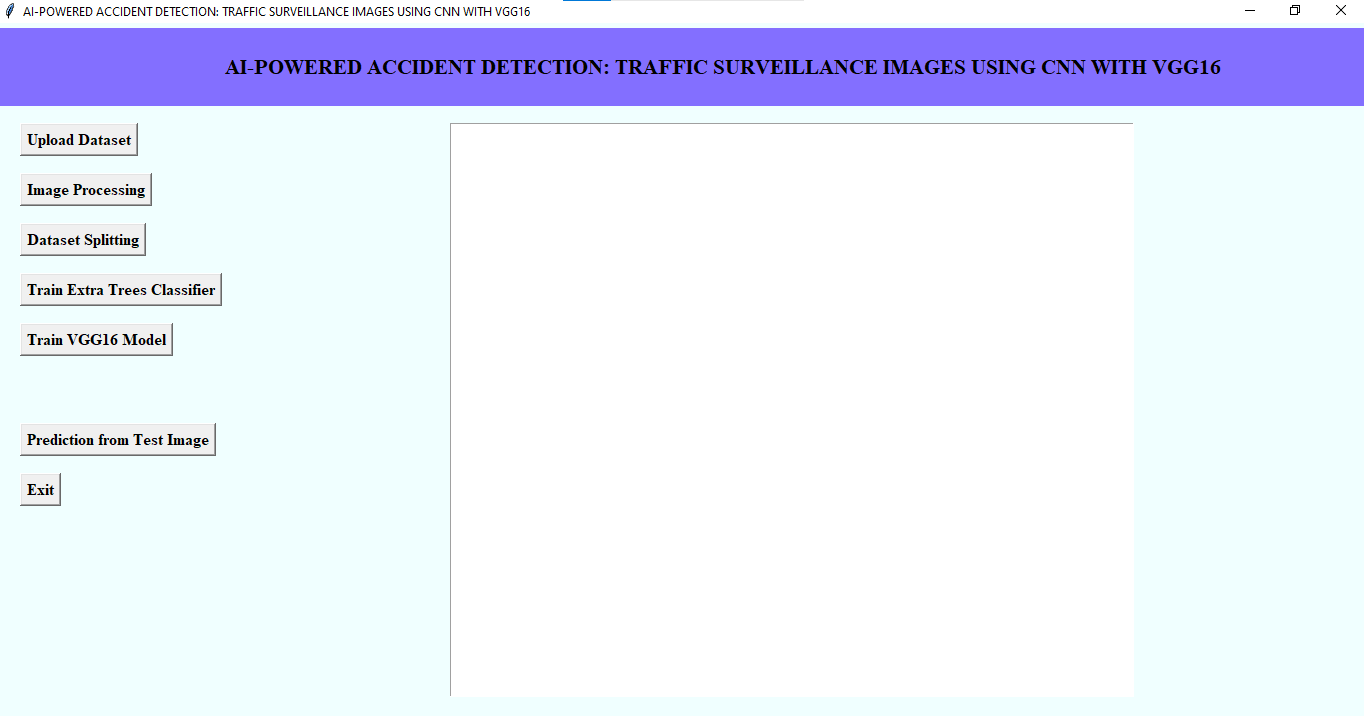
9.1 Accident images

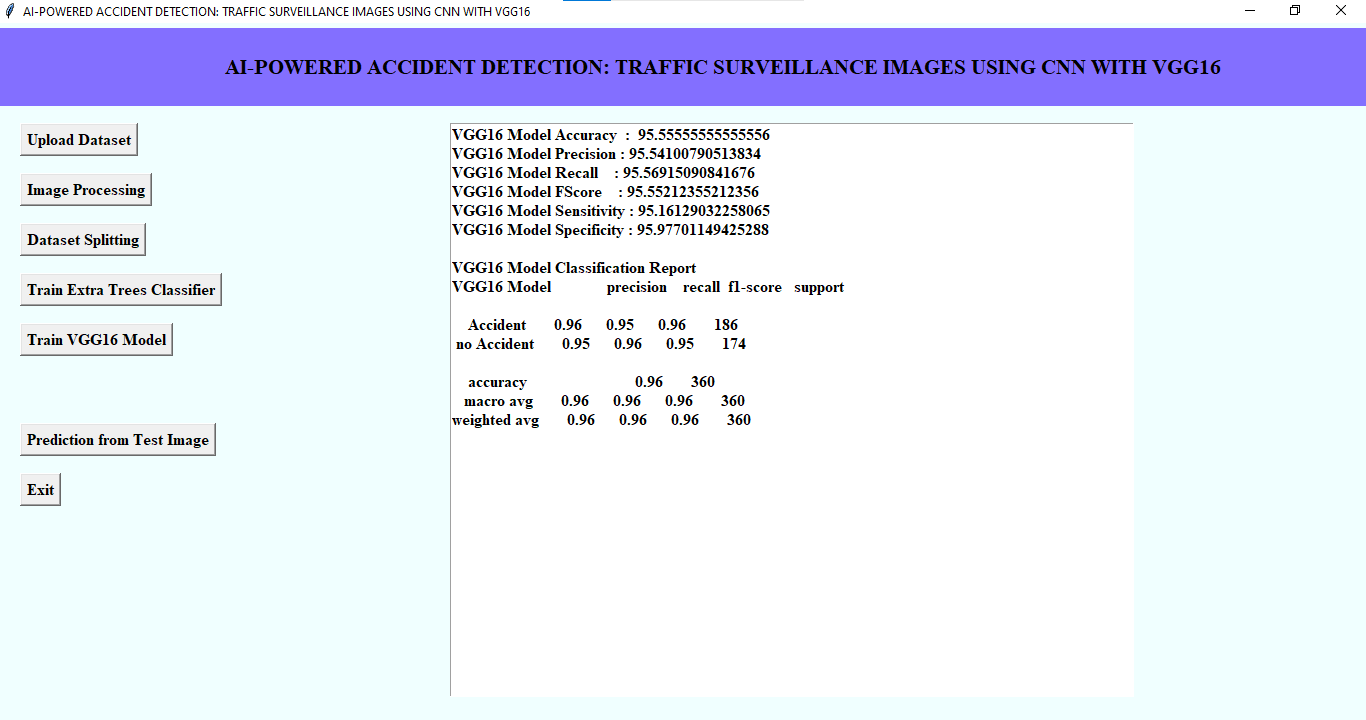
9.2 Not Accident images

**9.3 Result and Description**

The Tkinter-based GUI for the **CNN** VGG16 **with Accident Detection System** offers a user-friendly interface for managing datasets, training the model, and detecting accidents in videos. It allows users to upload training and testing datasets, train the CNN with VGG16 model using image data, and analyze image for accident detection

