

CHAPTER 1

INTRODUCTION

1.1 Motivation

With rapid economic growth and improved living standards, road usage has increased significantly, leading to a rise in road traffic accidents. These incidents result in considerable losses of life and property. A critical factor contributing to the high fatality rates is the delayed emergency response. Research shows that even a 60-second reduction in response time can improve the survival rate by 6%. Technological interventions like increased seatbelt use have already demonstrated the potential to save lives — a 1% increase saves approximately 136 lives annually. Thus, leveraging modern technologies to detect accidents early and trigger emergency responses promptly is crucial to improving road safety and saving lives.

1.2 Problem Definition

Traditional accident detection and response systems rely heavily on human monitoring, which can lead to delayed or missed reactions, especially under poor visibility conditions. Current surveillance systems are not equipped to detect vehicle accidents automatically and promptly alert emergency services. The major problem is the absence of an automated, real-time accident detection system that can operate effectively across varying weather and lighting conditions using existing road-side CCTV infrastructure.

1.3 Objective of the Project

The primary objective of this project is to develop a real-time accident detection and emergency alert system using deep learning techniques. The system aims to:

Detect accidents using object tracking and movement anomalies from CCTV footage.

Analyze vehicle speed and trajectory to identify unusual behavior indicative of collisions.

Achieve high detection accuracy with a low false alarm rate under diverse environmental conditions such as daylight, rain, snow, and low visibility.

Minimize emergency response time by automatically generating alerts when accidents are detected.

Reduce reliance on human monitoring by implementing a fully automated accident detection pipeline.

1.4 Limitation of the Project

Despite its effectiveness, the project has some limitations:

Dataset Dependency: The system's performance heavily relies on the quality and diversity of the training dataset. It may struggle in unseen or rare accident scenarios not represented in the training data.

Environmental Conditions: Although the model is tested under varied conditions, extremely poor visibility (e.g., heavy fog or night without proper lighting) can affect detection accuracy.

Hardware Constraints: Real-time performance requires high-end computational resources, which may not be feasible in all deployment scenarios.

Limited Context Understanding: The system detects based on motion and visual cues without a deeper contextual understanding, which may sometimes result in false positives in complex scenes (e.g., sudden stops not due to accidents).

Scalability: Large-scale deployment across cities might require significant integration with existing traffic surveillance and emergency systems.

CHAPTER 2

LITERATURE SURVEY

2.1. Introduction

With the rapid rise in vehicle usage, road accidents have become a major concern, causing significant loss of life and property. A key issue is the delay in emergency response, which greatly impacts survival rates. Studies show that reducing response time by even one minute can save more lives.

Manual monitoring of CCTV footage is inefficient and prone to delays. To address this, the project proposes an automated, real-time automatic detection of unexpected accident under bad cctv monitorint condtions using deep learning and computer vision. By analyzing vehicle movement and detecting anomalies through CCTV footage, the system aims to quickly identify accidents and notify emergency services, ultimately reducing response times and saving lives.

2.2. Existing System

In the existing system detects vehicle object and classifies the type of vehicle by Convolutional Neural Network (CNN). The vehicle object tracking algorithm tracks the vehicle object by changing the tracking center point according to the position of the recognized vehicle object on the image. Then, the monitor shows a localized image like a bird's viewpoint with the visualized vehicle objects, and the system calculates the distance between the driving car and the visualized vehicle objects

Title: Object Detection for dummies part 3: R-cnn family

Authors: Abrantes and Marques

Overview: This article addresses the problem of tracking moving objects using deformable models. A Kalman-based algorithm is presented, inspired by a new class of constrained clustering methods, proposed by in the context of static shape estimation. A set of data centroids is tracked using intra-frame and inter-frame recursions. Centroids are computed as weighted sums of the edge points belonging to the object boundary. The use of centroids introduces competitive learning mechanisms in the tracking algorithm leading to improved robustness with respect to occlusion and contour sliding. Experimental results with traffic sequences are provided.

Title: An algorithm for centroid-based tracking of moving objects

Authors: J. C. Nascimento, A. J. Abrantes, and J. S. Marque

Overview: Two fuzzy logic based image-centroid tracking algorithms are presented. Fuzzy logic finds numerous applications in several signal/image processing, control, and sensor data fusion applications, and it is also one of the ingredients for building practical artificial intelligence (AI) systems. These centroid tracking algorithms (CTA) are based on: a) adaptive neuro fuzzy inference system (ANFIS), and b) fuzzy logic (FL)-function process; and both use Kalman filter (KF) for tracking. These centroid tracking algorithms are evaluated using MATLAB-based simulated synthetic image data, and although the performance of both the CTAs has been found to be very satisfactory, the FL-based CTKF performed better than the ANFIS-based CTKF. In certain image-based air traffic control and/or air defense systems (ATC/ADS), automatic target acquisition, identification and tracking by processing a sequence of real images of the moving object or target are very essential. Many such and similar applications require an algorithm for image detection, segmentation, feature computation, selection, classification and tracking. In tracking of a moving

target using image data that involves image processing, at each time sampling time, the estimates of the target's current position and velocity are obtained.

2.3. Proposed system

In the proposed system we attempt is made for generate an object detection & tracking system (ODTS) with yolo, that can obtain moving information of target objects with names by combining object tracking algorithm with the deep learning-based object detection process. It is assumed that ODTS has been trained enough to perform object detection properly on a given image frame. ODTS receives selected frames of video at specified time interval c and gains sets of coordinates, n BBoxes are detected. BBoxT of objects on the given image frame at the time T , from the trained object detection system. The corresponding type or class $ClassT$ of each detected object BBoxT is simultaneously classified by the object detection module.

Advantages

1. A deep learning model of R-CNN was used for training with YOLO object detection model.
2. This object tracking module was composed by introducing an object tracking model called YOLO.

CHAPTER 3

SYSTEM ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

1. Accident Detection

- The system must be able to detect accidents in real-time using sensors (e.g., vehicle crash sensors, cameras) or image recognition software (e.g., through deep learning models for object detection).
- The system must identify different types of accidents such as collisions, rollovers, and pedestrian accidents.

2. Data Processing

- The system must process input data (from cameras, sensors, etc.) and analyze it using deep learning algorithms (e.g., Convolutional Neural Networks, Recurrent Neural Networks).
- The system should extract critical information such as the severity of the accident, the location, and the number of people involved.

3. Alert Generation

- Once an accident is detected, the system must immediately send an alert to emergency services (police, fire, medical) with the necessary details like the location and type of accident.
- Alerts should be sent in multiple formats (e.g., SMS, email, app notifications) to ensure timely delivery.

4. Location Tracking

- The system must automatically detect the geographic location of the accident using GPS or a similar system.
- It should integrate with mapping systems to provide accurate location data to emergency

3.2. NON-FUNCTIONAL REQUIREMENTS

1. Performance

The system should process data in real-time with minimal delay (e.g., within seconds of an accident occurrence). The deep learning model should perform at high speed, especially in processing high-resolution images or sensor data without latency

2. Scalability

The system should be scalable to handle a large number of accidents and locations.

It must support expansion for integration with additional sensors or cameras across various returns.

3. Availability

The system should have high availability, ensuring 24/7 operation with minimal downtime.

There should be redundancy for critical components (e.g., cloud backup, failover mechanisms).

4. Reliability

system should reliably detect and report accidents without failure. If a failure occurs, it must automatically recover or alert system administrators.

It should ensure high accuracy in accident detection and reduce the number of false negatives (i.e., missing real accidents)

3.3. HARDWARE REQUIREMENTS (Minimum Requirement)

- System : Intel Core i3.
- Hard Disk : 5 GB(min).
- Monitor : 14' Colour Monitor
- Input Devices : Keyboard, Mouse⁷⁹
- Ram : 4 GB.

3.4. SOFTWARE REQUIREMENTS (Minimum Requirement)

Operating system	:	Windows 10.
Coding Language	:	Python
Tool	:	Visual Studio Code
Database	:	SQLite

CHAPTER 4

SYSTEM DESIGN

4.1. SYSTEM ARCHITECTURE

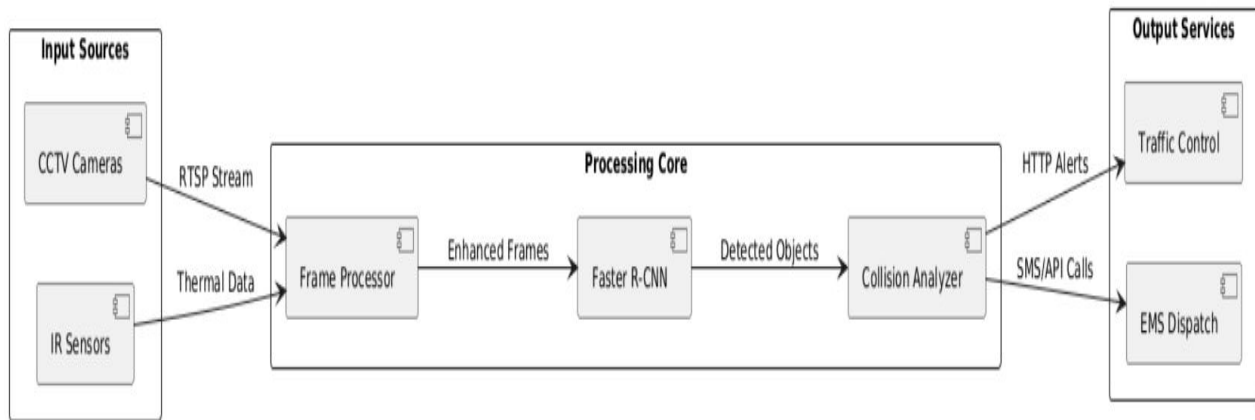


Fig 4.1 Video Detection System

When a video detection system installed in a CCTV camera detects an accident, an emergency alert is issued, and a message with the percentage of collision severity is sent to the relevant accident and emergency departments' computers through servers. The proposed framework can detect accidents with some Detection Rate and False Alarm Rate on accident footages captured in various contextual circumstances.

4.2 MODULES

- Object Detection and Tracking
- RCNN
- Average Precision

1. Object Detection and Tracking Module

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the bounding box of the image are considered detections. We apply a Regional Convolution neural network to the full image.

This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. Our model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image.

2. RCNN (Regional Convolution Neural Network) Module

R-CNN models first select several proposed regions from an image (for example, anchor boxes are one type of selection method) and then label their categories and bounding boxes (e.g., offsets). Then, they use a CNN to perform forward computation to extract features from each proposed area. Afterwards, we use the features of each proposed region to predict their categories and bounding boxes. Then, based on the detected object information, a dependent object tracking module is initiated to assign the unique ID number to each of the detected objects, ID_t and predict the next position of each of the objects, BBOX. The number of tracking BBox u is different from n . But If past tracked BBox is 0, the number of tracking BBox equals to the number of the detected objects.

