REAL TIME OBJECT DETECTION SYSTEM

MINOR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

Submitted By:

Aditya Kumar (2104057)

Akash Kumar(2104062)

Anmol Kumar(2104070)

Submitted To.:

Prof. Jasbir Singh Saini

Department of Computer Science and Engineering

Guru Nanak Dev Engineering College

Ludhiana. 141006

ABSTRACT

This project focuses on real-time object detection using advanced computer vision techniques. With the rapid advancement in technology, real-time object detection has become a crucial component in various fields including surveillance, autonomous vehicles, and augmented reality. This project employs state-of-the-art deep learning models such as YOLOv8 (You Only Look Once) for efficient and accurate detection of objects in live video streams.

The project involves several key phases including data collection, preprocessing, model training, and deployment. An extensive dataset comprising diverse object classes is utilised for training the detection model. Preprocessing techniques such as image augmentation and annotation and normalisation are applied to enhance the model's robustness and generalisation capabilities.

During the Model Training Phase, the YOLO architecture is trained on the annotated dataset using powerful GPUs to accelerate the training process. The trained model is optimised for real-time performance, ensuring fast and accurate object detection in live video feeds.

For Deployment, the trained model is integrated into a real-time system capable of processing video streams from various sources such as cameras or video files. enabling seamless object detection in real-world scenarios.

The performance of the real-time object detection system is evaluated using standard metrics such as mean Average Precision (mAP) and processing speed.

ACKNOWLEDGEMENT

We are highly grateful to Dr. Sehijpal Singh , Principal, Guru Nanak Dev Engineering College (GNDEC), Ludhiana, for providing this opportunity to carry out the minor project work on Real Time Object Detection..

The constant guidance and encouragement received from Dr. Kiran Jyoti H.O.D. CSE Department, GNDEC Ludhiana has been of great help in carrying out the project work and is acknowledged with reverential thanks.

We would like to express a deep sense of gratitude and thanks profusely to Project Guide Er. Jasbir Singh Saini , without his wise counsel and able guidance, it would have been impossible to complete the project in this manner.

We express gratitude to other faculty members of the computer science and engineering department of GNDEC for their intellectual support throughout the course of this work.

Finally, We are indebted to all whosoever have contributed in this report work.

Aditya Kumar

Akash Kumar

Anmol Kumar

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

INTRODUCTION TO PROJECT

Most of the cases, Humans perform work by observing their environment and perform action accordingly, and the observation of the environment is done using their eyes, meaning that they take input through their eyes and this observed data is sent to their mind for processing and the process is done through their mind instantly in real time. The brain observes this data and detects different objects and sends commands to our body parts to do action.

In our project we aim to make the same system which can perform that work in real time just like the human brain. Here our camera will work as an eye for our system and we will use a neural network's different techniques to make a system which can detect different objects in our scene in a real time instant process.

There are several techniques which we use to build such project such as :-

- Convolutional Neural Networks (CNNs): These are deep learning models that consist of multiple layers of neural features that learn features from image data.
- Region based Methods: These are methods that divide an image into multiple regions and apply CNNs to each region to detect objects.
- Single shot Methods: These are methods that perform object detection in one pass over image, using a single CNN. Some examples of single shot methods are You Only Look Once (YOLO) and Single Shot Multibox Detector (SSD).

There are many fields where we need such kinds of systems which can be used to detect object like Video surveillance, Crowd counting, Anomaly detection, Self -Driving Cars and many more. And most importantly we can use these techniques in fields of robotics where robots can detect different objects.

PROJECT CATEGORY

Our project falls under the category of "System Development" with a strong emphasis on research. Our team is actively developing a real-time object detection system using the YOLOv8 model. This involves the creation and implementation of software that can detect objects in real-time, a crucial application in various fields such as surveillance and robotics. Additionally, the project involves substantial research efforts as the team collects and utilises thousands of images to train the model effectively. This research aspect includes fine-tuning the YOLOv8 model, optimising its performance, for object detection. Thus, the project blends the practicality of system development with the depth of research, making it significant in advancing the capabilities of real-time object detection systems.

OBJECTIVES

1. To Collect the Data from publicly available resources.
2. To develop a real time object detection system using YOLO and integrate its visualisation using Deep Learning.
3. Validation and Testing.

PROBLEM FORMULATION

To develop a real-time object detection system that can identify specific objects of interest within Live Video streams with high accuracy and low latency. The system should be adaptable to different environments and object types.

IDENTIFICATION/REORGANISATION OF NEED

Identification:

- Object Detection Accuracy: Improved accuracy in detecting objects in real-time. This involves identifying instances where the current system may not accurately detect objects or may have false positives/negatives.
- Real-time Performance: Understanding the requirement for the system to perform detection swiftly, ensuring it meets the criteria for real-time processing. This includes identifying the desired frames per second (FPS) for detection.

Reorganisation:

- Prioritise Objects: Reorganising the needs to prioritise certain objects based on their importance. For example, in a surveillance application, detecting people and vehicles might be more critical than detecting animals or static objects.

EXISTING SYSTEM

1. **YOLO(You only look Once)**

   YOLO is a popular real-time object detection system known for its speed and accuracy.It processes the entire image in one pass through a convolutional neural network (CNN), making it very fast.YOLO divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell.Different versions of YOLO, such as YOLOv3 and YOLOv4, have been released, each improving upon the previous one in terms of accuracy and speed.

2. **SSD (Single Shot Multibox Detector)**

   SSD is another real-time object detection system that offers a good balance between speed and accuracy.It generates bounding box predictions and class probabilities using multiple layers with different scales and aspect ratios.SSD is capable of detecting objects at multiple scales within a single shot, making it efficient for real-time applications.

3. **Fastest R-CNN (Region based Convolutional Neural Network)**

Faster R-CNN is a two-stage object detection system that achieves high accuracy but may be slower than YOLO and SSD.It first generates region proposals using a region proposal network (RPN) and then refines these proposals using a CNN-based classifier.While Faster R-CNN may not be as fast as YOLO or SSD, it often achieves better accuracy, especially for detecting small objects.

4. **Mask R-CNN**

   Mask R-CNN extends Faster R-CNN by adding a branch for predicting segmentation masks in addition to bounding boxes and class labels.It is capable of segmenting objects at pixel-level accuracy while still maintaining real-time performance for many applications.

PROPOSED SYSTEM

The proposed system addresses the need for improved object detection accuracy by using the YOLOv8 model with a higher Mean Average Precision (mAP) value. This choice ensures that the system surpasses the limitations of the previous system, offering more precision and reliability. Additionally, the system trained and tested on extra annotated images, prioritising critical objects. This approach enriches the training dataset, overcoming previous challenges such as struggles with certain object classes or variations in lighting conditions. Real-time performance is a key focus, with the system designed to meet and exceed the required frames per second (FPS) for detection, leveraging the YOLOv8 model known for its speed and accuracy. The reorganisation of needs includes prioritising critical objects like people and vehicles, ensuring they are detected with higher accuracy and priority compared to less critical objects. Furthermore, the system's versatility adapts to various environments such as indoor, outdoor, low-light, or crowded settings, surpassing the limitations of the previous system. Addressing scalability needs, the system efficiently handles a larger number of objects and cameras for detection.

UNIQUE FEATURES OF THE SYSTEM

1. **Speed and Efficiency:** YOLO is renowned for its speed and efficiency. It processes the entire image in one pass through the neural network, enabling real-time performance even on devices with limited computational resources. This speed is crucial for applications requiring rapid object detection, such as autonomous vehicles or video surveillance systems.
2. **Single-stage Detection:** YOLO is a single-stage object detection system, meaning it predicts bounding boxes and class probabilities directly from the input image without requiring multiple stages or complex post-processing. This simplicity contributes to its speed and ease of implementation.
3. **Accuracy:** While YOLO sacrifices some accuracy compared to two-stage detectors like Faster R-CNN, its performance is still impressive, especially considering its speed. YOLO strikes a good balance between accuracy and efficiency, making it suitable for a wide range of real-time applications.
4. **Versatility:** YOLO is versatile and can detect a wide variety of objects across different scales and aspect ratios. It handles overlapping objects well and is robust to occlusions, making it suitable for diverse scenarios such as crowded scenes or complex environments.
5. **Availability:** YOLO implementations are readily available in popular deep learning frameworks such as TensorFlow and PyTorch. Pre-trained models and training scripts are also widely available, simplifying the development and deployment process for real-time object detection application.

CHAPTER 2

REQUIREMENTS ANALYSIS AND SYSTEM SPECIFICATION

FEASIBILITY STUDY

1. **Technical Feasibility**
    a. Technology Assessment :- Evaluate the available technologies and algorithms for real-time object detection using YOLO. Assess their suitability based on factors like accuracy, speed, and resource requirements.
    b. Hardware and Software Requirements :- Determine the hardware (e.g., GPUs, CPUs) and software (e.g., deep learning frameworks,python libraries) needed to develop and deploy the object detection system.
    c. Data Requirements :- Assess the availability and quality of data required for training the object detection model. Consider factors such as dataset size, diversity, and annotation requirements.
2. **Economical Feasibility**
    a. Cost Estimation: Estimated the costs associated with developing, deploying, and maintaining the object detection system. This includes costs for hardware, software, data acquisition and annotation from online sources.
    b. Revenue Projection: Estimate potential revenue streams from the sale or licensing of the object detection system, as well as any additional services or products that could be offered.
3. **Operational Feasibility**
    a. Implementation Plan: Developed a detailed plan outlining the steps required to implement the real-time object detection system, including timeline, milestones, and resource allocation.

b. Integration: Assess the feasibility of integrating the object detection system with existing hardware, software, or infrastructure, such as surveillance cameras, IoT devices, or cloud platforms.

c. Scalability and Maintenance: The scalability of the system to handle increasing data volumes or user demands. Evaluate the feasibility of maintaining and updating the system over time to ensure optimal performance and reliability.

SOFTWARE REQUIREMENT SPECIFICATION

Data Requirement

- Data Sources: Specify sources for training data, including image datasets with annotated bounding boxes for object detection.
- Data Format: Define the format for input data (e.g., image files, video streams) and annotation data (e.g., XML, JSON).

Functional Requirement

- Object Detection: Specify the system's ability to detect and localise objects in real-time video streams or images.
- Classification: Define the system's capability to classify detected objects into predefined categories or classes.
- Output Format: Define the format for displaying detection results (e.g., bounding boxes, object labels).

Performance Requirement

- Real-time Processing: Specify the system's requirement for real-time performance, including minimum frames per second (FPS) for object detection.
- Accuracy: Define the desired accuracy metrics (e.g., mAP(Mean Average Precision)) for object detection and classification.

Dependability Requirement

- Reliability: Specify requirements for system reliability, including uptime and error handling.
- Robustness: Define the system's ability to perform accurately under varying environmental conditions (e.g., lighting conditions).

Maintainability Requirement

- Modularity: Specify requirements for modular design to facilitate system maintenance and updates.
- Documentation: Define documentation requirements for system architecture, code, and user manuals.

Security Requirement

- Data Security: Specify measures to ensure the security and privacy of data used by the system.

Look and Feel Requirement

- User Interface: Define requirements for the user interface, including ease of use and visual design.
- Feedback: Specify requirements for providing feedback to users during object detection (e.g., visual cues, notifications).

VALIDATION

Performance Validation:

- Speed: Measure the processing speed of the object detection system to ensure it meets real-time requirements. This involves calculating the average frames per second (FPS) achieved by the system.
- Latency: Evaluate the time taken by the system to detect objects in a video stream or image. Ensure that the latency is minimal to provide timely responses.
- Scalability: Test the system's performance with varying data loads and hardware configurations to assess its scalability.

