A Midterm Progress Report
On
REAL TIME OBJECT DETECTION

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	AKASH KUMAR	ANMOL KUMAR
2104057	2104062	2104070

UNDER THE GUIDANCE OF

ER. JASBIR SINGH SAINI

(FEB 2024)

Department of Computer Science and Engineering

GURU NANAK DEV ENGINEERING COLLEGE,

LUDHIANA

INDEX

INTRODUCTION

Most of the cases, Humans performs work by observing their environment and perform action accordingly, and the observation of the environment is done using their eyes, means that they take input through their eyes and this observed data is send to their mind for processing and process is done through their mind instantly in real time. The brain observes this data and detect different object and send command to our body part to do action.

In our project we are aims to make 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 neural network's different techniques to make a system which can detect different object in our scene in 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 consists of multiple layers of neural that learns 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 example of single shot methods are You Only Look Once (YOLO) and Single Shot Multi-Box 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 5detect different object.

Objectives :-

1. To collect the data and annotate it.
2. To develop a real time object detection system using YOLO and integrate it's visualisation using deep learning
3. Validation and testing with existing system

SYSTEM REQUIREMENTS

Software Requirement:-

- **Operating System:**

  Depending upon requirements of the project different operating system like windows or linux based system can be used for better compatibility and efficiency.

- **Python and it's library (Such as Pytorch, Tensorflow)**

  Python(3.11.6) as a language is used to develop this project along with various dependencies and libraries of python needs to be installed e.g matplotlib,cv2 etc.

- **Computer Vision library(Opencv)**

  Computer vision library CV2 is used :

  - opencv-contrib-python (4.9.0.80): This includes core OpenCV functionality and additional contributed modules, with GUI support.
  - opencv-contrib-python-headless (4.9.0.80): This offers the same functionality as opencv-contrib-python but without GUI support.
  - opencv-python (4.8.1.78): This is the core OpenCV library without the contributed modules, and its version is older than the others.
  - opencv-python-headless (4.9.0.80): This provides the core functionality without GUI or contributed modules, with the same version as opencv-contrib-python-headless.

- **IDE(Integrated Development Environment):**

  Various IDEs can be used to organise the necessary files related to project that offers better code optimisation:

  - Visual Studio Code
  - Pycharm
  - IDLE Python

Hardware Requirement:-

- CPU(Central Processing Unit):
    1. CPU is responsible for general processing tasks like running the operating system and software.
    2. More cores (e.g., 4+) and higher clock speeds (e.g., 3 GHz+) are better for computationally intensive tasks like object detection.
    3. Consider Intel Core i5/i7 or AMD Ryzen 5/7 series processors for good balance and performance.
- RAM(More than 8GB):
    1. RAM is used to store data actively used by the system.
    2. More than 8GB is recommended to handle object detection models, input/output data, and other running processes.
    3. 16GB or more is ideal for smoother performance, especially with larger models or video processing.
- GPU(Graphics Processing Unit):
    1. Stores data actively used by the system.
    2. More than 8GB is recommended to handle object detection models, input/output data, and other running processes.
    3. 16GB or more is ideal for smoother performance, especially with larger models or video processing.
- Camera:
    1. Captures image or video data for object detection.
    2. Resolution (e.g., 720p, 1080p) and frame rate depend on your needs. Higher resolution and frame rate require more processing power.
    3. USB webcams are common for basic projects.

SOFTWARE REQUIREMENT ANALYSIS

In the context of the Real Time Object Detection, Software Requirement Analysis serves as a foundational phase in the development process, guiding the design and implementation of a robust system that detects objects of various classes at much higher speeds. This crucial step involves a systematic approach of understanding and documenting the specific needs and functionalities required to classify objects effectively. Software Requirement Analysis lays the groundwork for the successful development and implementation of the Real Time Object Detection, ensuring that it addresses the identified needs and challenges associated with inaccurate object detection at slow speeds. Through a comprehensive understanding of user requirements and system constraints, the project team can deliver a solution that is intuitive, efficient, and user-friendly, ultimately enhancing communication and collaboration across diverse linguistic contexts.