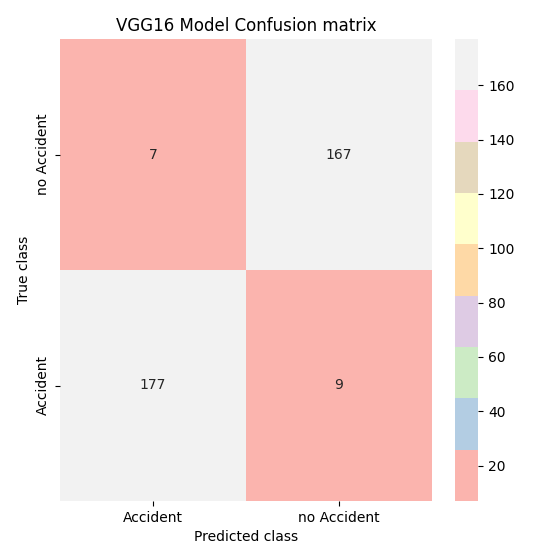


9.3 GUI



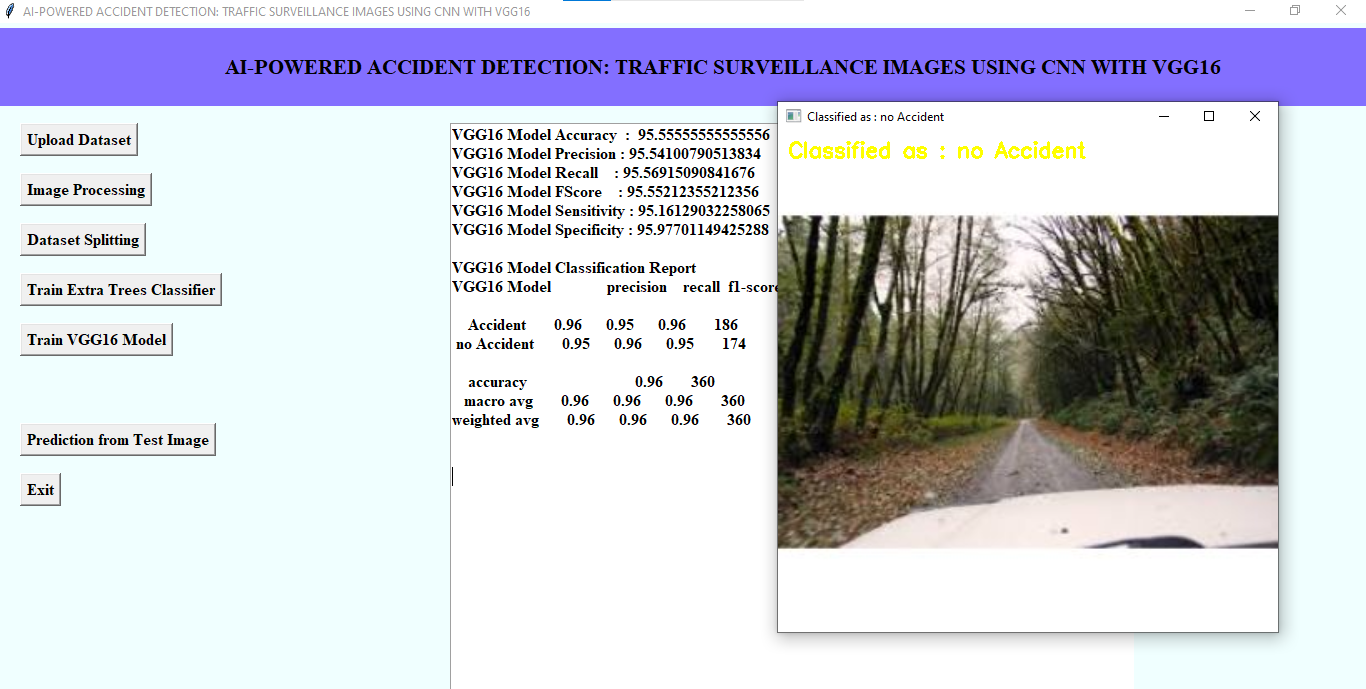
9.4 Performance Evaluations

The CNN with VGG16 Based Accident Detection Model has demonstrated a remarkable 95% prediction accuracy, as illustrated in Figure 2, demonstrating its great reliability in differentiating between accident and non-accident scenarios. In order to avoid overfitting, the model has VGG layers, followed by MaxPooling2D and Dropout layers. After the Flatten layer transforms the retrieved features into a 1D vector, a fully connected layer with 1024 neurons processes the data, and a final Dense layer with 2 neurons classifies the results.