Accuracy Validation:

- Precision and Recall: Calculate precision (the ratio of correctly detected objects to total detected objects) and recall (the ratio of correctly detected objects to total ground truth objects) to evaluate the system's accuracy.
- Mean Average Precision (mAP): Compute mAP, a widely used metric for object detection systems, to measure the overall accuracy across different object classes and detection thresholds.
- Class-wise Accuracy: Assess the accuracy of object detection for each class to identify any biases or weaknesses in the system.

Reliability Validation:

- Robustness Testing: Test the system under various conditions such as changes in lighting, occlusions, and object orientations to evaluate its robustness.
- Error Handling: Assess the system's ability to handle errors and edge cases gracefully, such as cases where objects are partially visible or occluded.

User Validation:

- User Feedback: Gather feedback from end-users or domain experts to assess the system's usability, intuitiveness, and effectiveness in real-world scenarios.
- User Acceptance Testing (UAT): Conduct UAT sessions where users interact with the system and provide feedback on its performance and features.

EXPECTED HURDLES

Hardware Limitations:

- Challenge: Limited computational resources (e.g., CPU, GPU) may hinder real-time processing capabilities.
- Solution: Optimise algorithms and models for efficiency, use hardware acceleration (e.g., GPUs), or consider deploying the system on cloud-based platforms with scalable resources.

Data Quality and Quantity:

- Challenge: Acquiring annotated datasets with sufficient diversity and quality for training object detection models can be challenging.
- Solution: Augment existing datasets, collect additional data through crowd-sourcing or synthetic generation, or explore transfer learning techniques to leverage pre-trained models.

Real-world Variability:

- Challenge: Variations in lighting conditions, object orientations, occlusions, and backgrounds can affect the performance of object detection algorithms.
- Solution: Enhance model robustness through data augmentation techniques, domain adaptation, or training on diverse datasets that capture real-world variability.

Model Complexity:

- Challenge: Complex deep learning models may require extensive computational resources for training and inference, leading to longer development cycles.
- Solution: Use lightweight architectures tailored for real-time applications (e.g., MobileNet, EfficientNet), apply model compression techniques (e.g., quantization, pruning), or explore model distillation approaches.

SDLC MODEL USED

The choice of Software Development Life Cycle (SDLC) model depends on various factors including project requirements, team expertise, timeline, and flexibility for changes. For a real-time object detection project, considering the need for iterative development, continuous testing, and rapid deployment, an Agile-based SDLC model would be appropriate. Here's why:

Agile Methodology:

- Iterative and Incremental: Agile emphasises iterative development cycles, allowing for continuous improvement and refinement of the real-time object detection system.
- Flexibility: Agile enables adaptation to changing requirements and priorities, which is important for a project involving cutting-edge technology and evolving user needs.
- Customer Collaboration: Agile encourages close collaboration with stakeholders, including end-users, to gather feedback and ensure that the system meets their expectations.

Scrum Framework:

- Sprints: Scrum divides the development process into short, time-boxed iterations called sprints, typically lasting 1-4 weeks. Each sprint delivers a potential increment in improvement of the project.
- Daily Stand-ups: Daily stand-up meetings foster communication and collaboration among team members, allowing them to discuss progress, identify obstacles, and adapt plans accordingly.
- Product Backlog: The product backlog serves as a prioritized list of features and requirements, allowing the team to focus on delivering high-value functionality in each sprint.

Continuous Integration and Delivery (CI/CD):

- Automation: CI/CD practices automate the process of building, testing, and deploying the real-time object detection system, ensuring rapid and reliable delivery of updates.
- Feedback Loop: CI/CD pipelines provide immediate feedback on the quality of code changes, allowing for early detection and resolution of issues.

CHAPTER 3

SYSTEM DESIGN

DESIGN APPROACH(FUNCTION ORIENTED OR OBJECT ORIENTED)

For a real-time object detection project, an Object-Oriented Design (OOD) approach would be more appropriate than a Function-Oriented Design (FOD) approach. Due to following reasons:

Modularity and Reusability:

Object-oriented design promotes modularity by breaking down the system into smaller, manageable components (objects) that encapsulate both data and behaviour. This modularity enhances reusability, as objects can be easily reused in different parts of the system or in future projects.

Encapsulation and Abstraction:

OOD emphasises encapsulation, which hides the internal details of an object and exposes only the necessary interface. This allows for better abstraction, simplifying the design and reducing complexity.

Inheritance and Polymorphism:

Inheritance allows objects to inherit properties and behaviours from parent classes, promoting code reuse and facilitating hierarchical relationships between objects. Polymorphism enables objects to exhibit different behaviours based on their specific types, enhancing flexibility and extensibility.

Scalability and Maintainability:

Object-oriented design facilitates scalability and maintainability by providing a clear structure and organisation to the codebase. Changes or enhancements to the system can

be localised to specific objects or classes, minimising the impact on other parts of the system.

DETAIL DESIGN

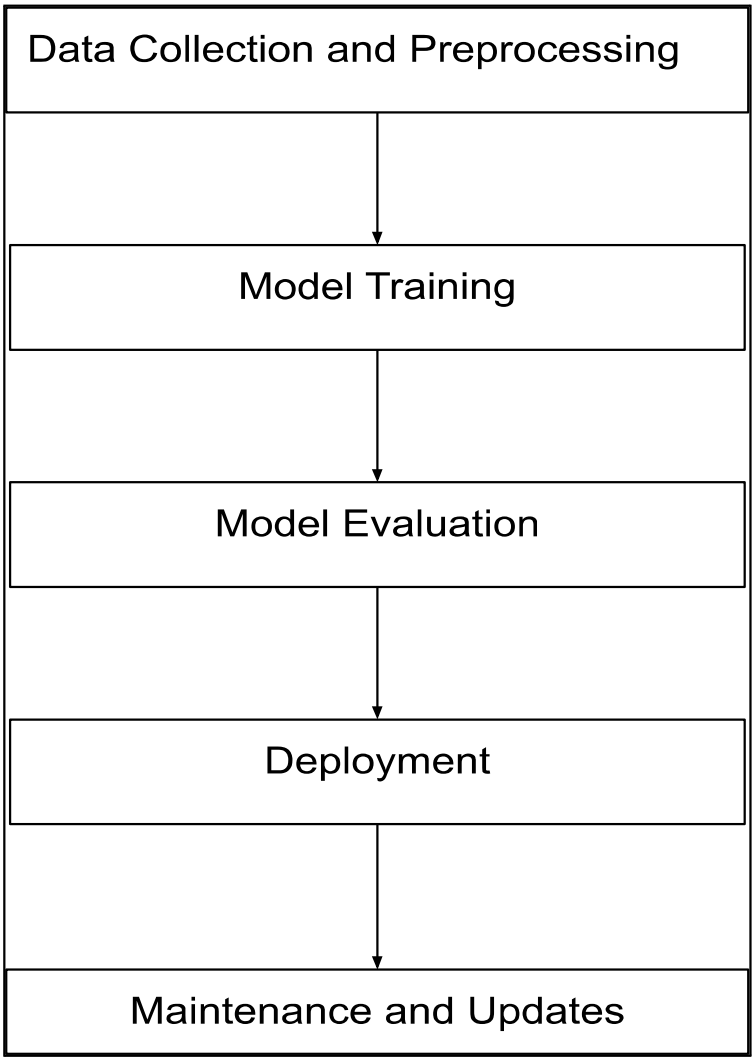

Fig 3.1 Detailed Design

Data Collection and Preparation:

- Dataset Acquisition: Start by gathering a diverse dataset relevant to your application domain. This dataset should ideally cover various scenarios, lighting conditions, and object orientations. Consider using publicly available datasets like COCO.

- Data Annotation: Annotate the images in your dataset with bounding boxes around the objects you want to detect. You can use tools like labelling, Roboflow or COCO Annotator for this task.
- Data Splitting: Divide your dataset into training, validation and testing sets. The training set is used to train the model, the validation set is used to tune hyperparameters and monitor training progress, and the testing set is used to evaluate the final model performance.

Model Training:

- Preprocessing: Preprocess the dataset images by resizing them to the input size required by the YOLOv8 model, typically 640 x 640 pixels. Perform data augmentation techniques like random rotation, flipping, cropping, and adjusting brightness and contrast to increase dataset diversity.
- Model Initialization: Initialise the YOLOv8 model with pre-trained weights obtained from a source like the Darknet framework. These pre-trained weights serve as a starting point for training and help accelerate convergence.
- Training Procedure: Train the YOLOv8 model on your annotated dataset. Use techniques like transfer learning to adapt the model to your specific domain if necessary. Train the model using a deep learning framework like TensorFlow or PyTorch, and optimise hyperparameters such as learning rate, batch size, and optimizer choice.
- Monitoring and Evaluation: Monitor the training progress using metrics like loss, mean Average Precision (mAP), and Intersection over Union (IoU). Validate the model's performance on the validation set at regular intervals to prevent overfitting. Adjust training parameters as needed based on validation results.

Model Evaluation:

- Testing Procedure: Evaluate the trained YOLOv8 model on the testing dataset to assess its performance in real-world scenarios. Compute metrics such as precision, recall, and F1-score to measure the detection accuracy. Visualise detection results to understand the model's strengths and weaknesses.
- Fine-Tuning: Fine-tune the model based on the evaluation results if necessary. This may involve retraining the model with additional data, adjusting model parameters, or exploring alternative architectures.

Deployment:

- Model Conversion: Convert the trained YOLOv8 model to a format suitable for deployment in real-time applications. Common formats include Darknet.
- Hardware Selection: Choose appropriate hardware for deployment based on factors like processing power, memory constraints, and energy efficiency. Consider using GPUs or specialised hardware accelerators like NVIDIA Jetson or Google Coral for accelerated inference.
- Integration: Integrate the YOLOv8 model into your real-time system architecture. Develop efficient data input pipelines to stream input data (e.g., video frames) to the model for inference.
- Optimization: Optimise the inference pipeline for real-time performance. This may involve techniques like model quantization, pruning, or layer fusion to reduce model size and inference latency while maintaining accuracy.
- Testing and Validation: Thoroughly test the deployed system to ensure it meets performance requirements and behaves correctly under various conditions. Validate the system's accuracy and robustness in real-world environments.

Maintenance and Updates:

- Monitoring: Monitor the deployed system for performance degradation, errors, or anomalies. Implement logging and alerting mechanisms to detect and address issues promptly.
- Continuous Improvement: Continuously update the YOLOv8 model with new data to improve accuracy and adapt to changes in the environment. Stay informed about advancements in object detection techniques and incorporate relevant updates into the system.
- Ethical Considerations: Consider ethical and privacy implications when deploying real-time object detection systems. Ensure compliance with relevant regulations and guidelines, especially concerning data privacy and consent.
- By following this detailed design process, you can develop and deploy a robust real-time object detection system using YOLOv8, incorporating the latest advancements in deep learning and computer vision techniques.