Problem Definition:

Develop a real-time object detection system that can identify specific objects of interest within video streams with high accuracy. The system should be adaptable to different environments and object types.

Functional Requirement:

1. **Object Detection:-** Identifies and localize object of interest in real time webcam by creating bounding boxes and annotation.
2. **Performance:-** Achieve real time processing with minimal latency for live video input.
3. **Adaptability:-** Allow for training and fine tuning the detection model on different dataset and environment.

Non-Functional Requirements:

1. **Reliability:-**The system should operate reliably with minimal errors and crashes.
2. **Maintainability:-**The code should be well documented and modular for easy maintenance and updates.
3. **Scalability:-**The system should be adaptable to handle different processing loads and data volumes.

Modules:

1. **Preprocessing Module:-** preprocessing refers to the steps performed on image data before feeding it into an object detection model. These steps aim to enhance the quality of the data and prepare it for effective training. Such as:- Data Cleaning,Noise reduction,Contrast Enhancement etc.
2. **Object Detection:-** Load and run a pre-trained or custom object detection model.it identifies the objects of interest and generate boundary boxes or other localization data.It has following functionalities:
    - Model loading and execution: Loads a pre-trained object detection model from a file. Initializes and runs the model on each video frame.
    - Object identification: Analyzes the input frame using the model to determine the presence and type of objects. Categorizes objects based on pre-defined classes (e.g., "person," "vehicle," "animal").

3. **Post Processing:-** Post-processing plays a crucial role in refining the output of an object detection model. It has following functionalities:
    - Filtering:Removes detections with low confidence scores or that violate spatial or temporal constraints.

- Tracking: Links detections across consecutive frames to create object tracks, improving temporal consistency.
- Refine bounding boxes: Adjust the size or position of bounding boxes based on additional information (e.g., aspect ratio, context).

4. **User Interface:-** Display video stream with detected objects highlighted. It has following functionalities:
    - Video stream display: Presents the real-time video stream with detected objects overlaid in a clear and visually appealing manner.
    - Bounding box visualisation: Draws bounding boxes around detected objects, optionally with labels or confidence scores.
    - Additional information display: Shows relevant data about detections, such as object type, confidence score, or tracking ID.

SOFTWARE DESIGN

The Software Design section of the Real Time Object Detection System serves as the architectural backbone, providing a comprehensive roadmap for the implementation of the system's functionality. This section meticulously outlines the design solutions tailored to address each of the defined objectives with precision and efficacy. Through detailed diagrams, conceptual models, and descriptive narratives, this section elucidates the structural framework and functional components essential for the successful development of the Real Time Object Detection system.

Software design is of paramount importance as it lays the foundation for the entire project, guiding the translation of conceptual ideas into tangible software solutions. By articulating the system's architecture, module interactions, and data flow, the design phase ensures clarity and coherence in the implementation process. Moreover, a well-designed system not only enhances its robustness and scalability but also streamlines the development workflow, mitigating risks and reducing the likelihood of errors or inefficiencies. As we delve into the intricacies of software design, we aim to provide a holistic view of the Real Time Object Detection architecture, encompassing innovative approaches to speech recognition, translation algorithms, user interface design, and system integration. Through this meticulous design process, we endeavour to realise the project's objectives with precision, efficiency, and reliability, ultimately delivering a transformative solution that empowers users to transcend Object Detection barriers.

Block Design:

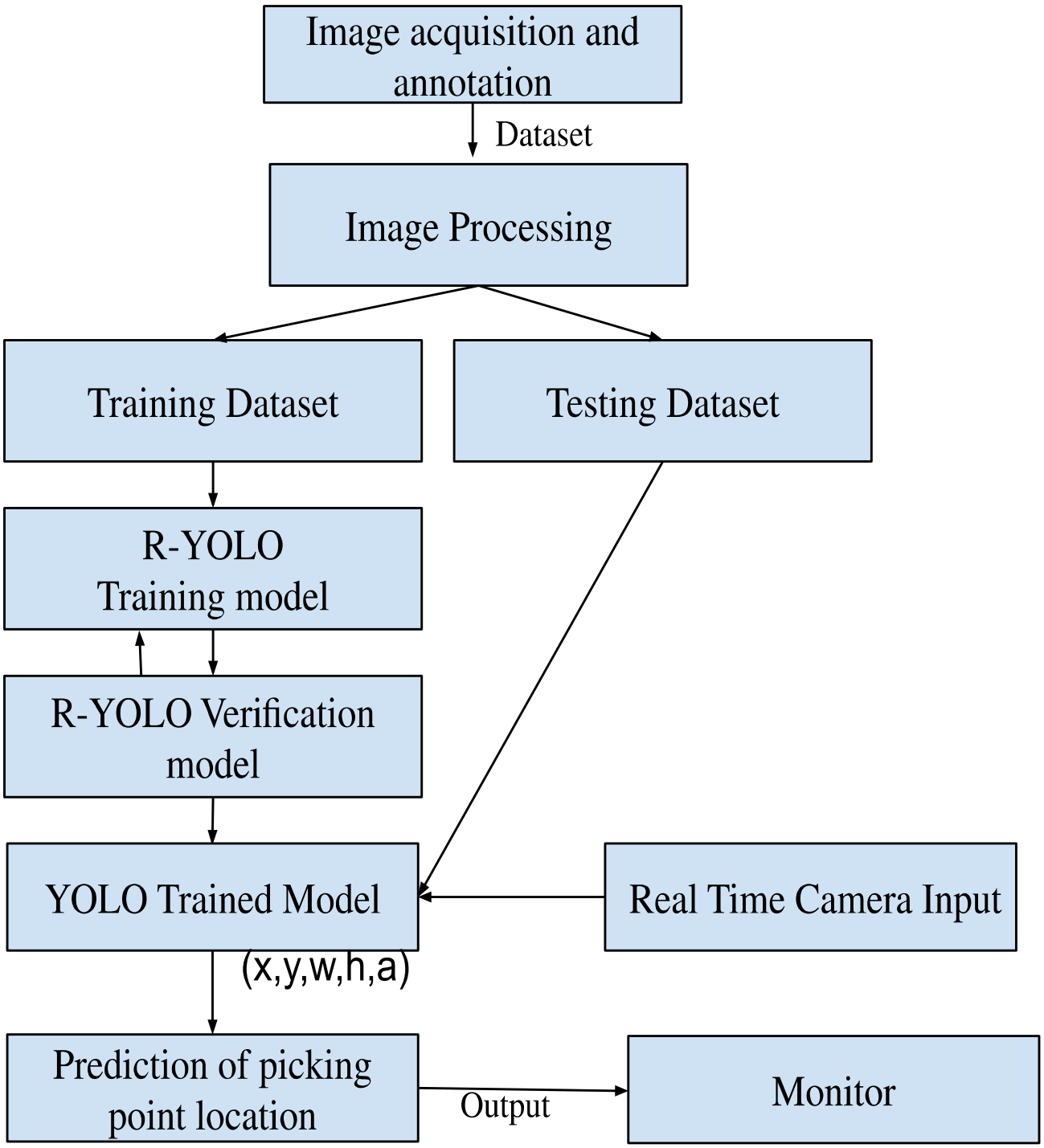


Fig No 1

This block diagram outlines a system that captures images, trains an object detection model to identify Object, and then uses the detections to predict their locations in the image.

Image Acquisition and Annotation:

This block captures images or videos from a real-time camera input.It might also include a mechanism to manually annotate images with bounding boxes around the humans and dogs present, which is used to train the object detection model later.

Dataset:

This block stores the captured images and their corresponding annotations, creating a dataset for training and testing the object detection model.

Image Processing:

This block pre-processes the images before feeding them into the object detection model.Common pre-processing steps include resizing, normalisation, and colour space conversion.

Training Dataset and Testing Dataset:

The dataset is split into two parts:Training Dataset: Used to train the object detection model. Testing Dataset: Used to evaluate the model's performance on unseen data.

R-YOLO:

This block refers to a specific object detection model called "R-YOLO," which is likely a variant of the popular YOLO (You Only Look Once) algorithm. This model is responsible for identifying humans and dogs in the images.