3. Average Precision

AP values for the target objects to be detected, in the training dataset, the number of Car objects is the largest object and very high AP value was obtained for the Car object in comparison with other classes. That is, the object detection performance of deep running of the Car in the video was expected to be highly reliable. On the other hand, AP for Person object results in relatively low value because Person object exists long, tiny shape in small size. The AP of Fire object was high, but false detection for the object might be highly possible as the number of the objects available for training was very small, None the less, training about deep learning, including No Fire objects, could reduce the false detection of Fire object. However, to detect the Fire in the tunnel control center, it was necessary to collect and involve more images of a Fire event in trainin

4.3 DATA FLOW DIAGRAM

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

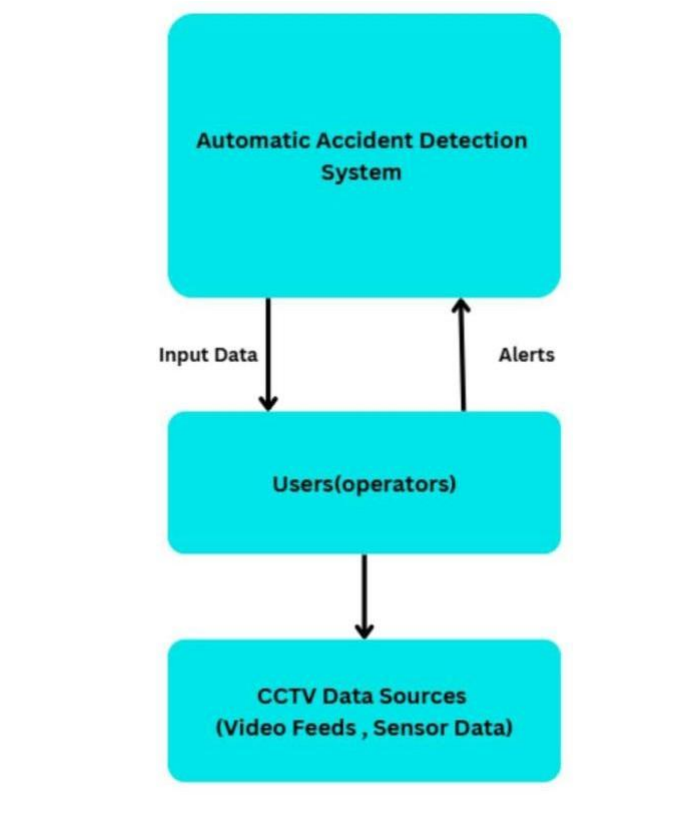


Fig 4.3.1 Level 0 Data Flow diagram

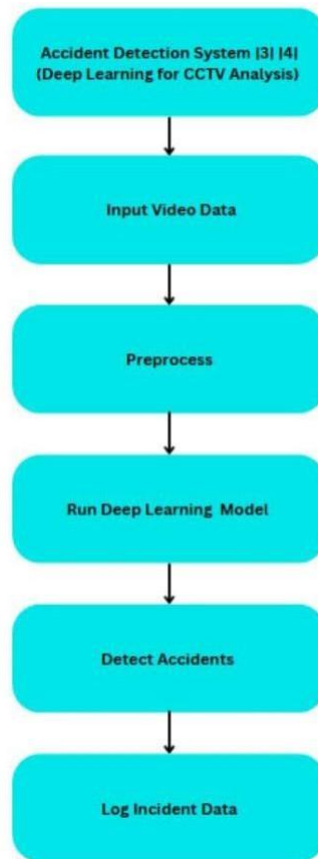


Fig 4.3.2 Level 1 Data Flow diagram

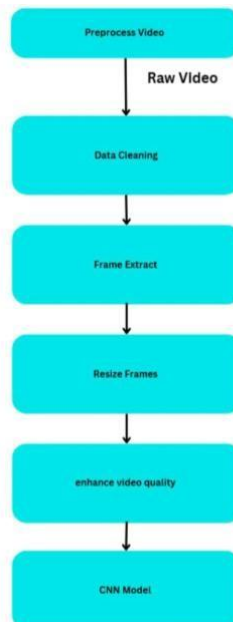


Fig 4.3.3 Level 2 Data Flow diagram

4.4 UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non- software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS

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

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

4.4.1 USE CASE DIAGRAM

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural 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.

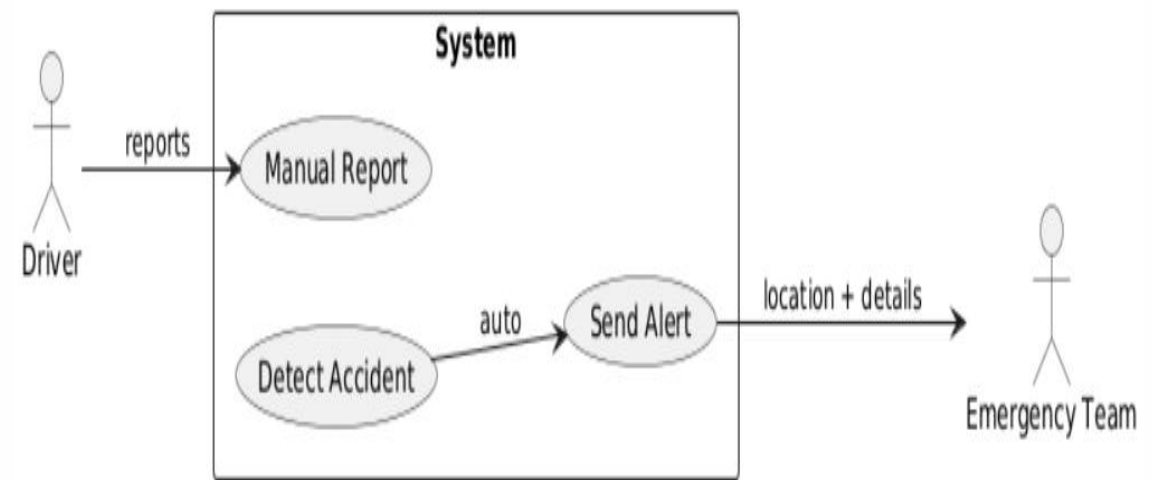


Fig 4.4.1 Use Case diagram

4.4.2 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

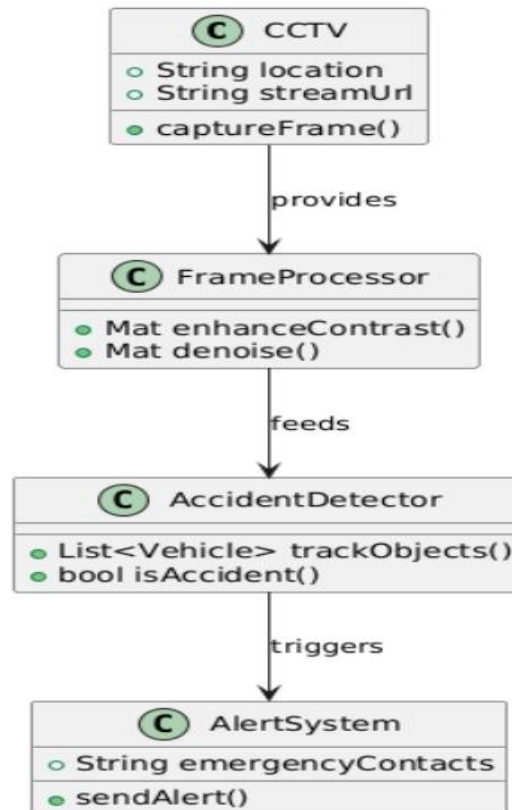


Fig 4.4.2 Class diagram

4.4. 3 SEQUENCE DIAGRAM

A sequence diagram in Unified Modelling 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. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

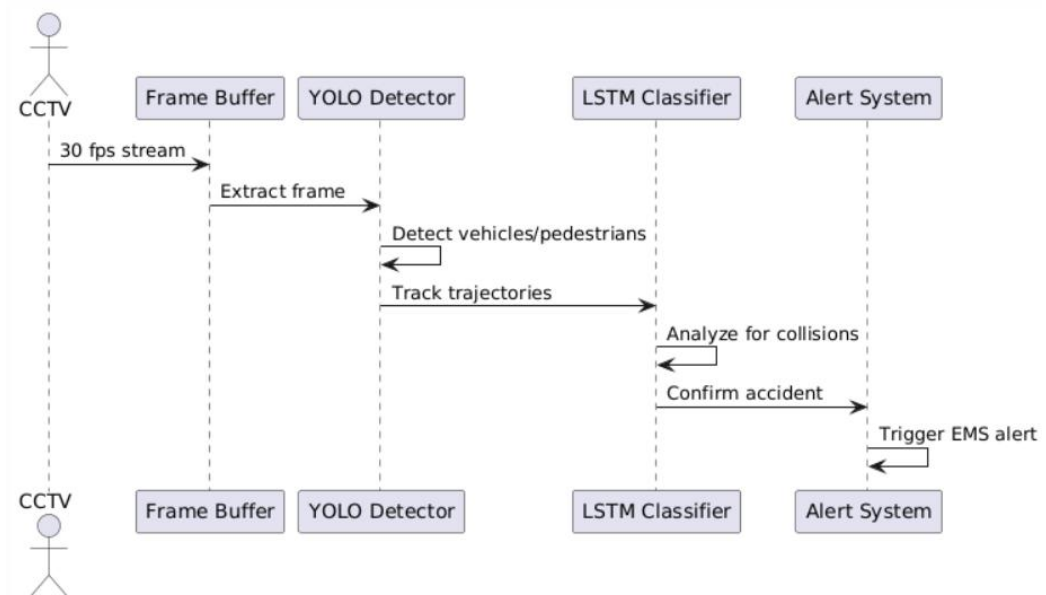


Fig 4.4.3 Sequence diagram

4.4.4 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

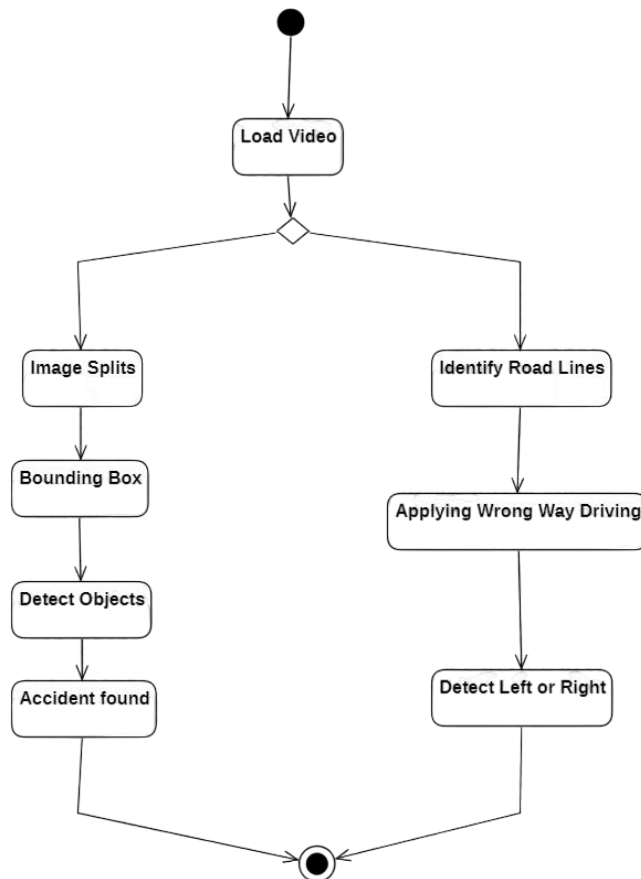


Fig 4.4.4 Activity diagram

CHAPTER 5

IMPLEMENTATION & RESULT

5.1 Introduction

Software Description: Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site <https://www.python.org/> and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in The Python Standard Library.