SYSTEM DESIGN

- **Level 0 DFD**

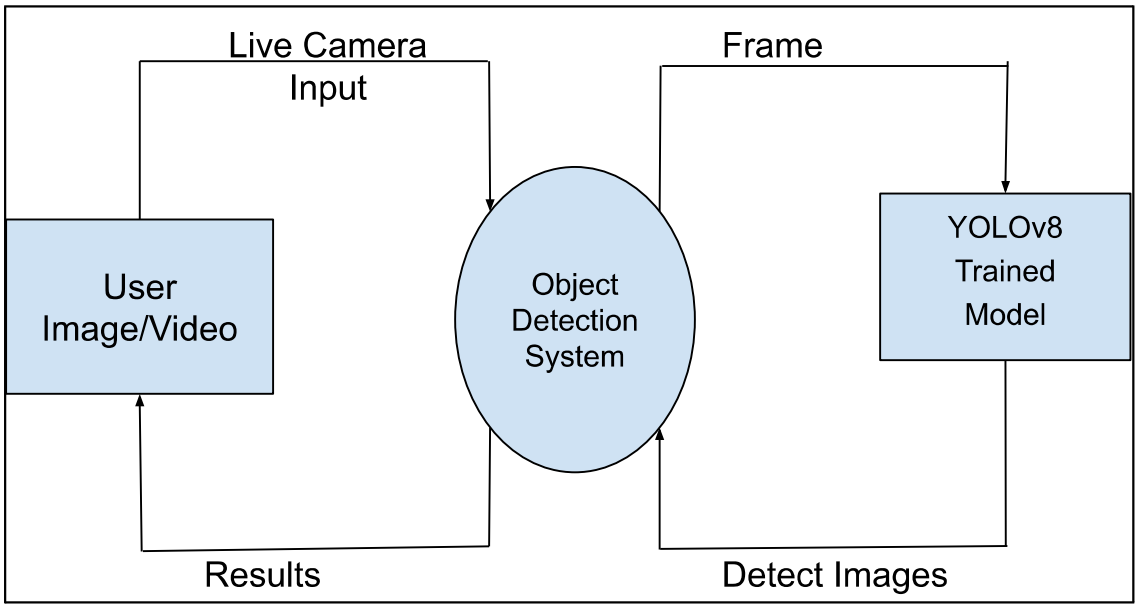

Fig 3.2 Level 0 DFD

- **Level 1 DFD**

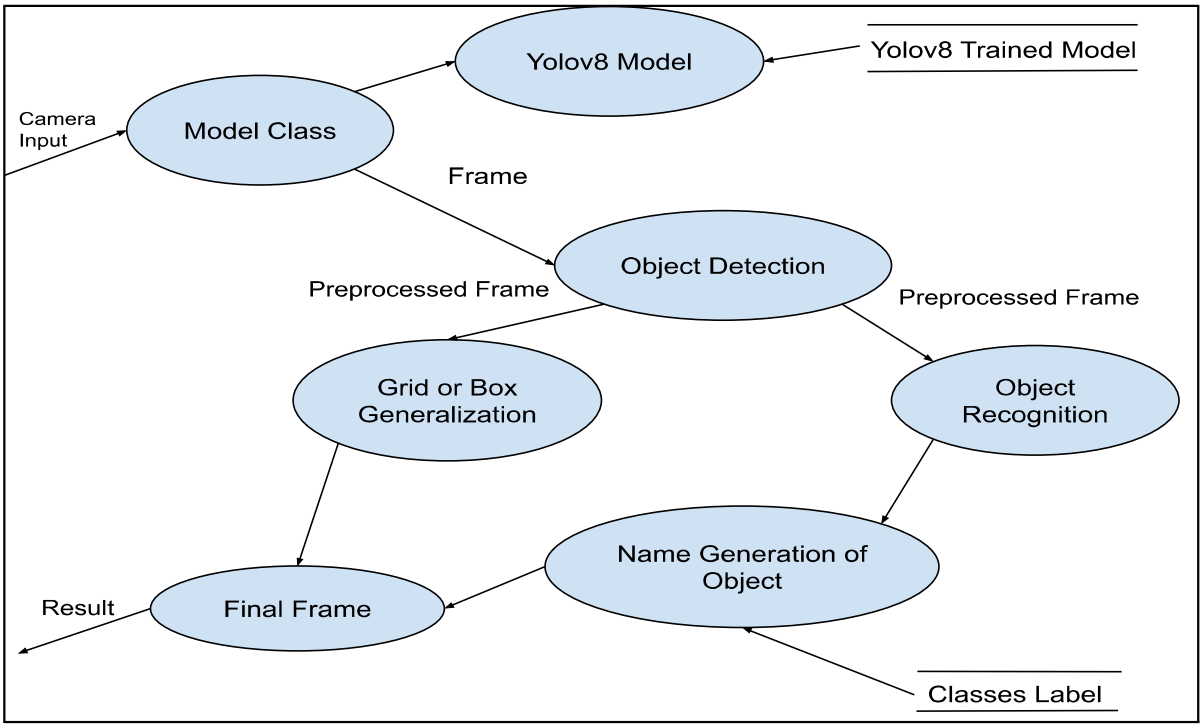

Fig 3.3 Level 1 DFD

DATA DICTIONARY

Data Element	**Description**	**Data Type**	**Constraints/Rules**
Camera Frame	A single frame extracted from a live camera input	Image	Dimension 640x640
Preprocessed Frame	Video frame after preprocessing	Image	Same dimension as input frame
Detected Object	An object Detected within a frame	Object	Bounding Box(x_min,y_min,x_ma,y_max)
Class Label	The Category or class of a detected object	String	
Confidence Score	The confidence level of the object Detected	Float	Range 0.0-1.0

Table 3.1 Data Dictionary

STRUCTURED CHART

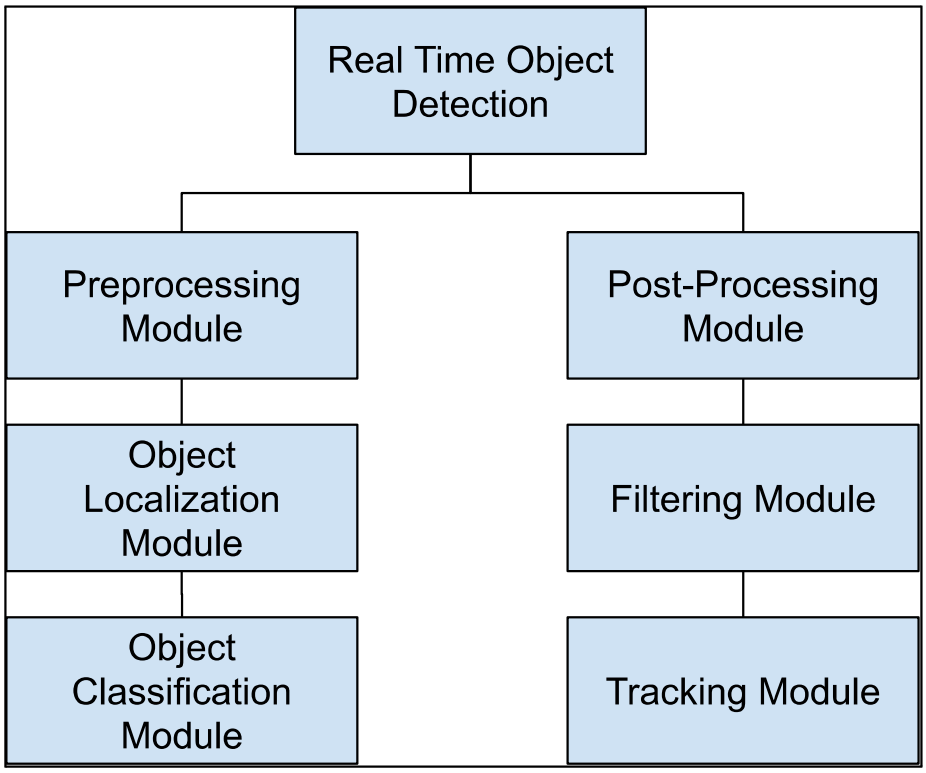

Fig 3.4 Structured Chart

UML DIAGRAM

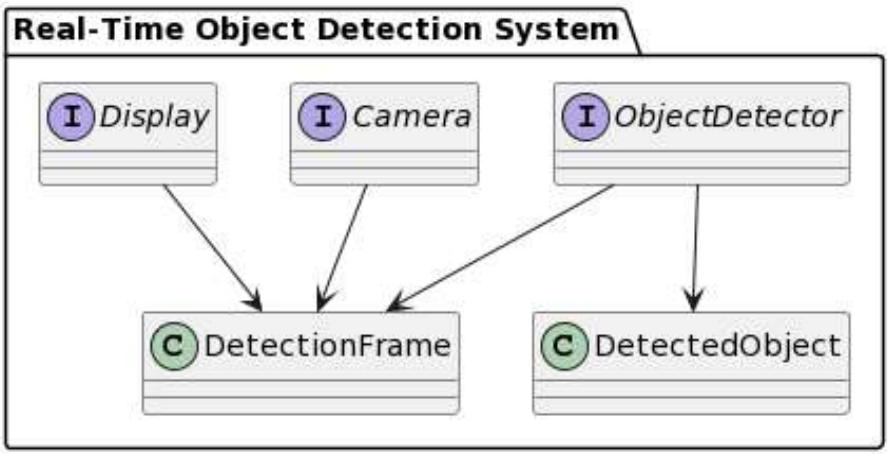

Fig 3.5 UML Diagram

ER(Entity Relationship) DIAGRAM

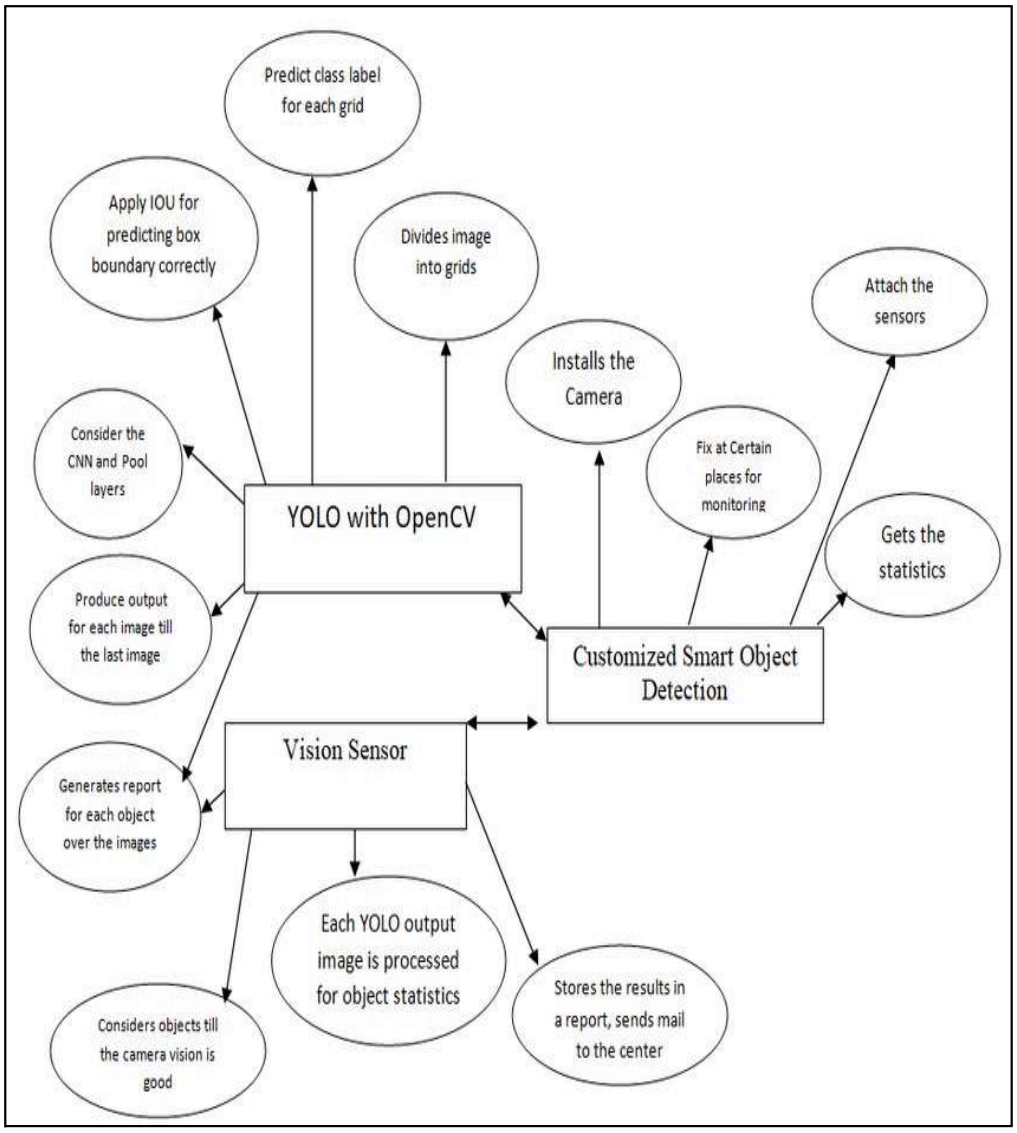

Fig 3.6 ER diagram

METHODOLOGY

Below is the generalised methodology that can be adapted for developing this real time object detection system.