Training Model:

This block trains the R-YOLO model on the training dataset. During training, the model learns to associate specific image features with the presence of humans and dogs.

R-YOLO Verification Model:

This block might refer to a separate model used to verify the detections made by the main R-YOLO model.It could add an extra layer of confidence or handle challenging cases where the main model is unsure.

Prediction of Picking Point Location:

This block likely refers to the final step where the system determines the location (e.g., coordinates) where a human or dog is detected in the image. This information might be used for tasks like picking objects associated with the detected humans or dogs.

Output:

This block represents the final output of the system, which could be: Bounding boxes around detected humans and dogs, along with their predicted locations.

UML Diagram:

UML (Unified Modelling Language) is a diagram that is generated from below code.

```
@startuml
        package "Real-Time Object Detection System" {
           interface Camera
           interface ObjectDetector
           interface Display
           class DetectionFrame
           class DetectedObject
           Camera --> DetectionFrame
           ObjectDetector --> DetectionFrame
           Display --> DetectionFrame
           ObjectDetector --> DetectedObject
        }
@enduml
```

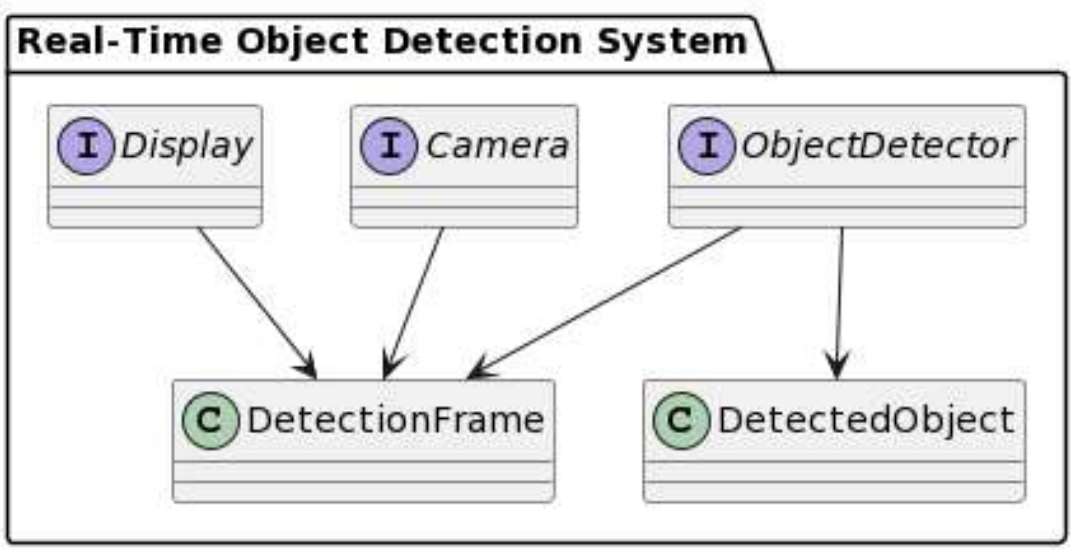


Fig No.2

Activity Diagram

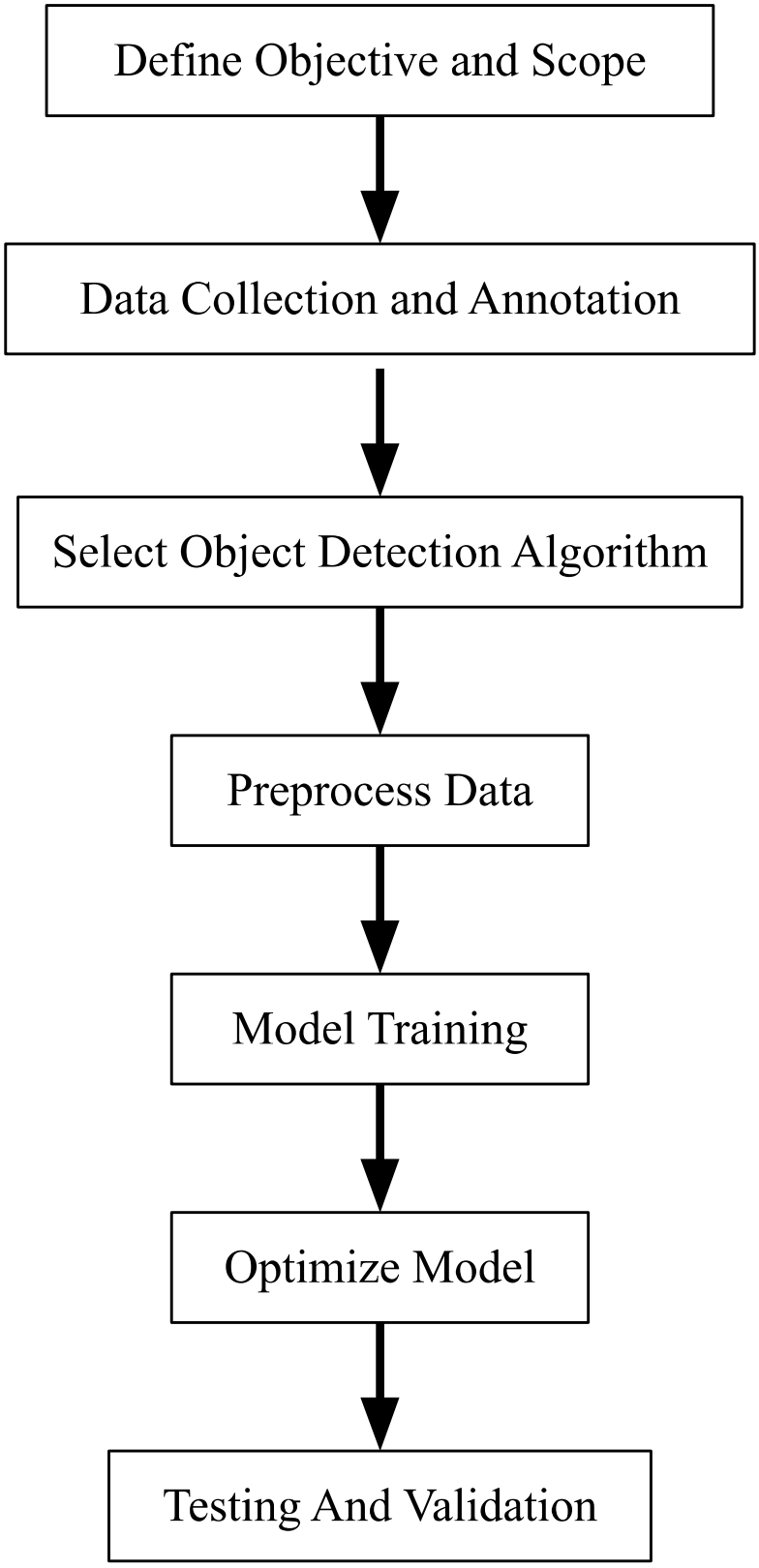


Fig No 3

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:** Optimise 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. **Testing and Validation:** Conduct thorough testing and validation of the real-time object detection system.