There are many programming languages available today; this is the usual first question of newcomers. Given that there are roughly 1 million Python users out there at the moment, there really is no way to answer this question with complete accuracy; the choice of development tools is sometimes based on unique constraints or personal preference. But after teaching Python to roughly 225 groups and over 3,000 students during the last 12 years, some common themes have emerged. The primary factors cited by Python users seem to be these:

Software quality

For many, Python's focus on readability, coherence, and software quality in general sets it apart from other tools in the scripting world. Python code is designed to be readable, and hence reusable and maintainable—much more so than traditional scripting languages. The uniformity of Python code makes it easy to understand, even if you did not write it. In addition, Python has deep support for more advanced software reuse mechanisms, such as object-oriented programming (OOP).

Developer productivity

Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++, and Java. Python code is typically one-third to one-fifth the size of equivalent C++ or Java code. That means there is less to type, 3 less to debug, and less to maintain after the fact. Python programs also run immediately, without the lengthy compile and link steps required by some other tools, further boosting programmer speed.

Program portability

Most Python programs run unchanged on all major computer platforms. Porting Python code between Linux and Windows, for example, is usually just a matter of copying a script's code between machines. Moreover, Python offers multiple options for coding portable graphical user interfaces, database access programs, web based systems, and more. Even operating system interfaces, including program launches and directory processing, are as portable in Python as they can possibly be.

Support libraries

Python comes with a large collection of prebuilt and portable functionality, known as the standard library. This library supports an array of application-level programming tasks, from text pattern matching to network scripting. In addition, Python can be extended with homegrown libraries and a vast collection of third-party application support software. Python's third-party domain offers tools for website construction, numeric programming,

serial port access, game development, and much more.

The NumPy extension, for instance, has been described as a free and more powerful equivalent to the Matlab numeric programming system.

Component integration

Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Such integrations allow Python to be used as a product customization and extension tool. Today, Python code can invoke C and C++ libraries can be called from C and C++ programs, can integrate with Java and .NET components, can communicate over frameworks such as COM, can interface with devices over serial ports, and can interact over networks with interfaces like SOAP, XML-RPC, and CORBA. It is not a standalone tool.

Enjoyment

Because of Python's ease of use and built-in toolset, it can make the act of programming more pleasure than chore. Although this may be an intangible benefit, its effect on productivity is an important asset.

Of these factors, the first two (quality and productivity) are probably the most compelling benefits to most Python users.

Python a “Scripting Language”

Python is a general-purpose programming language that is often applied in scripting roles. It is commonly defined as an object-oriented scripting language—a definition that blends support for OOP with an overall orientation toward scripting roles. In fact, people often use the word “script” instead of “program” to describe a Python code file. In this book, the terms “script” and “program” are used interchangeably, with a preference for “script” to describe a simpler top-level file and “program” to refer to a more sophisticated multiline application.

Because the term “scripting language” has so many different meanings to different observers, some would prefer that it not be applied to Python at all. In fact, people tend to make three very different associations, some of which are more useful than others, when they hear Python labelled as such:

Shell Tools

Sometimes when people hear Python described as a scripting language, they think it means that Python is a tool for coding operating-system-oriented scripts

Control language

To others, scripting refers to a “glue” layer used to control and direct (i.e., script) other application components. Python programs are indeed often deployed in the context of larger applications. For instance, to test hardware devices, Python programs may call out to components that give low-level access to a device. Similarly, programs may run bits of Python code at strategic points to support end-user product customization without the need to ship and recompile the entire system’s source code. Python’s simplicity makes it a naturally flexible control tool. Technically, though, this is also just a common Python role; many (perhaps most) Python programmers code standalone scripts without ever using or knowing about any integrated components. It is not just a control language.

Ease of use

Probably the best way to think of the term “scripting language” is that it refers to a simple language used for quickly coding tasks. This is especially true when the term is applied to Python, which allows much faster program development than compiled languages like C++. Its rapid development cycle fosters an exploratory, incremental mode of programming that has to be experienced to be appreciated. Don’t be fooled, though—Python is not just for simple tasks. Rather, it makes tasks simple by its ease of use and flexibility. Python has a simple feature set, but it allows programs to scale up in sophistication as needed. Because of that, it is commonly used for quick tactical tasks and longer-term strategic development.

So, is Python a scripting language or not? It depends on whom you ask. In general, the term “scripting” is probably best used to describe the rapid and flexible mode of development that Python supports, rather than a particular application domain.

Uses Of Python Today

Python enjoys a large user base and a very active developer community. Because Python has been around for some 19 years and has been widely used, it is also very stable and robust. Besides being employed by individual users, Python is also being applied in real revenue generating products by real companies. For instance:

- Google makes extensive use of Python in its web search systems, and employs Python's creator.
- The YouTube video sharing service is largely written in Python.
- The popular BitTorrent peer-to-peer file sharing system is a Python program.
- Google's popular App Engine web development framework uses Python as its application language.
- EVE Online, a Massively Multiplayer Online Game (MMOG), makes extensive use of Python.
- Maya, a powerful integrated 3D modelling and animation system, provides a Python scripting API.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.
- Industrial Light & Magic, Pixar, and others use Python in the production of animated movies.
- JPMorgan Chase, UBS, Getco, and Citadel apply Python for financial market forecasting.
- NASA, Los Alamos, Fermilab, JPL, and others use Python for scientific programming tasks.
- iRobot uses Python to develop commercial robotic devices.
- ESRI uses Python as an end-user customization tool for its popular GIS mapping products.
- The NSA uses Python for cryptography and intelligence analysis.
- The IronPort email server product uses more than 1 million lines of Python code to do its job.
- The One Laptop Per Child (OLPC) project builds its user interface and activity model in Python.

And so on. Probably the only common thread amongst the companies using Python today is that Python is used all over the map, in terms of application domains. Its general-purpose nature makes it applicable to almost all fields, not just one. In fact, it's safe to say that virtually every substantial organization writing software is using Python, whether for short-term tactical tasks, such as testing and administration, or for long-term strategic product development. Python has proven to work well in both modes.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. It provides a variety of functions to visualize data and create static, animated, and interactive plots. Matplotlib is widely used for tasks ranging from simple line plots to complex visualizations.

The Visualization Design using matplotlib

1. Bar Graph
2. Pie Chart
3. Box Plot
4. Histogram
5. Line Chart and Subplots
6. Scatter Plot

Bar Graph using Matplotlib

Bar graphs are best used when we need to compare the quantity of categorical values within the same category. Bar graphs should not be used for continuous values.

Bar graph is generated using **plt.bar()** in matplotlib

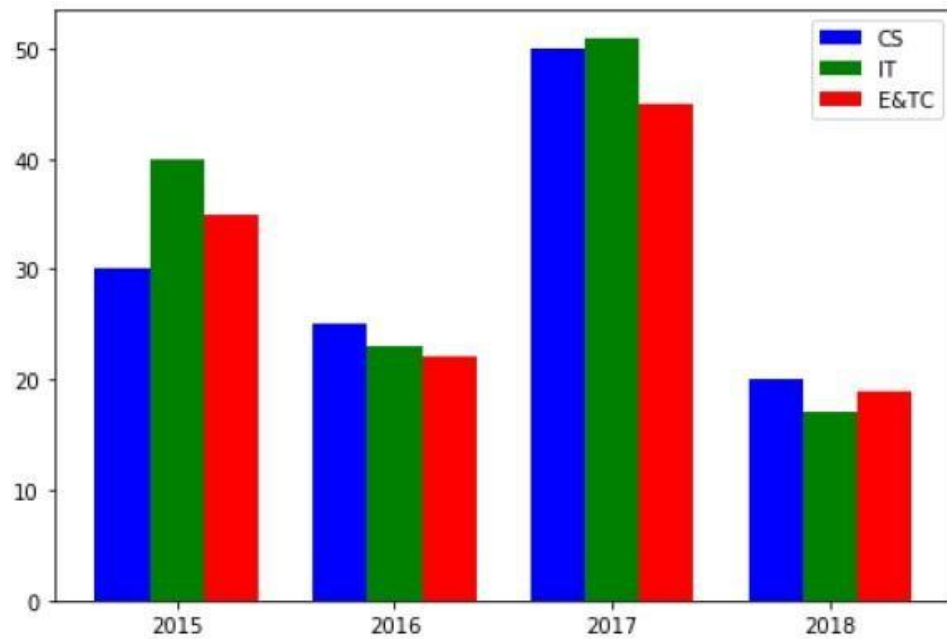


Figure 5.1 Bar Graph

Pie Chart using Matplotlib

A pie chart is suitable to show the proportional distribution of items within the same category. `plt.pie()` is used to draw the pie chart and adjust its parameters to make it more appealing.

A pie chart is rendered useless when there are a lot of items within a category. This will decrease the size of each slice and there will be no distinction between the items.