1. Define Objective and Scope: As mentioned above that the scope of our project is to detect, identify and locate an object from a given video or images.
2. Data Collection and Annotation: We need to gather a diverse and representative dataset that includes images or video frames relevant with the project.
3. Select Object Detection Algorithm: We need to choose an algorithm which is most feasible with our project. Common choices include faster R-CNN,SSD,YOLO or their variants.
4. Preprocess Data: Implement data preprocessing steps to prepare the input data for the object detection model. This may include resizing images, normalising pixel values, and augmenting the dataset for increased diversity.
5. Model Training: Train the chosen object detection model using the annotated dataset. Consider using transfer learning with pre-trained models to expedite training and improve performance.
6. Optimise Model for inference: Optimize the trained model for real-time inference. This may involve model quantization, pruning, or other techniques to reduce the model size and computational requirements.
7. Choose Hardware and Software Stack: Select the appropriate hardware for running the object detection model in real-time. Consider GPUs, TPUs or specialised accelerators depending on the computational requirements. Choose a software stack, including deep learning frameworks (e.g., TensorFlow, PyTorch), for model deployment.
8. Testing and Validation: Conduct thorough testing and validation of the real-time object detection system.

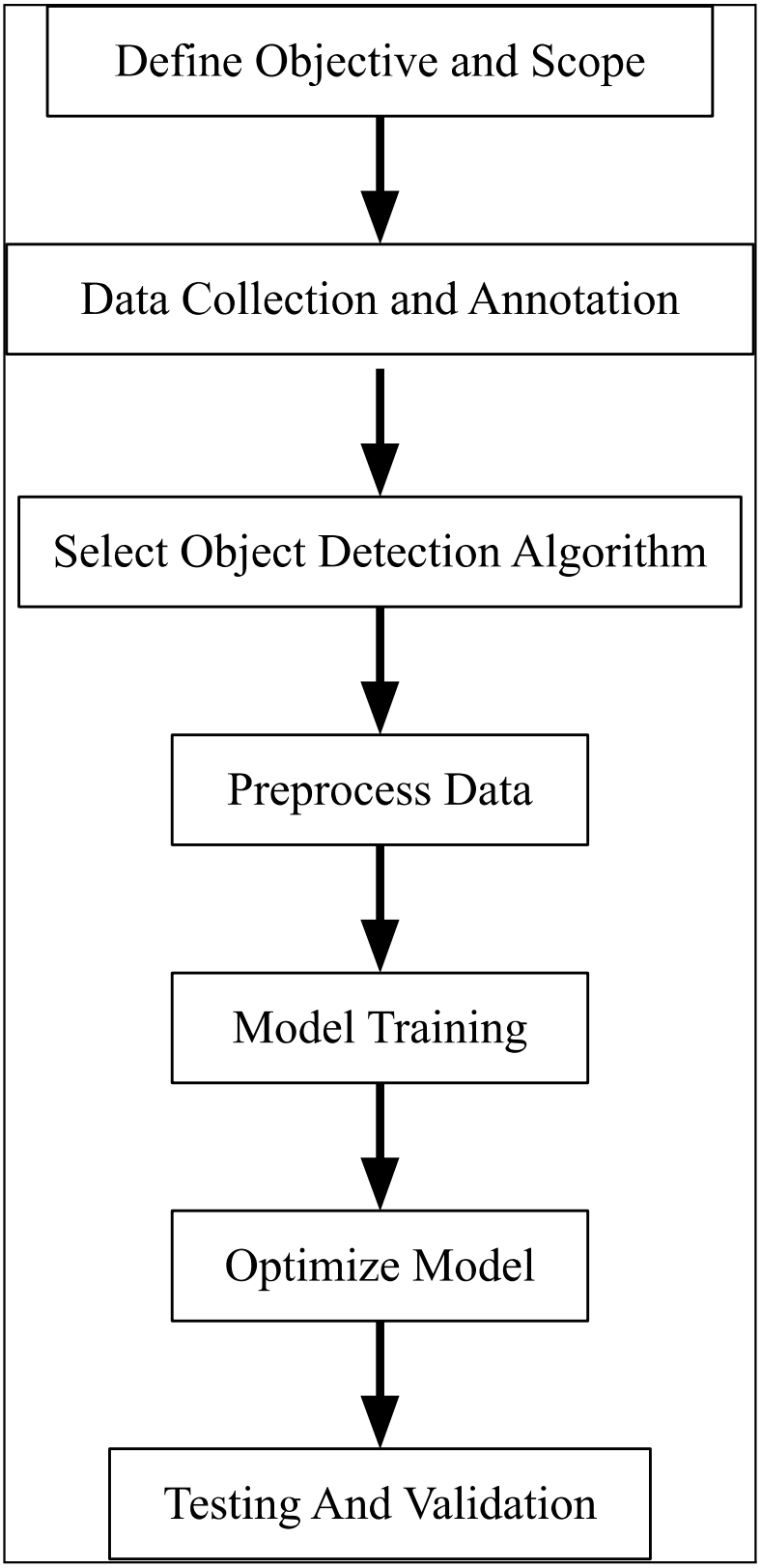

Fig 3.7 Methodology/Planning of Work

CHAPTER 4

IMPLEMENTATION, TESTING AND MAINTENANCE

INTRODUCTION TO LANGUAGE

Python is a versatile and beginner-friendly programming language known for its readability. It has clear syntax. This, along with its extensive libraries, makes Python a popular choice for tasks like data analysis, web development, and machine learning. Even automation and scripting become accessible with Python's ability to handle various file formats and interact with the system.

IDE'S

- Google collab:-Google collab is an online IDE which we have used for training our YOLO model using GPU.
- Visual studio code:-Visual Studio Code by microsoft is used for capturing image and video output from camera and feeding it to the trained YOLO model using python language.

TOOLS AND TECHNOLOGIES USED FOR IMPLEMENTATION

- Opencv:-Opencv is a python library used for computer vision tasks such as read image,video,camera input,and also apply different operations in image frame.
- Numpy:-Numpy is also a Python library tool which is used for mathematics and matrix calculation at ease.because images are stored in matrix form so this library is more useful for image processing.
- Matplotlib:-matplotlib in a python library which is used for drawing graphs or show charts or histogram.we have used matplotlib in our project to show histogram of images.

- Ultralytics:-Ultralytics is a library which is used to import the YOLOv8 version into our project to train our model.
- YOLO version 8:-We have used YOLO version 8 for training our model.

CODING STANDARD OF LANGUAGE USED

Python Coding Standards

1. PEP 8 Compliance:
    - Follow the Python Enhancement Proposal (PEP) 8 guidelines for code formatting, including indentation, line length, naming conventions, and spacing.
2. Descriptive Variable Names:
    - Use descriptive and meaningful variable names to enhance code readability and maintainability.
3. Docstrings:
    - Include docstrings for classes, functions, and modules to provide documentation and usage information.
4. Modularization:
    - Organise code into modular functions and classes, following the principles of encapsulation and abstraction.
5. Exception Handling:
    - Implement robust error handling and exception handling mechanisms to gracefully handle unexpected errors and edge cases.
6. Import Statements:
    - Follow best practices for import statements, including organising imports alphabetically and grouping them by standard library, third-party libraries, and local imports.
7. Code Comments:

- Include inline comments to explain complex logic, algorithms, or non-obvious code sections.

PROJECT SCHEDULING

Gantt Charts

A Gantt chart is a visual representation of a project schedule that shows tasks, milestones, and dependencies over time. It provides a clear and comprehensive overview of the project timeline, allowing project managers and team members to understand the sequence of activities, deadlines, and resource allocation.

	Name	Duration	Start	Finish
1	Define Objective and Scope	3 days	15/1/24 8:00 AM	17/1/24 5:00 PM
2	Data Collection	7 days	18/1/24 8:00 AM	26/1/24 5:00 PM
3	Data Annotation	8 days?	27/1/24 8:00 AM	7/2/24 5:00 PM
4	Algorithm Selection	6 days?	8/2/24 8:00 AM	15/2/24 5:00 PM
5	Data Preprcessing	6 days?	22/2/24 8:00 AM	29/2/24 5:00 PM
6	ModelTraining	11 days?	1/3/24 8:00 AM	15/3/24 5:00 PM
7	Testing	3 days?	16/3/24 8:00 AM	20/3/24 5:00 PM
8	Optimization	13 days?	12/3/24 8:00 AM	28/3/24 5:00 PM
9	Validation	7 days?	29/3/24 8:00 AM	8/4/24 5:00 PM