Entity Relationship (ER) Diagram

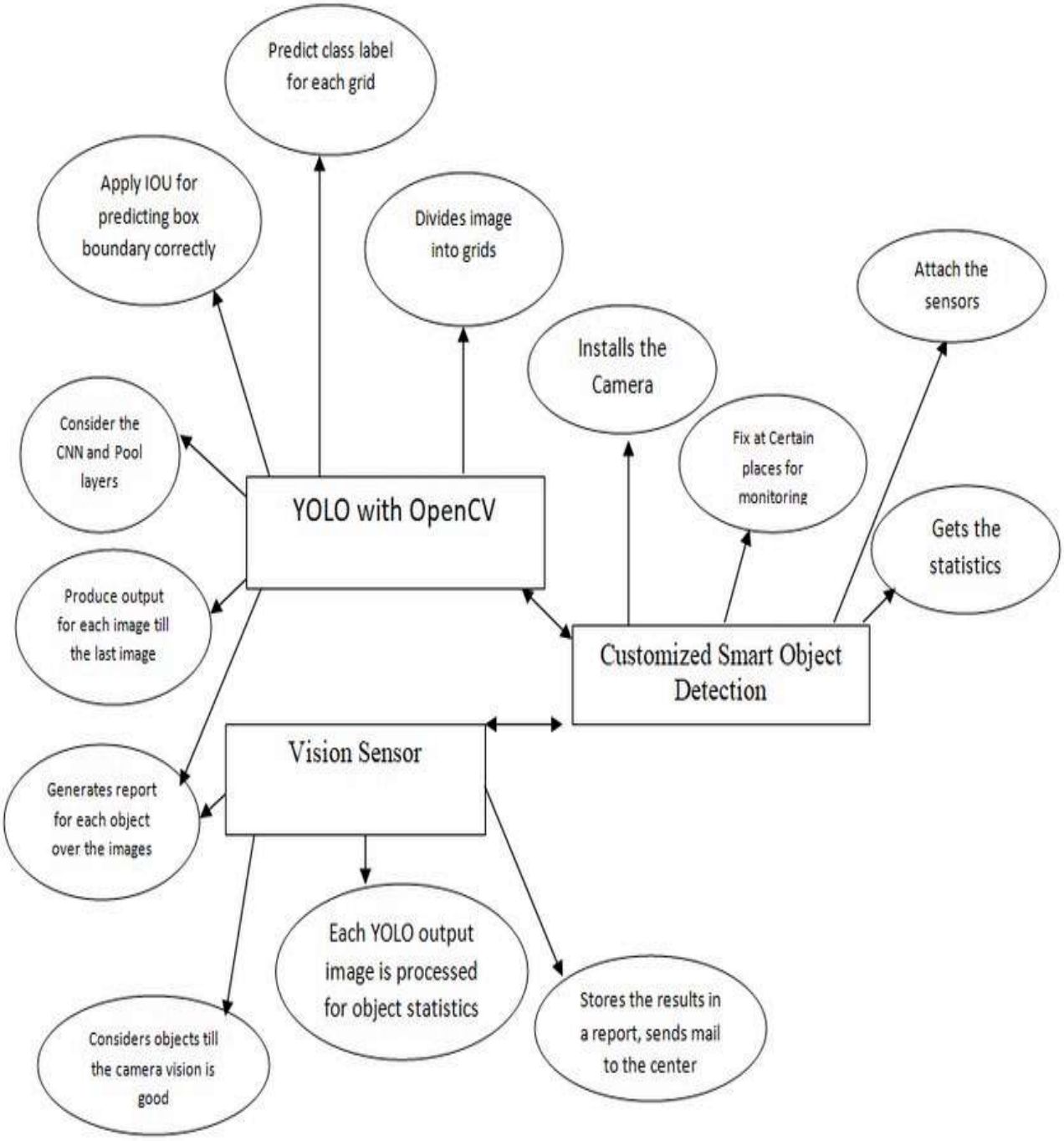


Fig No 4

This ER Diagram outlines the steps involved, from setting up the camera to obtaining object statistics:

1. Camera Installation:
    - Begin by installing a camera or using an existing one.
    - Ensure proper connectivity and configuration.
2. Image Capture:
    - The camera captures an image or video frame.
    - This raw input serves as the starting point for object detection.
3. Preprocessing:
    - Apply preprocessing techniques to enhance the image quality:
        - Resize the image to a standard size.
        - Normalise pixel values.
        - Remove noise (if any).
4. YOLO Model Loading:
    - Load the pre-trained YOLO model using OpenCV.
    - YOLO is a deep learning-based object detection algorithm.
5. Object Detection:
    - Pass the preprocessed image through the YOLO model.
    - YOLO identifies objects and their bounding boxes.
    - Classify detected objects (e.g., person, car, dog).
6. Bounding Box Visualization:
    - Draw bounding boxes around detected objects.
    - Annotate each box with the corresponding class label.
7. Post-processing:
    - Filter out low-confidence detections (based on confidence scores).
    - Non-maximum suppression: Remove overlapping boxes.