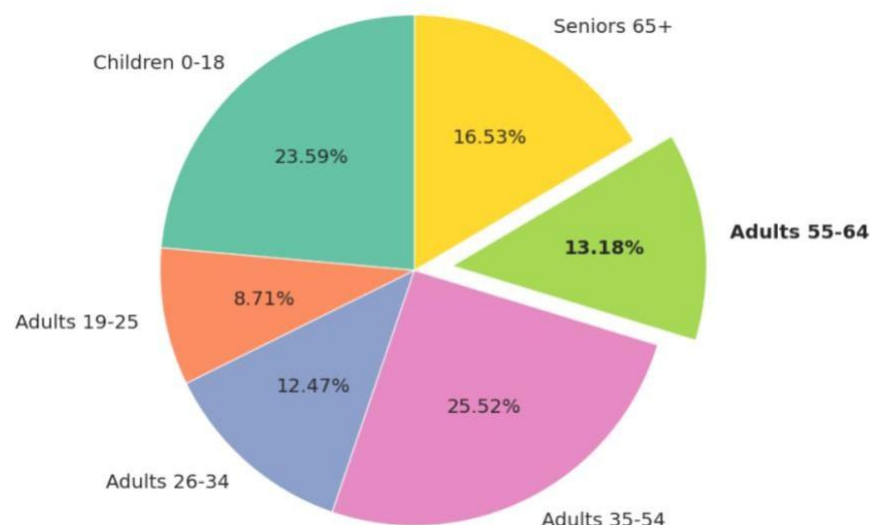


Figure 5.2 Pie Chart

Box Plot using Matplotlib

Box plot gives statistical information about the distribution of numeric data divided into different groups. It is useful for detecting outliers within each group. The lower, middle and upper part of the box represents the **25th**, **50th**, and **75th percentile** values respectively. Box plot does not show the distribution of data points within each group.

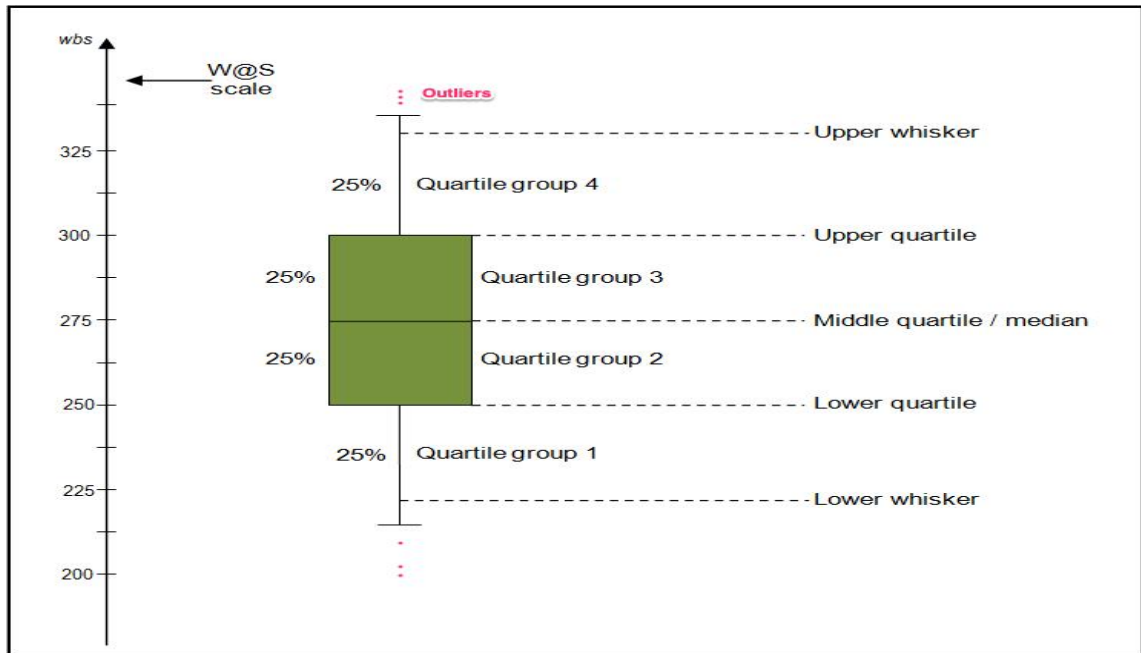


Figure 5.3 Box Plot

Histogram using Matplotlib

A histogram shows the distribution of numeric data through a continuous interval by segmenting data into different bins. Useful for inspecting skewness in the data. It is easy to confuse histograms with bar plots. But remember, histograms are used with continuous data whereas bar plots are used with categorical data.

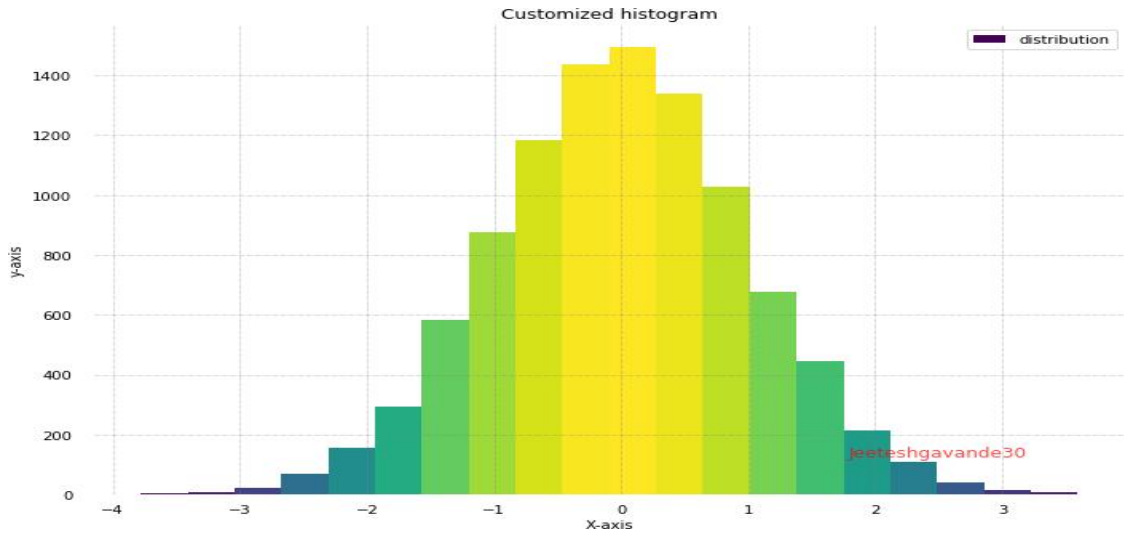


Figure 5.4 Histogram

Line plot and sub plots using Matplotlib

A line plot is useful for visualizing the trend in a numerical value over a continuous time interval. Matplotlib subplots makes it easy to view and compare different plots in the same figure. The `plt.subplots()` figure returns the figure and axes. You can provide as an input to the function how you want to display the axes within the figure. These will be adjusted using the *nrows* and *ncols* parameters. You can even adjust the size of the figure using the *figsize* parameter.

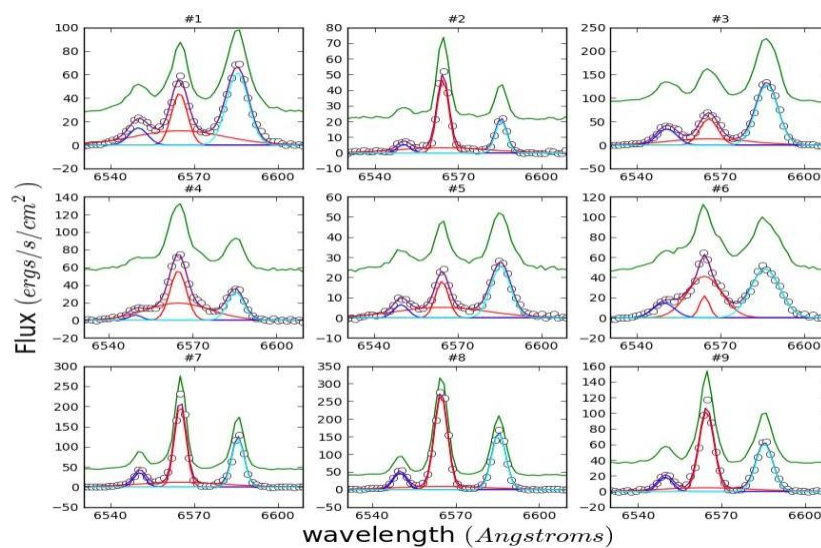


Figure 6.5 Subplot

Scatter plot using Matplotlib

Scatter plots are useful for showing the relationship between two variables. Any correlation between variables or outliers in the data can be easily spotted using scatter plots.

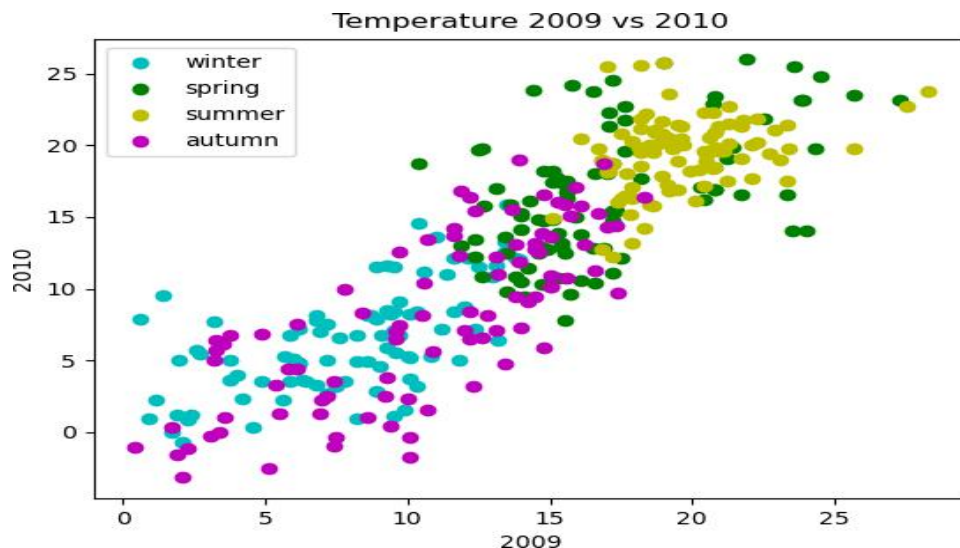


Figure 5.6 Scatter Plot

Seaborn

Seaborn is a statistical data visualization library based on Matplotlib. Seaborn is a powerful and flexible data visualization library in Python that offers an easy-to-use interface for creating informative and aesthetically pleasing statistical graphics. It provides a range of tools for visualizing data, including advanced statistical analysis, and makes it easy to create complex multi-plot visualizations. Seaborn's key benefit lies in its capability to generate attractive plots with minimal coding efforts. It provides a range of default themes and color palettes, which you can easily customize to suit your preferences. Additionally, Seaborn offers a range of built-in statistical functions, allowing users to easily perform complex statistical analysis with their visualizations. Another notable feature of Seaborn is its ability to create complex multi-plot visualizations. With Seaborn, users can create grids of plots that allow for easy comparison between multiple variables or subsets of data. This makes it an ideal tool for exploratory data analysis and presentation.