Table 4.1 Activity Table

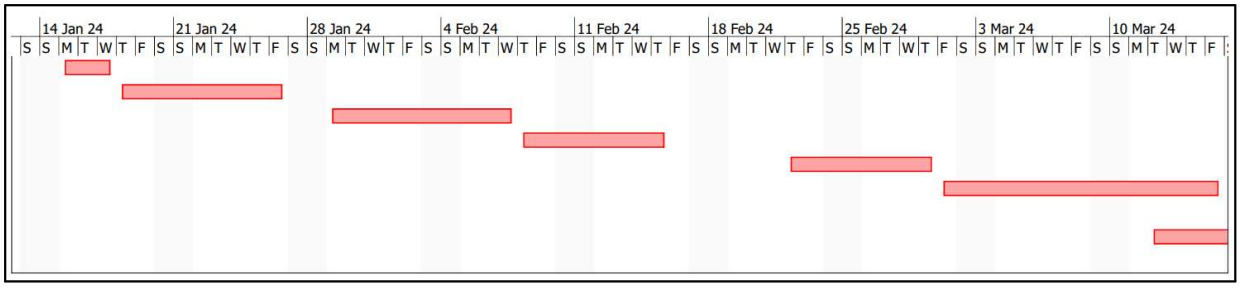

Fig 4.1.0 Gantt Chart

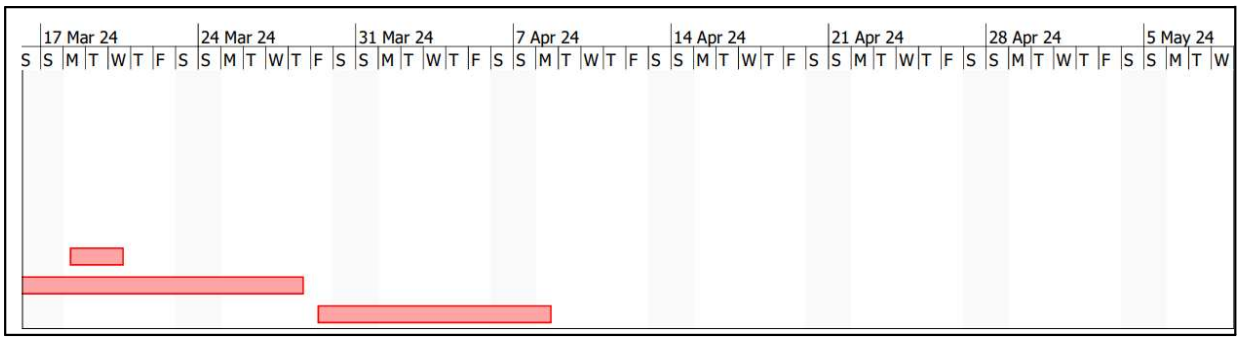

Fig 4.1.1 Gantt Chart

Pert Chart

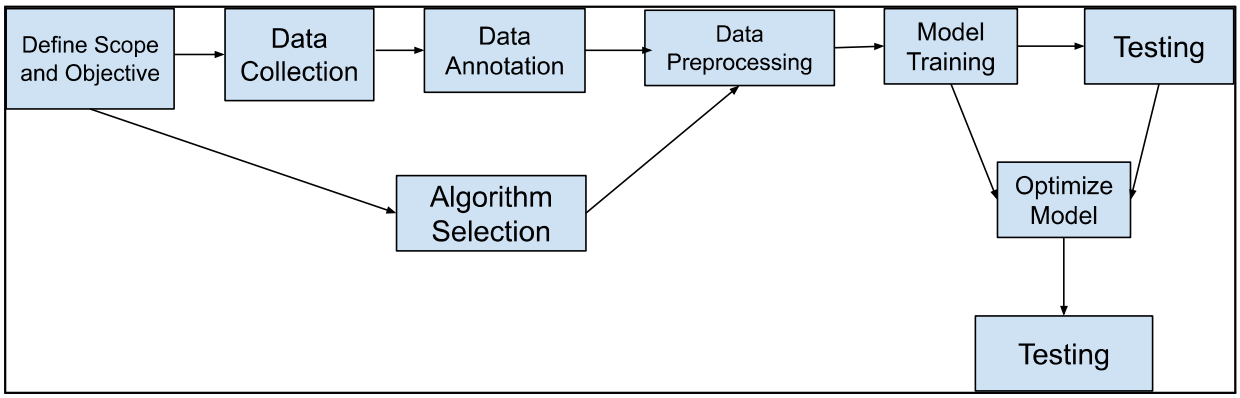

Fig 4.2 Pert Chart

PERT charts, standing for Program Evaluation and Review Technique, are visual project management tools. They map out a project's timeline and break down tasks into a clear sequence. A flowchart specifically designed for projects, where boxes represent milestones and arrows depict the tasks that move you between them. PERT charts are known for using estimated times for tasks, considering optimistic, most likely, and pessimistic scenarios to create a more realistic project schedule.

TESTING TECHNIQUE AND TEST PLANS

Testing Techniques

- Unit Testing: Test individual components/modules of the system in isolation to verify their correctness. This could include testing algorithm for object detection, data preprocessing methods, and model training procedures.
- Integration Testing: Verify that different modules/components of the system work together seamlessly. Test how well the object detection algorithm integrates with other parts of the system such as data input/output mechanisms such as different cameras input and real-time processing components.
- Functional Testing: Evaluate the system's functionality against its specifications. This involves testing various features of the object detection system, such as the ability to detect objects in real-time video streams, accuracy in different lighting conditions, and robustness against occlusions.
- Performance Testing: Assess the system's performance in terms of speed, accuracy, and resource utilization. Measure the time taken for object detection on different hardware configurations and under varying workloads. Check if the system meets the real-time processing requirements.

Relevant Test Cases/Plans:

Unit Test Cases:

- Test data preprocessing functions to ensure correct resizing, normalisation, and augmentation of input images.
- Verify the output of feature extraction functions to ensure they extract relevant features for object detection.
- Test classification functions to ensure they correctly classify objects into predefined categories.

Integration Test Cases:

- Verify that data acquired from sensors or cameras is properly passed to the preprocessing module.
- Test the integration between the preprocessing module and the object detection algorithm to ensure correct input data format.
- Validate that the output of the object detection module is correctly passed to the visualisation module for display.

System Test Cases:

- Test real-time object detection performance by evaluating detection speed and accuracy on a variety of test scenarios.
- Assess the system's robustness by testing its performance under different lighting conditions, occlusions, and cluttered environments.
- Verify that the system meets non-functional requirements such as response time and resource utilisation.

Performance Test Cases:

- Measure the detection speed of the system on different hardware configurations (e.g., CPU vs. GPU).
- Evaluate memory and CPU usage during real-time object detection to ensure optimal performance.
- Test the scalability of the system to handle a large number of concurrent detection tasks.

User Acceptance Test Cases:

- Gather feedback from end-users on the ease of use and intuitiveness of the system interface.
- Address any usability issues or concerns raised by end-users during testing.

Regression Test Cases:

- Re-run previously executed test cases to verify that recent updates or changes to the system have not introduced any regressions.
- Ensure that the system's performance and functionality remain consistent across different versions or iterations.

By employing these testing techniques and executing relevant test cases, the real-time object detection system can be thoroughly evaluated for functionality, performance, integration, user acceptance, and regression, ensuring its reliability and effectiveness in real-world scenarios.

CHAPTER 5

RESULTS AND DISCUSSIONS

USER INTERFACE REPRESENTATION

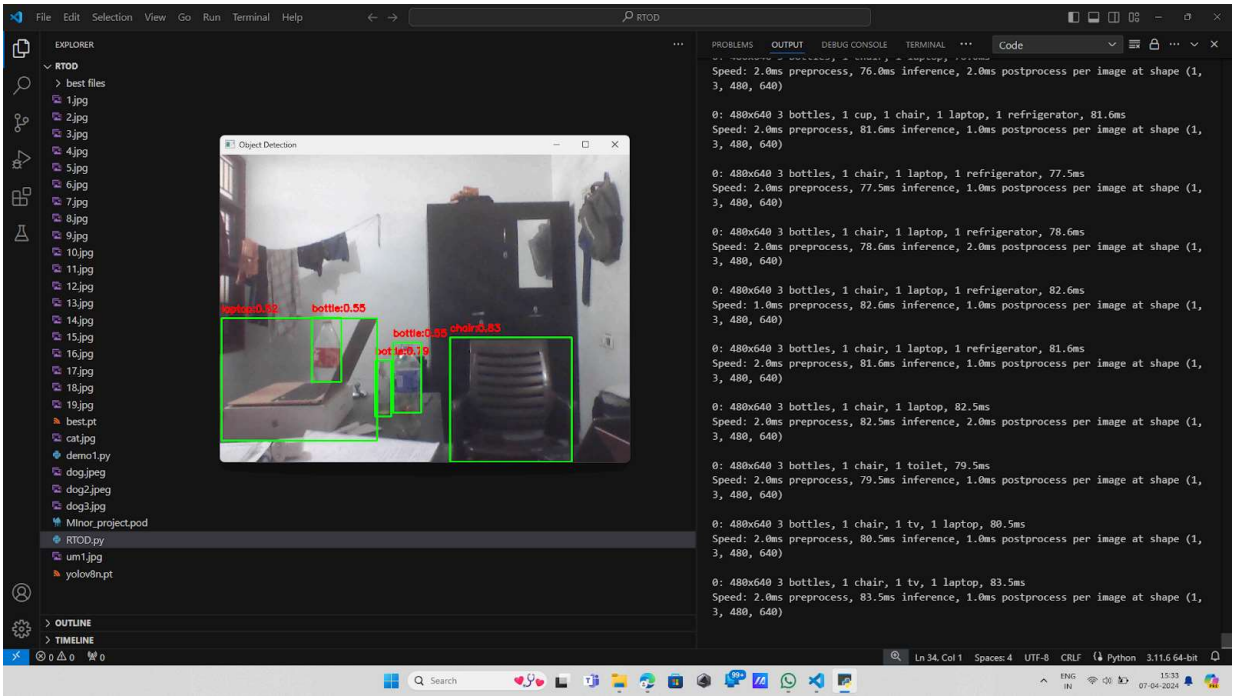

Fig 5.1 (User Interface)

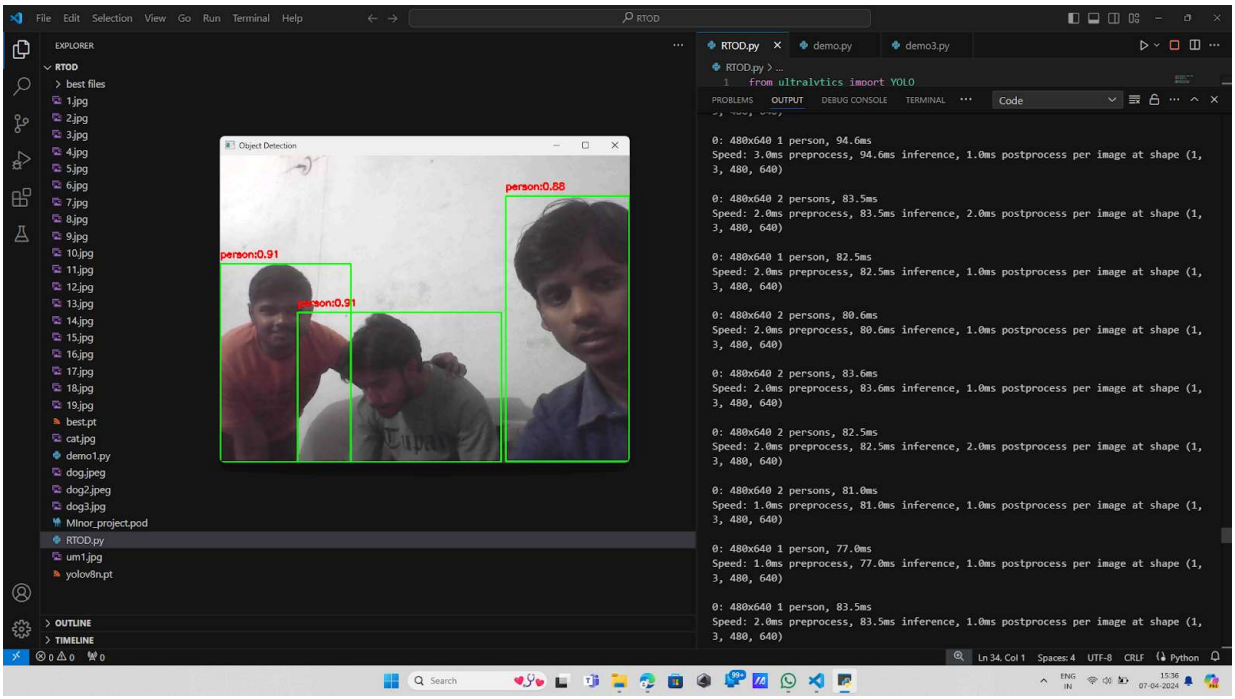

Fig 5.2 (User Interface)

BRIEF DESCRIPTION OF VARIOUS MODULES OF THE SYSTEM

Annotated Output Module: This module includes the output of image or live webcam videostream with annotation of different coloured bounding boxes along with the confidence scores ranging from 0 and 1.