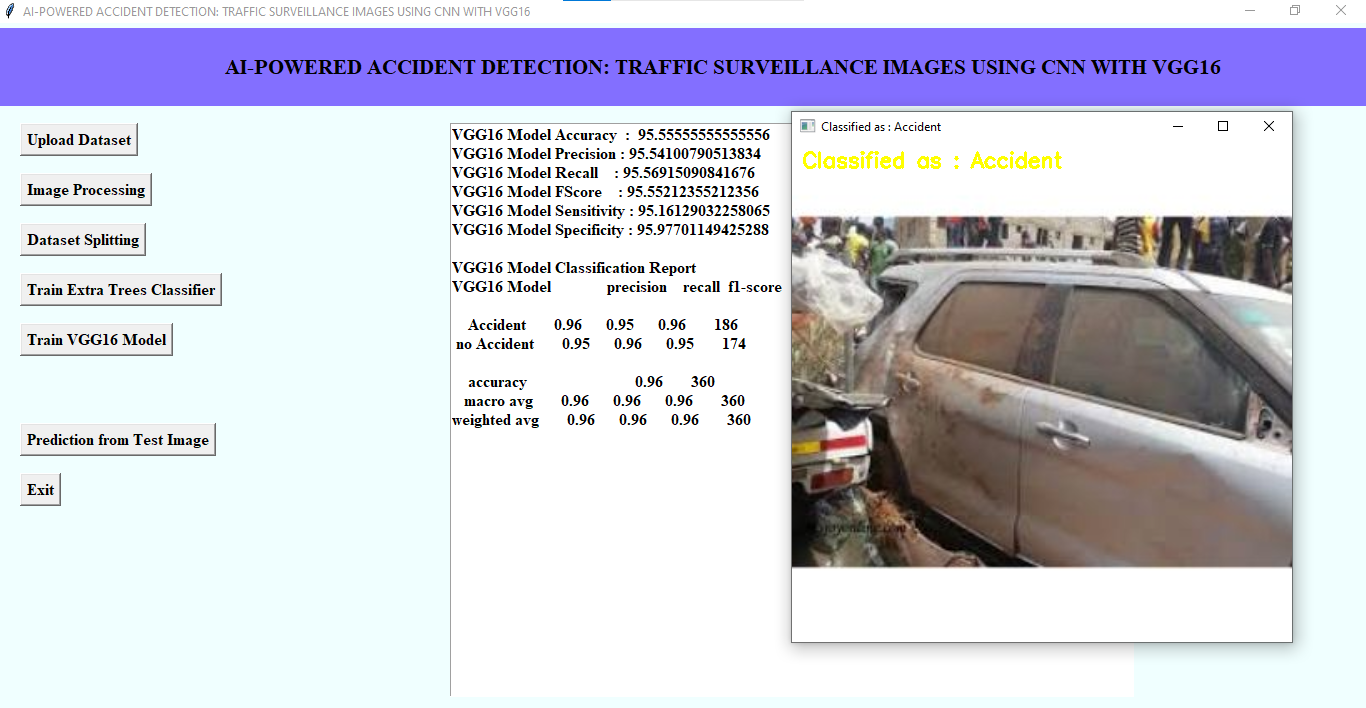
9.4 Confusion matrix of CNN with VGG16

Figure 3 show that For classification models, a confusion matrix is a performance assessment instrument for aggregating predictions against real-world class labels. The columns of the supplied VGG16 model confusion matrix show the predicted classes; the rows show the true classes. The model distinguished 177 cases of "Accident" from 167 cases of "No Accident." But it misclassified nine "Accident" cases as "No Accident" (false negatives) and seven "No Accident" cases as "Accident" (false positives).



9.6 Detected as no accident

Figure 4 illustrates how the CNN with VGG16 based Accident Detection System examines a frame of CCTV footage. The wording that is seen in the upper-left corner of the image while the system is actively processing it reads "No Accident Detected," indicating that the model has not detected any accidents in this specific image.



9.7 image detected as accident

Figure 5 show that the given figure shows the CNN with VGG16 based Accident Detection System successfully identifying an accident in the CCTV footage. The upper-left corner of the screen displays the text "Accident Detected" indicating that the deep learning model has classified this image as an accident scene. The system processes real-time images, using a CNN with VGG16 to detect anomalies such as crashes or collisions.

**10. CONCLUSION AND FUTURE SCOPE**

**Conclusion**

By using CCTV footage to automatically identify accidents in real time, the CNN with VGG16 Accident Detection System, a powerful deep learning technology, improves road safety. Through the use of CNNs, the system efficiently examines images and accurately identifies collision events. Incorporating elements such as picture processing, categorization, which may lower casualties and enhance emergency support. The model is dependable in distinguishing between typical traffic situations and accident situations, as seen by its high forecast accuracy (95.6%). Further bolstering the system's usefulness in smart surveillance is its capacity to deliver real-time alerts, including visual warnings and audio messages. Road safety can be improved by integrating this technology into transportation management, smart city development, and traffic monitoring systems. Proactive accident prevention measures and increased detection accuracy may be made possible by future developments like the incorporation of IoT-based sensors and sophisticated deep learning models.