Plot types in seaborn

- **Line Plot:** Line plots are used to visualize trends in data over time or other continuous variables. In a line plot, each data point is connected by a line, creating a smooth curve. In Seaborn, line plots can be created using the `lineplot()` function.
- **Histograms:** Histograms visualize the distribution of a continuous variable. In a histogram, the data is divided into bins and the height of each bin represents the frequency or count of data points within that bin. In Seaborn, histograms can be created using the `histplot()` function.
- **Box Plot:** Box plots are a type of visualization that shows the distribution of a dataset. They are commonly used to compare the distribution of one or more variables across different categories.
- **Violin Plot:** A violin plot is a type of data visualization that combines aspects of both box plots and density plots. It displays a density estimate of the data, usually smoothed by a kernel density estimator, along with the interquartile range (IQR) and median in a box plot-like form. The width of the violin represents the density estimate, with wider parts indicating higher density, and the IQR and median are shown as a white dot and line within the violin.

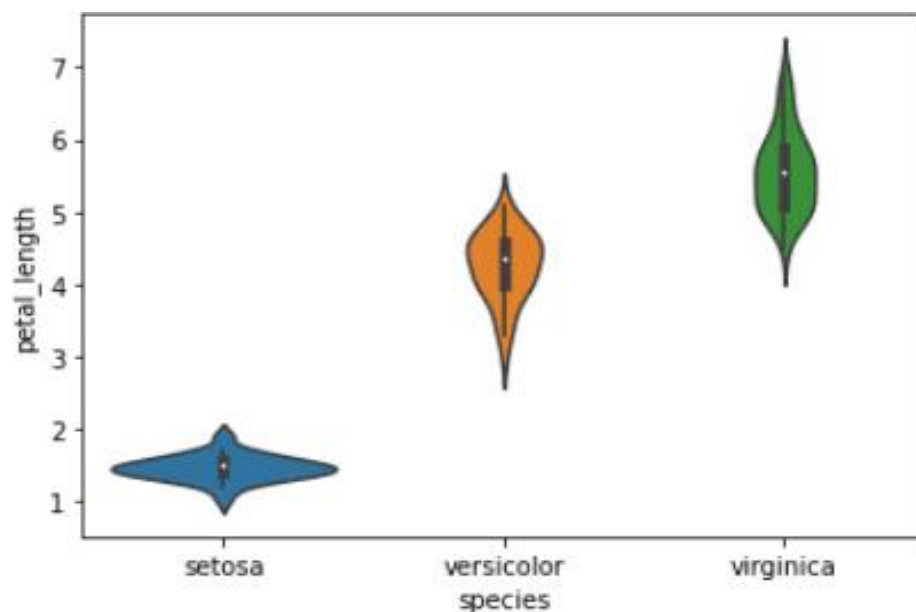


Figure 5.7 Violin Plot

HeatMap

A heatmap is a graphical representation of data that uses colors to depict the value of a variable in a two-dimensional space. Heatmaps are commonly used to visualize the correlation between different variables in a dataset.

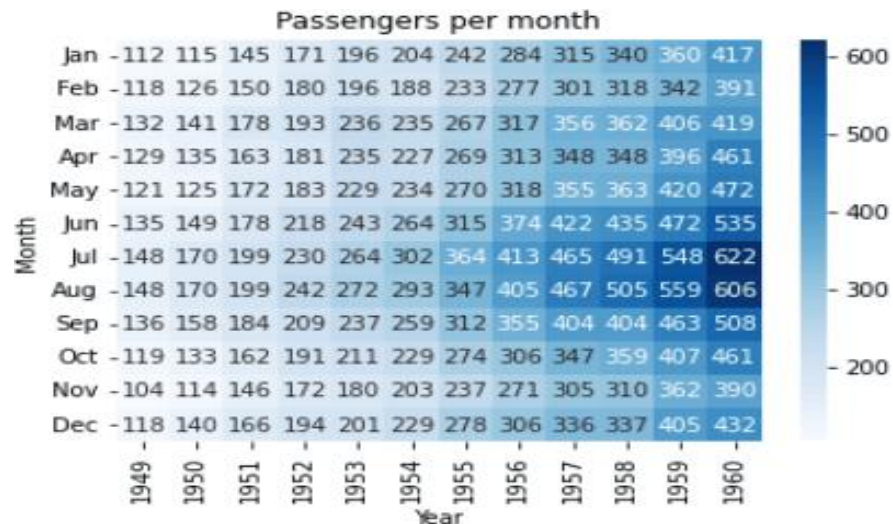


Figure 5.8 Heat Map

PairPlot

Pair plots are a type of visualization in which multiple pairwise scatter plots are displayed in a matrix format. Each scatter plot shows the relationship between two variables, while the diagonal plots show the distribution of the individual variables.

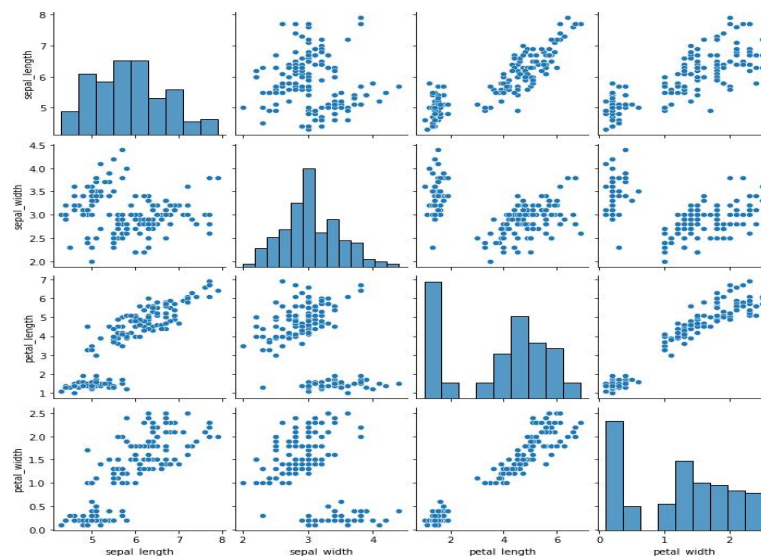


Figure 5.9 Pair Plot

5.2 INPUT DESIGN AND OUTPUT DESIGN

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs.

In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user.

Efficient and intelligent output design improves the system's relationship to help user decision- making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
2. Select methods for presenting information.
3. Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the † Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

5.3 ALGORITHMS

RCNN

RCNN (Region-based Convolutional Neural Network) is a model used for object detection in images. Here's a brief breakdown of its key components, including neurons, weights, biases, forward propagation, epochs, ReLU, and activation functions:

1. Neurons

Neurons are the basic units of a neural network that process input data. Each neuron receives input (image pixels or features), processes it, and outputs a value. Neurons are organized into layers: input layer, hidden layers, and output layer.

2. Weights and Biases

Weights: These are parameters that control the importance of each input to a neuron. During training, the weights are adjusted to minimize error and improve predictions.

Biases: Biases are additional parameters that allow the model to shift the output, helping the network learn patterns more effectively, even when all inputs are zero.

Learning: Both weights and biases are learned through the process of training, and they are updated during backpropagation to reduce the error.

3. Forward Propagation

Forward propagation is the process where the input data is passed through the network to make predictions. In RCNN

- The input image is divided into regions (Region Proposals).
- Each region is processed by a CNN, which applies convolutions, pooling, and fully connected layers to extract features.
- The final output is produced after passing through these layers, including a classifier and bounding box regressor.

4. Epoch

An **epoch** is one complete cycle through the entire training dataset. In each epoch, the model processes all training examples, makes predictions, and adjusts the weights and biases.

Multiple epochs are necessary for the model to learn from the data and improve its performance.

5. ReLU (Rectified Linear Unit)

ReLU is an activation function used in deep learning models, including RCNN.

It outputs the input if it's positive and zero if it's negative. Mathematically, it's expressed as:

$$\text{ReLU}(x) = \max(0, x)$$

Purpose: ReLU introduces non-linearity into the model, allowing it to learn complex

patterns. It also helps prevent the vanishing gradient problem, which is common in deep networks.

6. Activation Function

Activation functions decide whether a neuron should be activated (i.e., produce an output). They introduce non-linearity, which helps the network model more complex relationships.

Common Activation Functions

ReLU: Used in most layers of RCNN for its efficiency and simplicity.

Sigmoid or Softmax: Often used in the output layer for classification tasks (to output probabilities).

Summary of the Workflow

1. Forward Propagation

- The image is divided into region proposals.
- Each region is passed through a CNN for feature extraction.
- The extracted features are processed by fully connected layers.
- The output consists of object classification and bounding box predictions.

2. Training (Epochs)

- The network is trained over multiple epochs to adjust weights and biases using backpropagation.
- In each epoch, the weights are updated to minimize the error.

3. Activation

- ReLU is used as the activation function in intermediate layers for non-linearity.
- The final output layer may use a softmax activation to classify objects.

YOLO

YOLO (You Only Look Once) is a popular real-time object detection algorithm. It detects and classifies multiple objects in an image in a single pass, making it fast and efficient. Here's a brief explanation of how YOLO works:

Key Features of YOLO

1. **Single Unified Model:** Unlike traditional object detection methods that first generate region proposals and then classify each region, YOLO treats the object detection problem as a single regression problem. It simultaneously predicts bounding boxes and class probabilities for all objects in an image in one forward pass.
2. **Grid-Based Prediction:**
 - The image is divided into an $S \times SS \times SS \times S$ grid.

- Each grid cell is responsible for predicting a fixed number of bounding boxes and their corresponding confidence scores (how likely the box contains an object) and class probabilities (which object is in the box).

3. Bounding Box Prediction

- YOLO predicts the following for each bounding box
 - xxx, yyy : Coordinates of the center of the box relative to the grid cell.
 - www, hhh : Width and height of the bounding box.
 - **Confidence score**: A measure of how confident YOLO is that the box contains an object and how accurate it believes the box is.

4. Class Prediction

- YOLO also predicts the probability distribution of the classes (such as person, car, dog) for each bounding box.

5. Output

- The output of YOLO is a tensor where each cell predicts multiple bounding boxes with associated class probabilities and confidence scores.
- After making predictions, YOLO applies a **non-maximum suppression** algorithm to remove duplicate and low-confidence boxes.

Advantages of YOLO

- **Real-time Speed**: Because YOLO processes the entire image in one go, it is much faster than methods that use region proposals, making it suitable for real-time applications.
- **Global Context**: YOLO looks at the whole image during prediction, allowing it to understand the global context and avoid false positives that might arise from local methods.