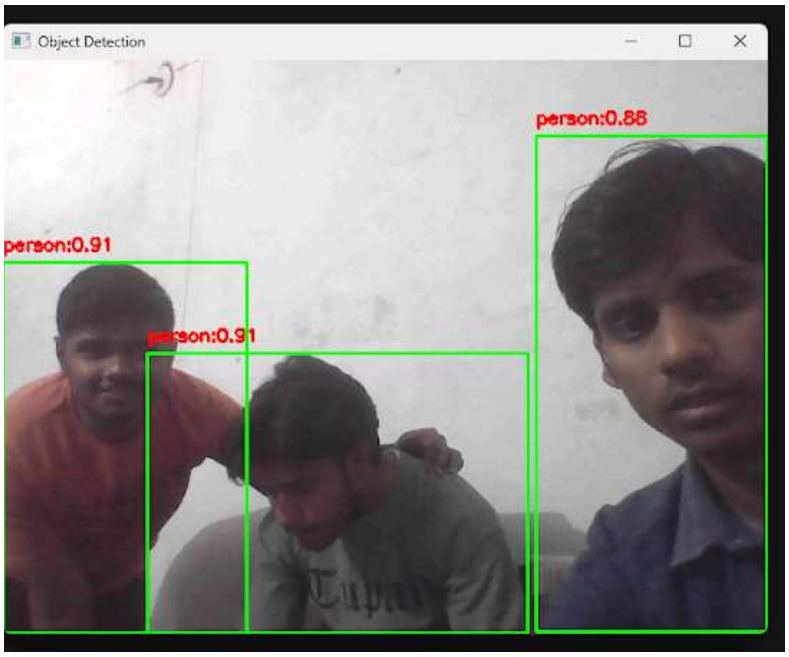

Fig 5.3 Annotated Output of Webcam Video Stream

Terminal Module: This module includes the description of or frame and detected objects which are annotated with different coloured bounding boxes along with following stats.

- Speed: the system calculates the preprocessing and postprocessing speeds taken to identify objects.
- Inference time: time required by the object detection system to take its learned knowledge and apply it to real-time situations, identifying and classifying new objects in new images or video frames
- Shape: Shape with length and breadth in unit pixel of bounding box measurements.

Below is the terminal module output:

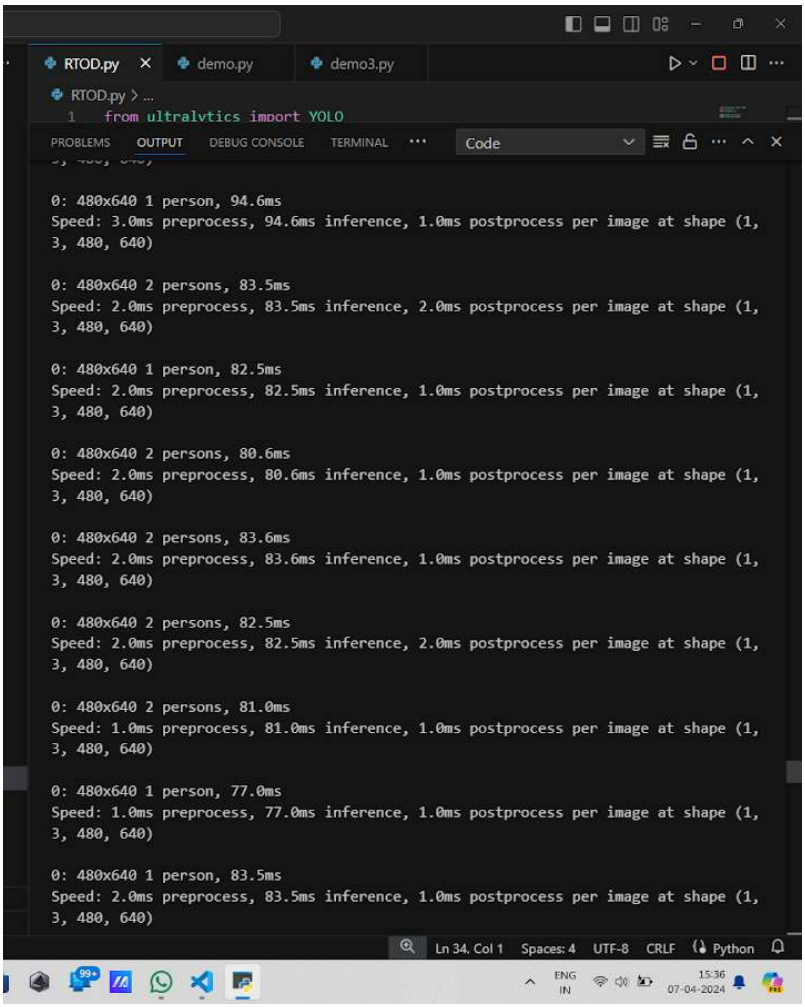

Fig 5.4 Terminal Output Module

SNAPSHOT OF SYSTEM WITH BRIEF DETAIL OF EACH

Roboflow for data annotation:-Roboflow is an online website which we have used for annotating our images.by using roboflow we can annotate a maximum of 10000 of images free of cost.If we want to annotate more images then we need to pay for it but we have used only free service and annotated out data.

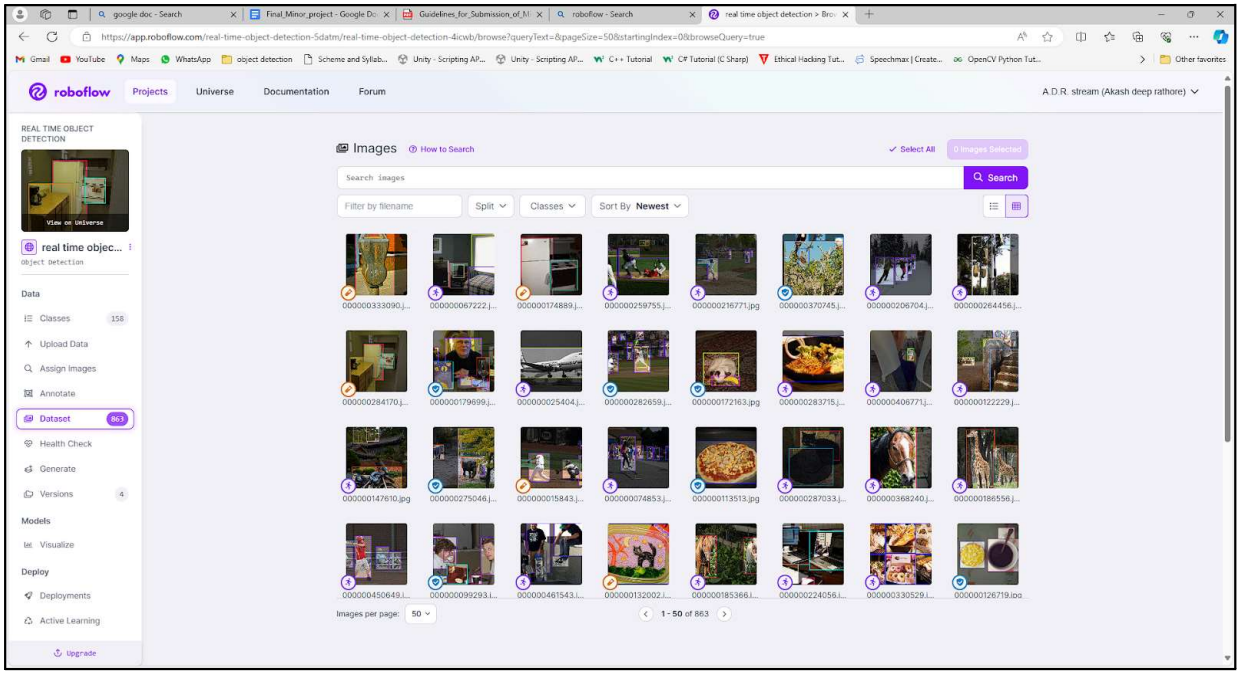

Fig 5.5 Roboflow Annotation tool used for Annotation

Google Colab:-Google Colab is an online IDE which we have used for train our model because it is most faster than our pc and use GPU for training our model which is more powerful than CPU for training purpose.

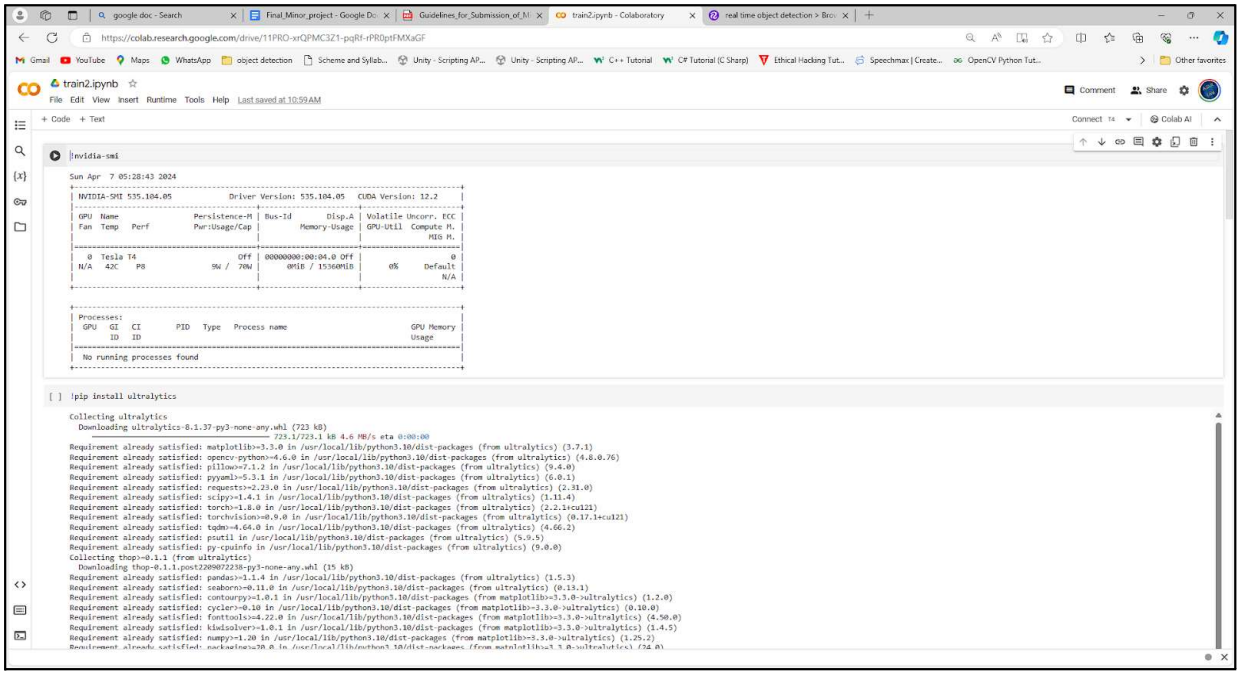

Fig 5.6 Google Colab used for Model Training

Visual Studio Code:-We have implemented our project in visual studio code which is more convenient to use because it comes with many predefined code snippets,and also exclusive language support.and it is also a powerful debugging tool.