**Future Scope**

The future scope of the CNN with VGG16-based Accident Detection System lies in its potential to integrate with emerging technologies for enhanced road safety. By incorporating Internet of Things (IoT)-based sensors and real-time traffic data, the system could achieve even higher accuracy and responsiveness, enabling predictive accident prevention and dynamic traffic management. Advanced deep learning models and reinforcement learning algorithms could further improve detection capabilities, while fusion with autonomous vehicle technologies could offer proactive safety solutions. Additionally, expanding the system to recognize a broader range of traffic incidents, such as pedestrian accidents and traffic violations, would create a comprehensive traffic surveillance and management solution, contributing significantly to smart city infrastructure and sustainable transportation systems.

**REFERENCES**

[1] World Health Organization, 2019. "Global status report on road safety 2015". [Website]. http://www.who.int/violence\_injury\_prevention/road\_safety\_status/20 15/en/ is what is accessible. [retrieved: 07-Mar-2019].

.[2] "An enhanced accident detection and victim status indicating system: Prototype." IEEE, 2012. Annual India Conference (INDICON).

[3] "Lexus Enform" (2019). [Website]. The website https://www.lexus.com/enform is accessible. [retrieved: 07-Mar-2019].

[4] "SOSmart automated vehicle collision detection and alerting application," SOSmart automated vehicle collision detection application, 2019. You can access it online at http://www.sosmartapp.com. [retrieved: 07-Mar-2019].

[5] C. Kockan, PhD thesis, "Vehicular Communication," Istanbul Technical University, 2008   
[6] Athanasios V. Vasilakos, Yuanyuan, Deshi Li, and Zeng. "Opportunistic fleets for road event detection in vehicular sensor networks." Networks Wireless 22.2 (2016): 503-521.   
[7] Szegedy, Christian, and colleagues, "Rethinking the inception architecture for computer vision." IEEE Conference Proceedings on Pattern Recognition and Computer Vision, 2016.   
[8] Christian Szegedy et al. "Going deeper with convolutions." IEEE Conference Proceedings on Pattern Recognition and Computer Vision, 2015.