Working Process

1. **Input Image**: The input image is resized to a fixed size (e.g., 416×416).
2. **Grid Division**: The image is divided into an $S \times SS \times SS \times S$ grid (e.g., $13 \times 13 \times 13 \times 13$ for smaller images).
3. **Bounding Box and Class Prediction**: Each grid cell predicts multiple bounding boxes, along with class probabilities and confidence scores.
4. **Non-Maximum Suppression**: After predictions, duplicate and low-confidence boxes are eliminated, keeping the best box for each detected object.

5.4 SAMPLE CODE

Source code

```
# USAGE

#python cctv_video.py --input linecctv.mp4 --yolo yolo-coco

import numpy as np
import argparse
import imutils
import time
import cv2
import os
import subprocess
import os, shutil

folder = 'output'
for filename in os.listdir(folder):
    file_path = os.path.join(folder, filename)
    try:
        if os.path.isfile(file_path) or os.path.islink(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)
    except Exception as e:
        print('Failed to delete %s. Reason: %s' % (file_path, e))

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", required=True,
                help="path to input video")
ap.add_argument("-o", "--output", required=False,
                help="path to output video")
ap.add_argument("-y", "--yolo", required=True,
                help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
                help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
```

```

        help="threshold when applying non-maxima suppression")
args = vars(ap.parse_args())
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n") #Changes here
# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
                             dtype="uint8")
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames() #Changes here
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers().flatten()]
# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture(args["input"])
writer = None
(W, H) = (None, None)
try:
    prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
        else cv2.CAP_PROP_FRAME_COUNT
    total = int(vs.get(prop))
    print("[INFO] {} total frames in video".format(total))
except:
    print("[INFO] could not determine # of frames in video")
    print("[INFO] no approx. completion time can be provided")
    total = -1
cnt = 0
fno = 0
# -----FRAME PART-----
counter = 0
while True:
    start1 = time.time()

```

```

(grabbed, frame) = vs.read()
if(cnt%2!=0):
    cnt+=1
    continue
fno+=1
print("Frame No:", fno)
if not grabbed:
    break
# if the frame dimensions are empty, grab them
if W is None or H is None:
    (H, W) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
    swapRB=True, crop=False)
net.setInput(blob)
start = time.time()
layerOutputs = net.forward(ln)
end = time.time()
boxes = []
confidences = []
classIDs = []
# loop over each of the layer outputs
for output in layerOutputs:
    # loop over each of the detections
    for detection in output:
        # extract the class ID and confidence (i.e., probability)
        # of the current object detection
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
        # filter out weak predictions by ensuring the detected
        # probability is greater than the minimum probability
        if confidence > args["confidence"]:
            box = detection[0:4] * np.array([W, H, W, H])
            (centerX, centerY, width, height) = box.astype("int")

```

```

x = int(centerX - (width / 2))
y = int(centerY - (height / 2))
# update our list of bounding box coordinates,
# confidences, and class IDs
boxes.append([x, y, int(width), int(height)])
confidences.append(float(confidence))
classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping
# bounding boxes
idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
                        args["threshold"])
# ensure at least one detection exists
if len(idxs) > 0:
    idArray = []
    for j in idxs.flatten():
        if classIDs[j]==2 or classIDs[j]==0:
            idArray.append(j)

    flag = 0
    for j in idArray:
        for k in idArray:
            if k==j:
                continue

            (x1, y1) = (boxes[j][0], boxes[j][1])
            (w1, h1) = (boxes[j][2], boxes[j][3])
            (x2, y2) = (x1+w1, abs(y1-h1))

            (x3, y3) = (boxes[k][0], boxes[k][1])
            (w2, h2) = (boxes[k][2], boxes[k][3])
            (x4, y4) = (x3+w2, abs(y3-h2))
            if(x1>x4 or x3>x2):
                flag = 1
                counter = 0
            if(y1 < y4 or y3 < y2):
                flag = 1
                counter = 0

```

```

        if flag == 0:
            counter+=1
            break

    if flag == 0:
        break

# loop over the indexes we are keeping
if(counter==4):
    print("CRASH ALERT")
    cv2.imwrite("output/Crash{}.jpg".format(fno), frame)
    counter=0

for i in idxs.flatten():
    # extract the bounding box coordinates
    (x, y) = (boxes[i][0], boxes[i][1])
    (w, h) = (boxes[i][2], boxes[i][3])
    # draw a bounding box rectangle and label on the frame
    color = [int(c) for c in COLORS[classIDs[i]]]
    cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
    text = "{}: {:.4f}".format(LABELS[classIDs[i]],
                               confidences[i])
    cv2.putText(frame, text, (x, y - 5),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    print(f'Box[{i}]: {x} {y} {w} {h} Labels[{i}]: {LABELS[classIDs[i]]}
classIDs[{i}]: {classIDs[i]} AveragePrecision[{i}]: {confidences[i]}')
    end1 = time.time()
    cv2.imwrite("output/frame{}.jpg".format(fno), frame)
    print("Complete time for algorithm", (end1-start1))
    cnt+=1

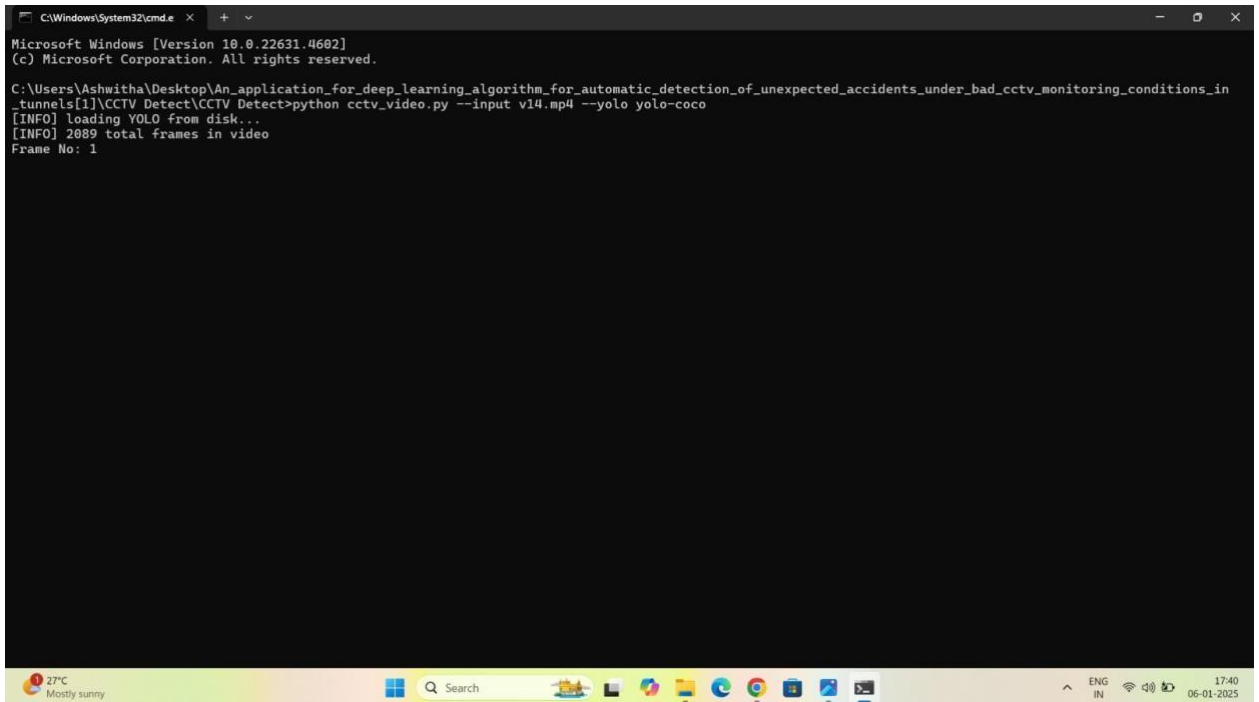
print("[INFO] cleaning up...") print("[Msg]
Accidents frames analysed") vs.release()

program = "wwd_analysis.py "+args["input"]+" TRUE"
print("[INFO] Wrong Driving Analysiss Started")
subprocess.call("python "+program)
print("[INFO] Making All Frames into a video File")
subprocess.call("python playCrash.py")

```

```
print("[INFO] Video File Playing Started")  
subprocess.call("python PlayVideo.py")
```

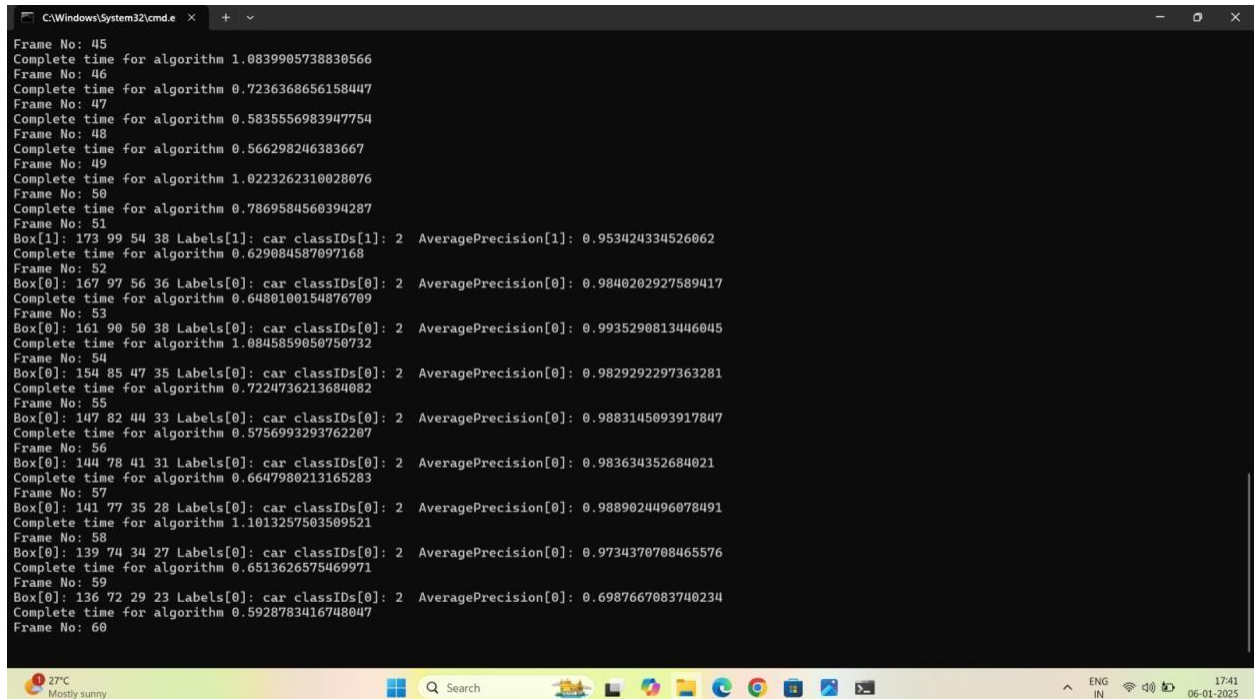
5.5 OUTPUT SCREEN



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ashwitha\Desktop\An_application_for_deep_learning_algorithm_for_automatic_detection_of_unexpected_accidents_under_bad_cctv_monitoring_conditions_in_tunnels[1]\CCTV Detect\CCTV Detect>python cctv_video.py --input v14.mp4 --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 2089 total frames in video
Frame No: 1
```

Fig 5.5.1 Input for the code execution