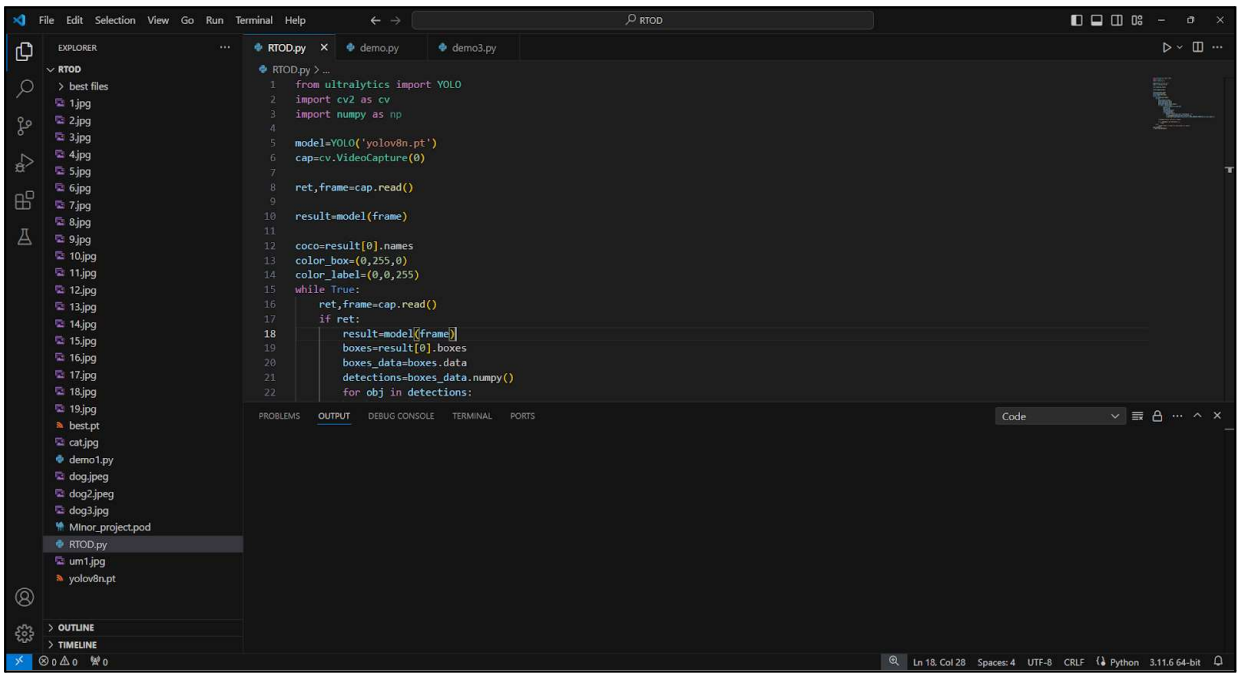

Fig 5.7 Visual Studio Code Snippet of code integrating model and input video/images