8. Object Statistics:
    - Calculate statistics for detected objects:
        - Count the total number of objects.
        - Compute average size or area.
        - Determine the most frequent class.
9. Display Results:
    - Display the annotated image with bounding boxes.
    - Show object labels and confidence scores.
10. Further Analysis:
    - Optionally, save results to a database or log file.
    - Use the object statistics for further analysis or decision-making.

Solutions To The Proposed Objectives:

1. **To collect the data from publicly available resources:-**

   To collect data for a real-time object detection system, leverage publicly available resources like online datasets, open repositories, and APIs. Explore datasets like COCO, ImageNet, and Open Images Dataset, which offer labelled images suitable for training models. Additionally, search open repositories on platforms like GitHub and Kaggle for object detection datasets and pre-trained models.

   Utilise APIs such as Google Cloud Vision and Microsoft Azure Computer Vision to access pre-trained models for object detection. Consider web scraping to gather images from relevant websites, ensuring proper metadata extraction. Engage in community forums and challenges like Kaggle competitions to access curated datasets and insights. Augment collected data using techniques like rotation and flipping for enhanced diversity.

   Prioritize proper annotation and labeling of data for effective model training. Maintain compliance with legal and ethical standards by respecting terms of use and licensing agreements. By combining these approaches, assemble a comprehensive dataset to train and evaluate your real-time object detection system effectively.

2. **To develop a real time object detection system using YOLO and integrate it's visualisation using deep learning**

   To develop a real-time object detection system using YOLO (You Only Look Once) and visualise it using deep learning, First, set up an environment with libraries like TensorFlow or PyTorch for deep learning and OpenCV for image processing.

   Next, gather a suitable dataset containing annotated images. Choose a YOLO variant based on your project's requirements and train the model using the dataset, fine-tuning pre-trained weights if available. Evaluate the model's performance on a validation dataset, adjusting hyperparameters as needed. Deploy the trained model for real-time inference, processing frames from a webcam or video feed. Visualise the detection results by drawing bounding boxes around objects and labelling them with their class names and confidence scores using OpenCV or Matplotlib.

   Optimise the model and inference pipeline for real-time performance using techniques like quantization or using lighter YOLO variants. Finally, deploy the system and monitor it for performance and maintenance, periodically retraining the model with new data. Following these steps will enable you to create a robust real-time object detection system with YOLO and visualise the detection results using deep learning techniques.

3. **Validation and testing with existing system**

To test and validate a real-time object detection system in comparison to an existing system, start by defining evaluation metrics such as precision, recall, and accuracy. Prepare a dataset reflecting real-world scenarios with ground truth annotations.

Choose a baseline model for comparison, like the current system or a benchmark model such as YOLO or Faster R-CNN. Set up a testing environment ensuring consistency between systems. Conduct performance comparisons using defined metrics on the same dataset. Analyze results to identify strengths and weaknesses of the new system. Iterate on improvements based on analysis, adjusting parameters and algorithms as needed.

Validate the final system on a separate dataset to ensure generalization. Document the testing process, results, and recommendations for deployment. This comprehensive approach ensures that the new object detection system is thoroughly evaluated, refined, and validated for real-world deployment, ultimately leading to improved performance and effectiveness compared to the existing system.

TESTING MODULE

The testing phase of the Real Time Object Detection project is a crucial stage in ensuring the reliability, accuracy, and performance of the translation system. This module encompasses a comprehensive set of testing techniques and methodologies designed to evaluate the functionality, usability, and robustness of the system across various scenarios and use cases. The overarching goal of testing is to identify and address any potential issues or defects in the system, thereby enhancing its overall quality and user experience.

Proposal of Testing Techniques:

1. Unit Testing: Test individual components of the system in isolation to ensure they perform as expected. This includes testing functions/methods responsible for data preprocessing, feature extraction, and classification.
2. Integration Testing: Verify the interaction between different modules/components of the system to ensure they work together seamlessly. Test the integration of data acquisition, preprocessing, object detection, and visualization modules.
3. System Testing: Evaluate the system as a whole to ensure it meets functional and non-functional requirements. Test real-time object detection performance, accuracy, and robustness under various conditions.
4. Performance Testing: Assess the system's performance in terms of speed and resource utilization. Measure the detection speed on different hardware configurations and evaluate memory and CPU usage.
5. User Acceptance Testing (UAT): Involve end-users or stakeholders to validate that the system meets their requirements and expectations. Gather feedback on usability, accuracy, and overall satisfaction with the system.