```
C:\Windows\System32\cmd.exe
Frame No: 45
Complete time for algorithm 1.0839905738830566
Frame No: 46
Complete time for algorithm 0.7236368656158447
Frame No: 47
Complete time for algorithm 0.5835556983947754
Frame No: 48
Complete time for algorithm 0.566298246383667
Frame No: 49
Complete time for algorithm 1.0223262310028076
Frame No: 50
Complete time for algorithm 0.7869584560394287
Frame No: 51
Box[1]: 173 99 54 38 Labels[1]: car classIDs[1]: 2 AveragePrecision[1]: 0.953424334526062
Complete time for algorithm 0.629084587097168
Frame No: 52
Box[0]: 167 97 56 36 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9840202927589417
Complete time for algorithm 0.6480100154076709
Frame No: 53
Box[0]: 161 90 50 38 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9935290813446045
Complete time for algorithm 1.0845859050750732
Frame No: 54
Box[0]: 154 85 47 35 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9829292297363281
Complete time for algorithm 0.7224736213684082
Frame No: 55
Box[0]: 147 82 44 33 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9883145093917847
Complete time for algorithm 0.5756993293762207
Frame No: 56
Box[0]: 144 78 41 31 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.983634352684021
Complete time for algorithm 0.6647980213165283
Frame No: 57
Box[0]: 141 77 35 28 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9889024496078491
Complete time for algorithm 1.1013257503509521
Frame No: 58
Box[0]: 139 74 34 27 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9734370708465576
Complete time for algorithm 0.6513626575469971
Frame No: 59
Box[0]: 136 72 29 23 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.6987667083740234
Complete time for algorithm 0.5928783416748047
Frame No: 60
```

Fig 5.5.2 Figure showing the frames being loading