RESULTS

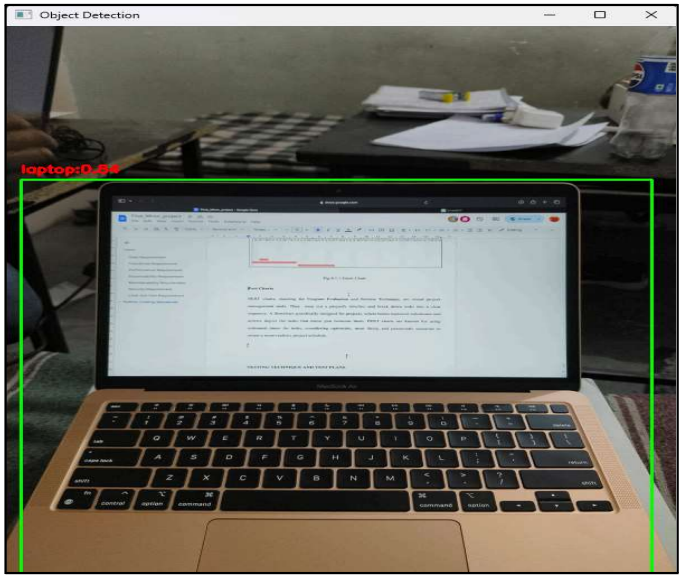

Fig 5.8.0

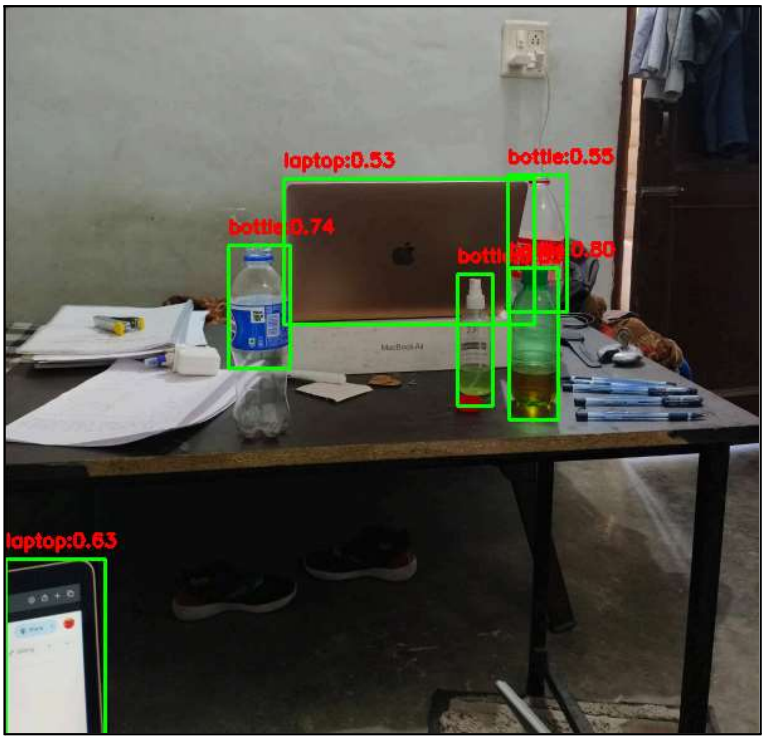

Fig 5.8.1

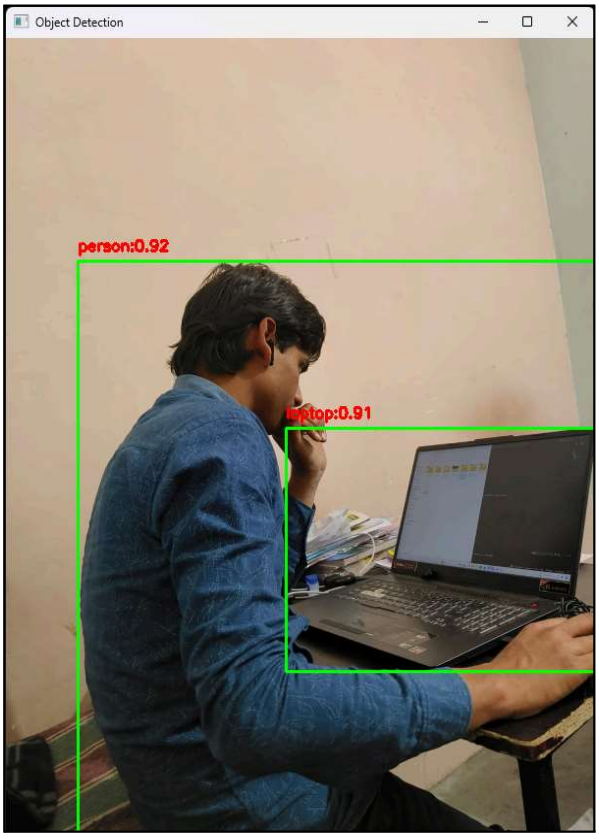

Fig 5.8.2

Fig 5.8.3

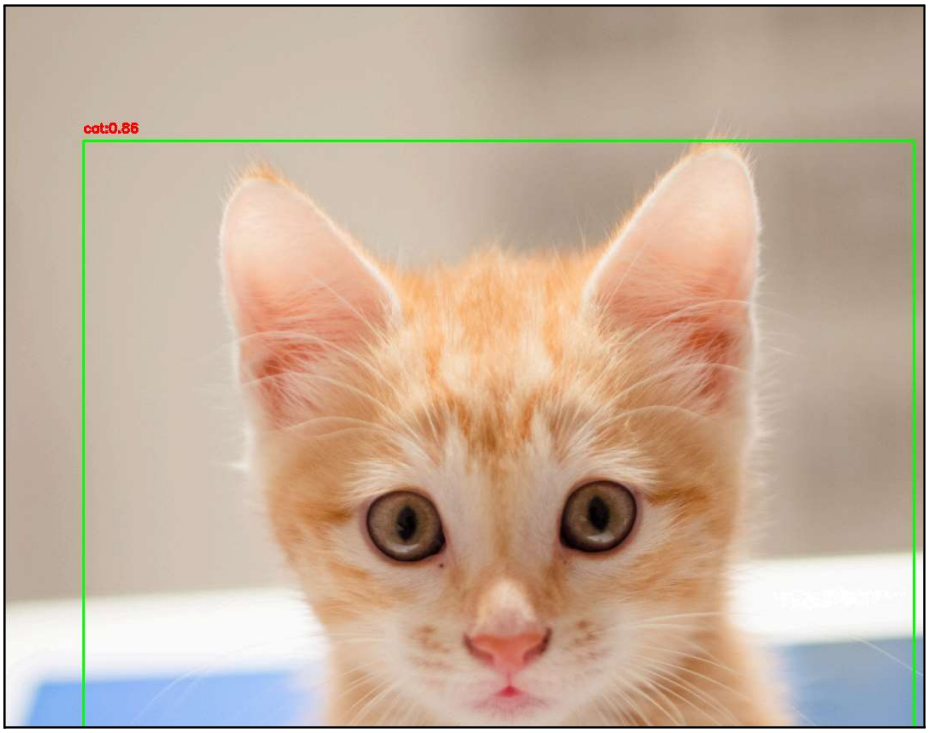

Fig 5.8.4

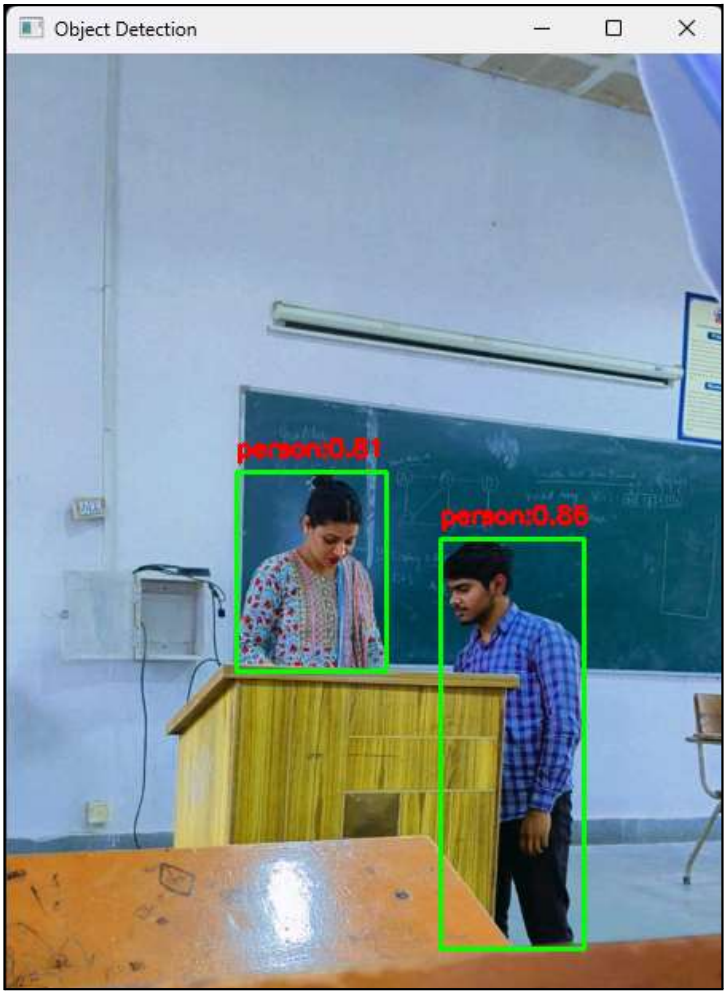

Fig 5.8.5

Fig 5.8.6

Fig 5.8.7

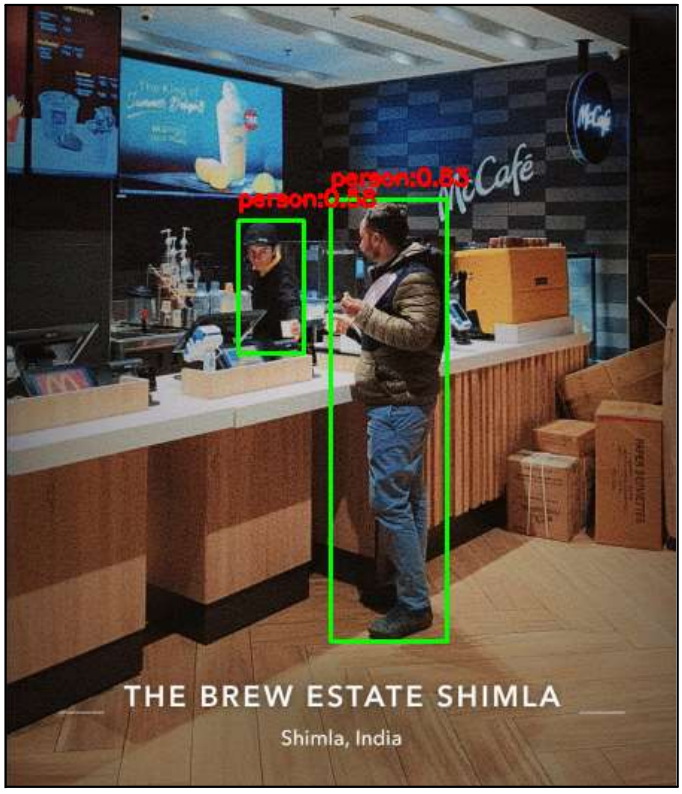

Fig 5.8.8

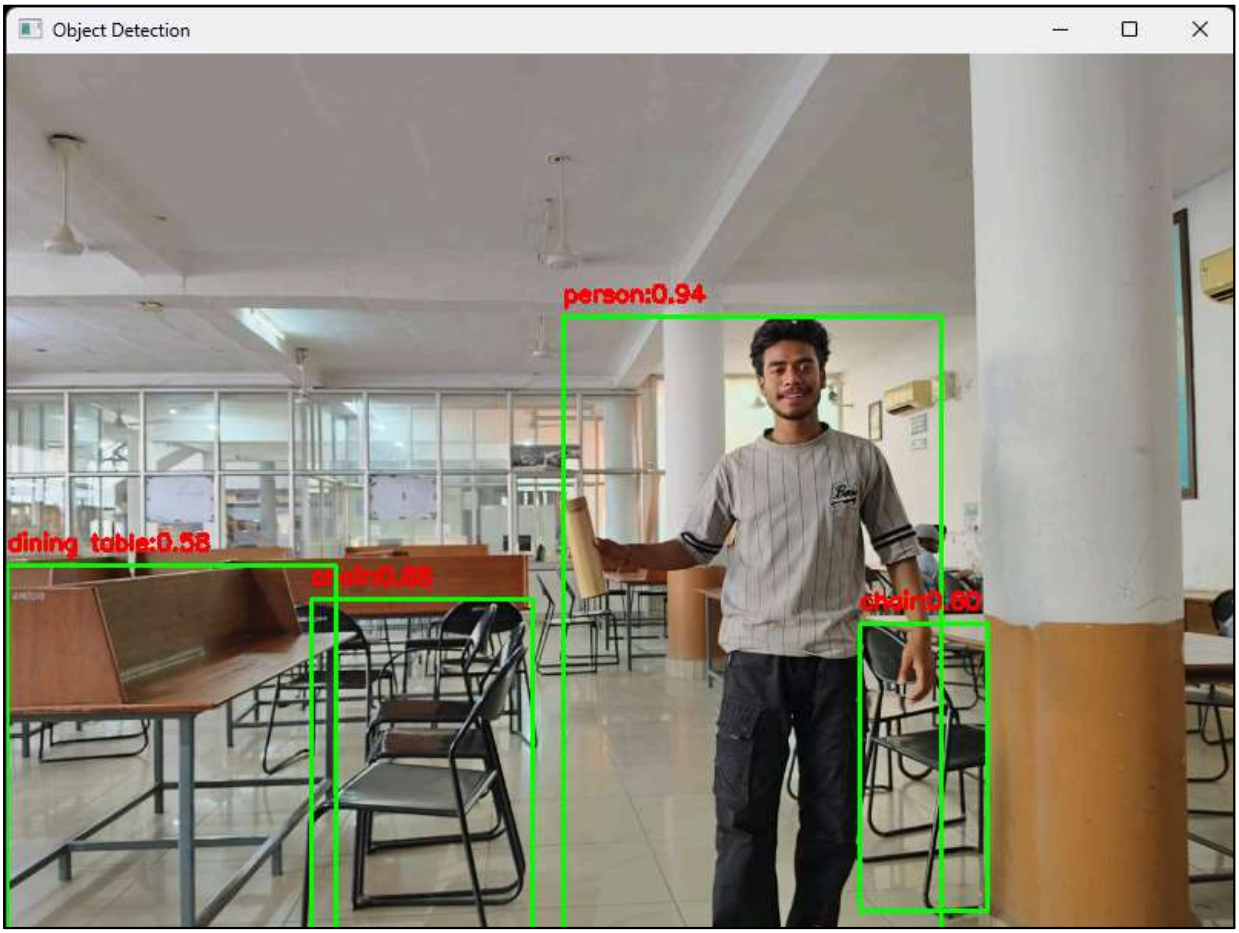

Fig 5.8.9

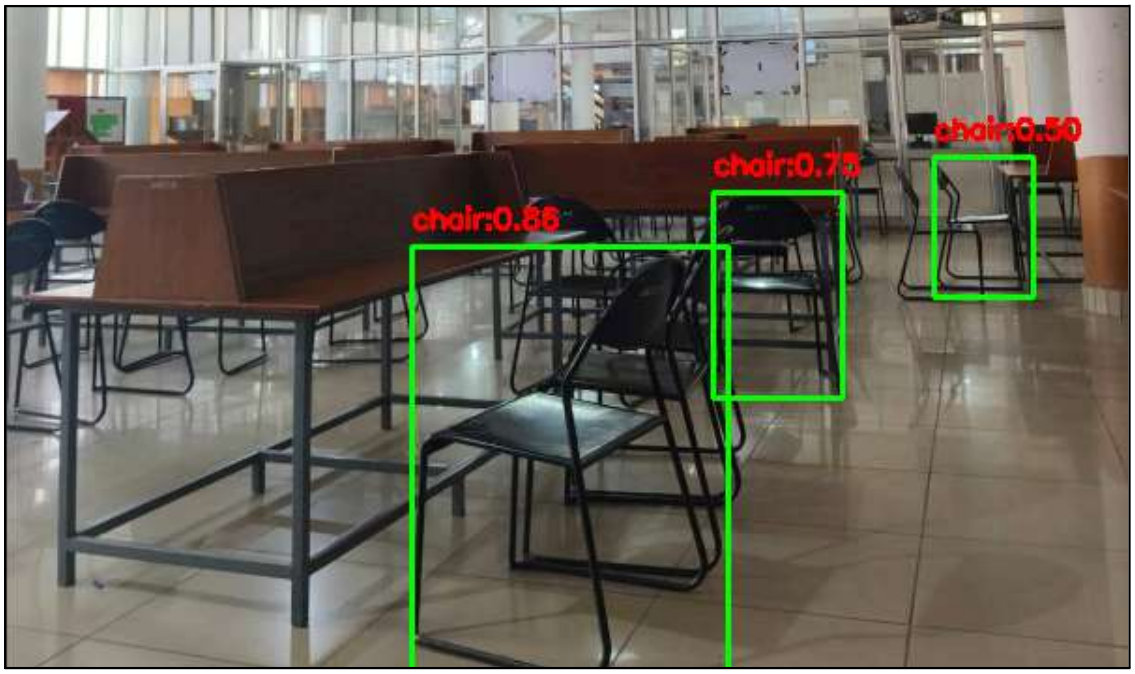

Fig 5.8.10

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

In conclusion, a real-time object detection system is a complex and valuable technology with numerous applications across various industries, including surveillance, autonomous vehicles, retail, healthcare, and more. Through the integration of advanced machine learning techniques, and models like YOLO (You Only Look Once), real-time object detection systems can accurately and efficiently identify objects within a video stream or image feed.

Key points to consider in the conclusion of a real-time object detection system:

- Accuracy and Efficiency: Real-time object detection systems must strike a balance between accuracy and efficiency. While achieving high detection accuracy is crucial for reliable performance, the system must also process data in real-time to meet application requirements.
- Robustness: The robustness of a real-time object detection system is essential for handling various real-world scenarios, including changes in lighting conditions, occlusions, variations in object size and orientation, and cluttered backgrounds.
- Performance Optimization: Optimization techniques such as model quantization, pruning, and hardware acceleration can enhance the performance of real-time object detection systems, allowing them to run efficiently on resource-constrained devices like embedded systems and edge devices.
- Integration and Deployment: Successful deployment of real-time object detection systems requires seamless integration with existing infrastructure, including hardware components, software platforms, and data pipelines. Integration with cloud services and APIs can further extend the capabilities of the system.
- User Interface and Interaction: User interface design plays a crucial role in facilitating user interaction with the real-time object detection system. Intuitive interfaces with

features like live video feeds, customizable settings, and interactive visualisation of detection results enhance user experience and usability.

- Ethical and Privacy Considerations: Real-time object detection systems raise ethical and privacy concerns related to data collection, surveillance, and potential misuse. It's essential to implement appropriate safeguards, including data anonymization, consent mechanisms, and adherence to privacy regulations, to protect user rights and privacy.
- Future Directions: Continued research and development in the field of computer vision and machine learning are expected to drive advancements in real-time object detection systems. Areas of future exploration include improved model architectures, multi-object tracking, 3D object detection, and semantic segmentation.

FUTURE SCOPE

The future scope of real-time object detection systems is vast and promising, with ongoing advancements in technology and applications. Here are several potential directions for future development and innovation in this field:

- Improved Accuracy and Efficiency: Future research will likely focus on enhancing the accuracy and efficiency of real-time object detection systems through advancements in model architectures, training techniques, and optimization algorithms. Techniques such as attention mechanisms, meta-learning, and reinforcement learning may be explored to improve detection performance further.
- Multi-Object Tracking and Interaction Understanding: Real-time object detection systems can evolve to not only detect objects but also track their movements over time and understand interactions between objects. Advancements in multi-object tracking algorithms and scene understanding techniques will enable more comprehensive analysis of dynamic environments, such as crowded urban streets or busy retail stores.

- Edge Computing and Edge AI: The integration of real-time object detection with edge computing and edge AI technologies will enable distributed processing and decision-making at the network edge. This approach reduces latency, conserves bandwidth, and enhances privacy by processing data locally on edge devices, such as IoT sensors, drones, and smart cameras.
- Robustness to Adversarial Attacks and Environmental Variability: Real-time object detection systems must become more robust to adversarial attacks, environmental variability, and real-world challenges such as occlusions, lighting changes, and background clutter. Research into adversarial training, domain adaptation, and robust optimization techniques will help improve system resilience and reliability.
- Human-Centric Applications and Assistive Technologies: Real-time object detection systems can be leveraged to develop human-centric applications and assistive technologies that enhance safety, accessibility, and quality of life. Examples include assistive navigation systems for the visually impaired, real-time hazard detection in industrial settings, and personalised healthcare monitoring.
- Ethical and Social Implications: As real-time object detection systems become more pervasive in society, it's essential to address ethical and social implications such as privacy concerns, bias and fairness issues, and societal impact. Future research and development efforts should prioritise ethical considerations and responsible deployment practices to ensure positive societal outcomes.