```

C:\Windows\System32\cmd.exe
Box[1]: 116 58 20 14 Labels[1]: car classIDs[1]: 2 AveragePrecision[1]: 0.564228355884552
Complete time for algorithm 0.8899145889282227
Frame No: 1036
Box[0]: 86 103 102 32 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9627473950386047
Box[1]: 115 58 21 14 Labels[1]: car classIDs[1]: 2 AveragePrecision[1]: 0.6681947112083435
Complete time for algorithm 0.645012378692627
Frame No: 1037
Box[0]: 86 102 102 33 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9759469628334045
Box[2]: 114 56 20 14 Labels[2]: car classIDs[2]: 2 AveragePrecision[2]: 0.856322705745697
Box[1]: 86 48 8 9 Labels[1]: car classIDs[1]: 2 AveragePrecision[1]: 0.6711363196372986
Complete time for algorithm 0.8644027709960938
Frame No: 1038
Box[0]: 88 101 100 33 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9452360272407532
Box[2]: 114 56 20 13 Labels[2]: car classIDs[2]: 2 AveragePrecision[2]: 0.6682839499282837
Box[1]: 86 48 8 9 Labels[1]: car classIDs[1]: 2 AveragePrecision[1]: 0.6569419679641724
Complete time for algorithm 1.0874197483862744
Frame No: 1039
Box[0]: 87 103 102 32 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9221706390380859
Complete time for algorithm 1.1222972869873047
Frame No: 1040
Box[0]: 85 102 103 33 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9158577919006348
Complete time for algorithm 1.1495976448059082
Frame No: 1041
Box[0]: 85 102 103 33 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9733548164367676
Complete time for algorithm 1.1284682750701904
Frame No: 1042
Box[0]: 86 102 102 34 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.971811056137085
Complete time for algorithm 1.0853650569915771
Frame No: 1043
Box[0]: 84 102 103 34 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.9273241758346558
Complete time for algorithm 1.1005034446716309
Frame No: 1044
Box[0]: 86 102 101 34 Labels[0]: car classIDs[0]: 2 AveragePrecision[0]: 0.8436521291732788
Complete time for algorithm 0.7674245834358586
Frame No: 1045
Complete time for algorithm 0.643242359161377
Frame No: 1046
[INFO] cleaning up...
[Msg] Accidents frames analysed
[INFO] Wrong Driving Analysis Started

```

Fig 5.5.3 frames being extracted from the images

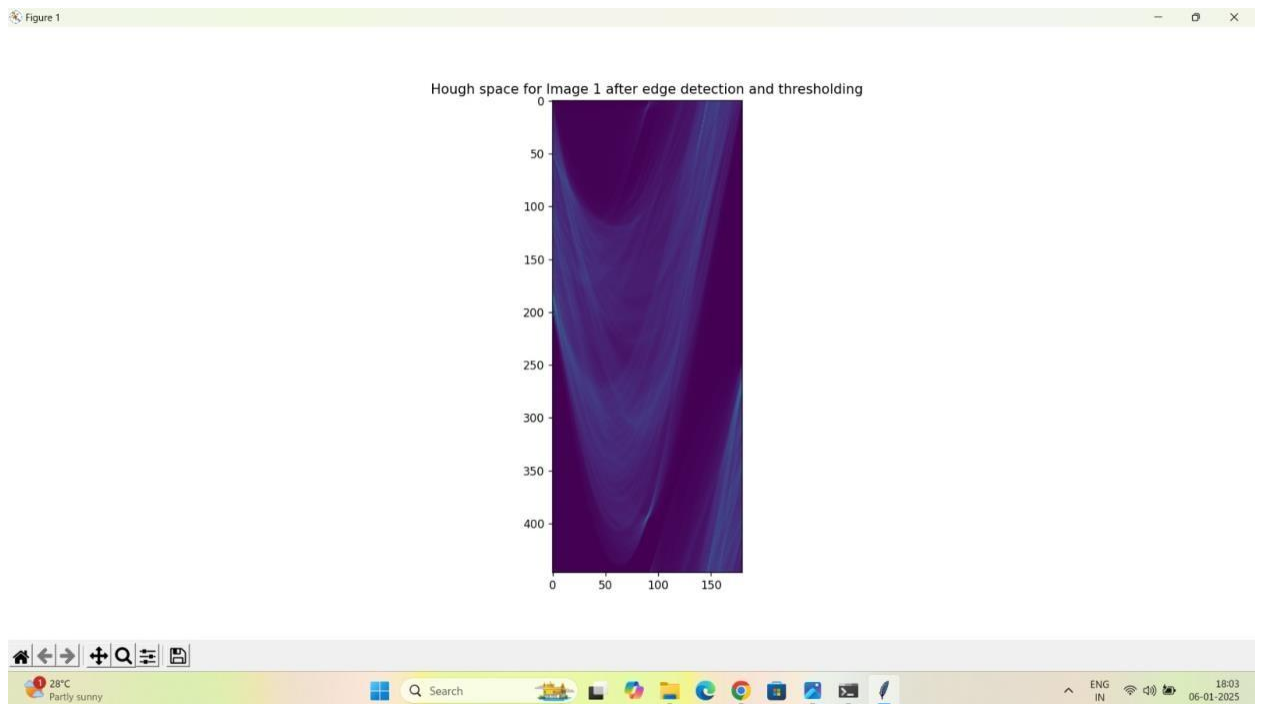


Fig 5.5.4 hough graph after edge detection

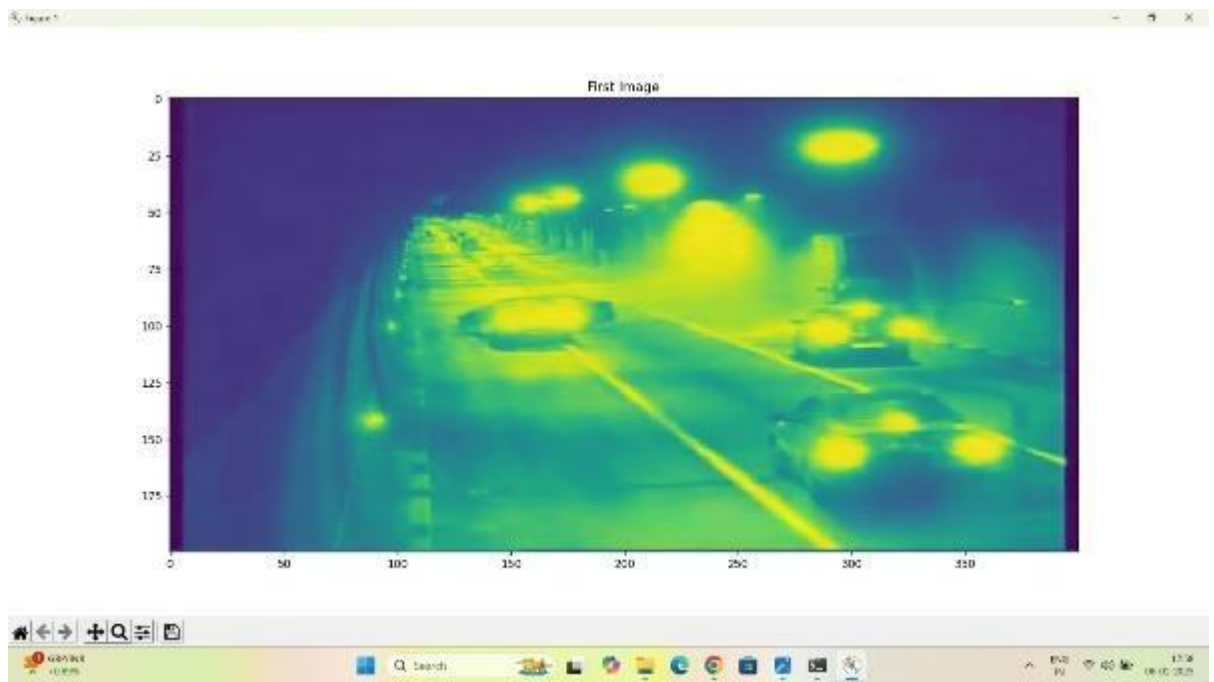


Fig 5.5.5 First Image In Accident Detection

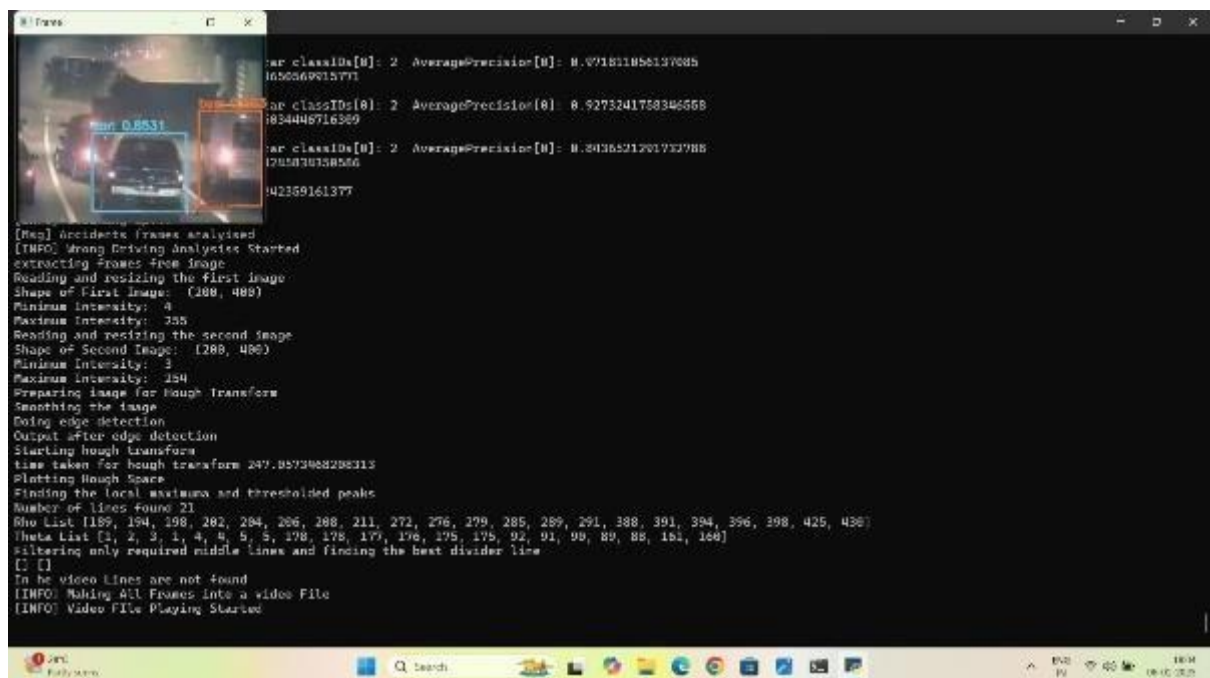


Fig 5.5.6 Output Image

CHAPTER 6

TESTING

6.1 INTRODUCTION OF TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTS

6.2.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application

.it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 User Acceptance testing

User Acceptance Testing (UAT) is a critical phase in software development where the intended end-users rigorously test the system to ensure it meets their specific requirements and functions as expected in a real-world environment.

This final stage of testing involves real users interacting with the software, evaluating its usability, functionality, and performance. UAT aims to verify that the software delivers the intended value to the business by aligning with user needs and expectations. By identifying and addressing potential issues before deployment, UAT helps minimize risks, reduce costs, and enhance user satisfaction. Different types of UAT include alpha testing (internal testing), beta testing (external user testing), and contract acceptance testing (CAT). Successful UAT builds confidence in the software's readiness for deployment and contributes to the overall success of the project.

6.2.4 Output Testing

Output testing is a critical aspect of software testing that focuses on verifying the accuracy, completeness, and consistency of the results produced by the software under test. This type of testing involves comparing the actual output of the software with the expected output based on predefined test cases. By meticulously examining the output, testers can identify and rectify errors, inconsistencies, or unexpected behaviours. Output testing encompasses various techniques such as black-box testing, white-box testing, and regression testing. It plays a crucial role in ensuring the quality and reliability of the software, ultimately enhancing user satisfaction and confidence.

6.2.5 Validation Testing

Validation testing is a critical phase in the software development lifecycle that focuses on determining whether the developed software meets the specified requirements and fulfills the intended purpose. It involves testing the software in a real-world environment or under conditions that simulate real-world usage. Validation testing aims to ensure that the software is fit for its intended use and delivers the expected value to the end-users. This type of testing often involves user acceptance testing (UAT), where end-users evaluate the software's usability, functionality, and performance. By conducting thorough validation testing, developers can identify and address potential issues before deployment, reducing risks, improving software quality, and enhancing user satisfaction.

6.3 Various Test case scenarios considered

1. Unit Testing

Test Case 1: Load YOLO Model

Input: Valid paths for **weightsPath** and **configPath**.

Expected Output: The model loads without errors.

Test Case 2: Process Single Frame

Input: A sample frame image and a mock YOLO model.

Expected Output: A list of bounding boxes, confidences, and class IDs.

Test Case 3: File Deletion Logic

Input: A directory with multiple files and subdirectories.

Expected Output: The directory is empty after execution.

2. Integration Testing

Test Case 1: End-to-End Video Processing

Input: A sample video file.

Expected Output: Output frames with bounding boxes saved in the specified output directory.

Test Case 2: Subprocess Execution

Input: A valid input video file and the necessary scripts.

Expected Output: The subprocess completes without errors, and the expected output files are generated.

3. User Acceptance Testing (UAT)

Test Case 1: User Feedback on Crash Alerts

Input: A video that contains crash scenarios.

Expected Output: Users find the alerts clear and actionable.

Test Case 2: Usability Testing

Input: A sample user interface (if applicable).

Expected Output: Users can complete tasks without confusion.

4. Output Testing

Test Case 1: Validate Output Frames

Input: A processed video file and the corresponding output frames.

Expected Output: Each output frame has bounding boxes that match the detected objects.

Test Case 2: Check Crash Alert Images

Input: A video with crash scenarios.

Expected Output: The output directory contains images with the correct naming convention and content.

5. Validation Testing

Test Case 1: Validate Crash Detection Logic

Input: A video with known crash scenarios.

Expected Output: The application generates crash alerts for each crash detected.

Test Case 2: Confidence Threshold Testing

Input: A video with both valid and invalid detections.

Expected Output: Only valid detections are reported based on the specified confidence threshold.

TEST CASES

Sno	Test Case	Precondition	Expected Results	Actual Results	Post Condition	Pass/Fail
1	Load YOLO Model	YOLO model files and weights are present	Model is successfully loaded without errors	Model loaded successfully with no errors.	Model is ready for use	Pass
2	Process Single Frame	YOLO model is loaded	The frame is processed, and objects are detected with confidence scores.	Frame processed; objects detected with confidence scores above threshold.	Processed frame is available	Pass
3	End-to-End Video Processing	YOLO model is loaded, video file is accessible	Entire video is processed frame by frame, generating detection outputs.	Video processed successfully; detections are accurate and match expected output.	Processed video output generated	Pass
4	User Feedback on Crash Alerts	YOLO model and crash alert mechanism are active	User receives a notification for detected crash scenarios.	User received timely crash alerts and confirmed their accuracy.	User feedback is captured	Pass
5	Validate Output Frames	YOLO model is loaded, and test frames are available	Output frames match expected detection results, including bounding boxes and labels.	Output frames matched expected results; all objects correctly identified.	Frame validation complete	Pass
6	Check Crash Alerts Images	YOLO crash alert system is active	Images of crashes are saved with proper labelling for review.	Crash alert images saved successfully with accurate labels.	Crash alert images stored	Pass

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

A basic framework has been established in this study for sensing vehicular crashes and transmitting an emergency notice to the nearest Accident and Emergency Department. Local characteristics such as trajectory collision, velocity estimation, and their different anomalies are used to build this framework. The major goal of this project is to reduce the numbers killed in car accidents caused by excessive speeding, to enhance public safety, and create a better system for regulating traffic on the roadways. The technology is cost-effective, scalable, and rapid, and it can be simply integrated into existing live surveillance systems. Our system's dependability is enhanced using numerous factors to assess the risk of an accident. One of the study's shortcomings is its function in high-traffic areas and in low-light settings, which causes problems with vehicle identification and tracking inaccuracies, which will be solved in future studies. Large impediments in the way of the cameras' range of vision may also hinder vehicle tracking and, as a result, collision detection. The suggested framework can accurately detect accidents with a 71 percent rate of detection and a 0.53 percent rate of False Alarm. The experimental findings are encouraging and illustrate the framework's effectiveness.

7.2 FUTURE SCOPE

The fire object detection performance of the deep learning object detection network should be improved by securing the Fire image later. Although the ODTS can be applied as an example of a Tunnel CCTV Accident Detection System, it is also used to fields that need to monitor the dynamic movement of a specific object such as vehicle speed estimation or illegal parking monitoring will be possible. To increase the reliability of the system, it is necessary to secure various images and to secure Fire and Person objects. Besides, through the application and continuous monitoring of the tunnel management site, the reliability of the system could be improved.

CHAPTER 8

REFERENCES

- [1] “Road traffic injuries and deaths global problem,” <https://www.cdc.gov/features/globalroadsafety/index.html>.
- [2] K. He, G. Gkioxari, P. Dollr, and R. Girshick, “Mask r-cnn,” in Proc. of IEEE International Conference on Computer Vision (ICCV), Oct 2017.
- [3] “Object detection for dummies part 3: R-cnn family,” <https://lilianweng.github.io/lil-log/assets/images/rcnn-family-summary.png>.
- [4] J. C. Nascimento, A. J. Abrantes, and J. S. Marques, “An algorithm for centroid-based tracking of moving objects,” in Proc. of IEEE International Conference on Acoustics,
- [5] R. J. Blissett, C. Stennett, and R. M. Day, “Digital cctv processing in traffic management,” in Proc. of IEE Colloquium on Electronics in Handling Road Capacity Demand, Nov 1993, pp. 12/1-12/5.
- [6] F. Baselice, G. Ferraioli, G. Matuozzo, V. Pascazio, and G. Schirinzi, “3d automotive imaging radar for transportation systems monitoring,” in Proc. of IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems, Sep 2014, pp. 1–5.
- [7] https://www.researchgate.net/publication/304808065_Evaluation_of_Roadside_Wrong-Way_Warning_Systems_with_Different_Types_of_Sensors
- [8]. A. Franklin, “The future of cctv in road monitoring,” in Proc. of IEE Seminar on CCTV and Road Surveillance, May 1999, pp. 10/1–10/4.
- [9] Z. Hui, X. Yaohua, M. Lu, and F. Jiansheng, “Vision-based real-time traffic accident detection,” in Proc. of World Congress on Intelligent Control and Automation, June 2014, pp. 1035–1038.
- [10] <https://www.geographyandyou.com/disaster/disaster-events/track-troubles-road-accidents-in-india/>