6. Regression Testing: Ensure that recent updates or changes to the system have not introduced regressions or unintended side effects. Re-run previously executed test cases to verify system stability.

Relevant Test Cases:

1. Unit Test Cases:
    - Test data preprocessing functions to ensure correct resizing, normalization, 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.
2. 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 visualization module for display.

3. 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 utilization.

4. 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.
5. User Acceptance Test Cases:
    - Gather feedback from end-users on the ease of use and intuitiveness of the system interface.
    - Validate that the system meets the accuracy requirements specified by end-users or stakeholders.
    - Address any usability issues or concerns raised by end-users during testing.
6. 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.

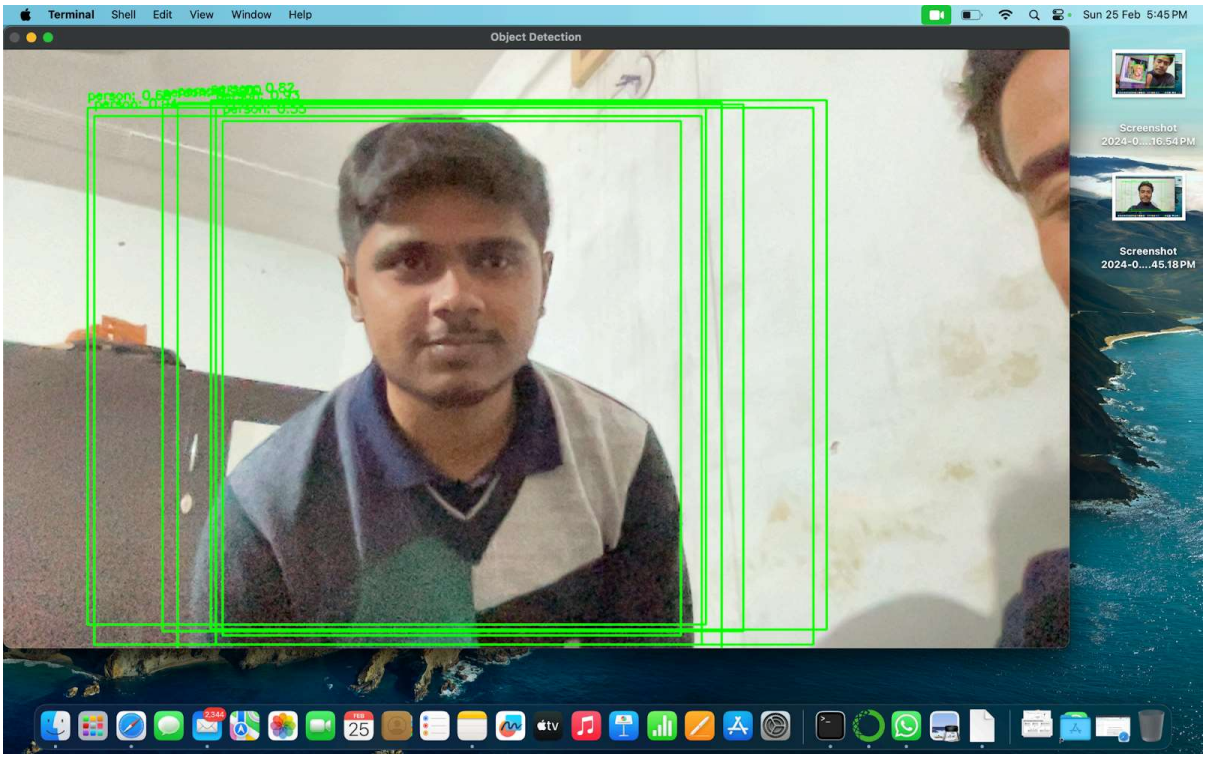
PERFORMANCE OF THE PROJECT DEVELOPED (SO FAR)

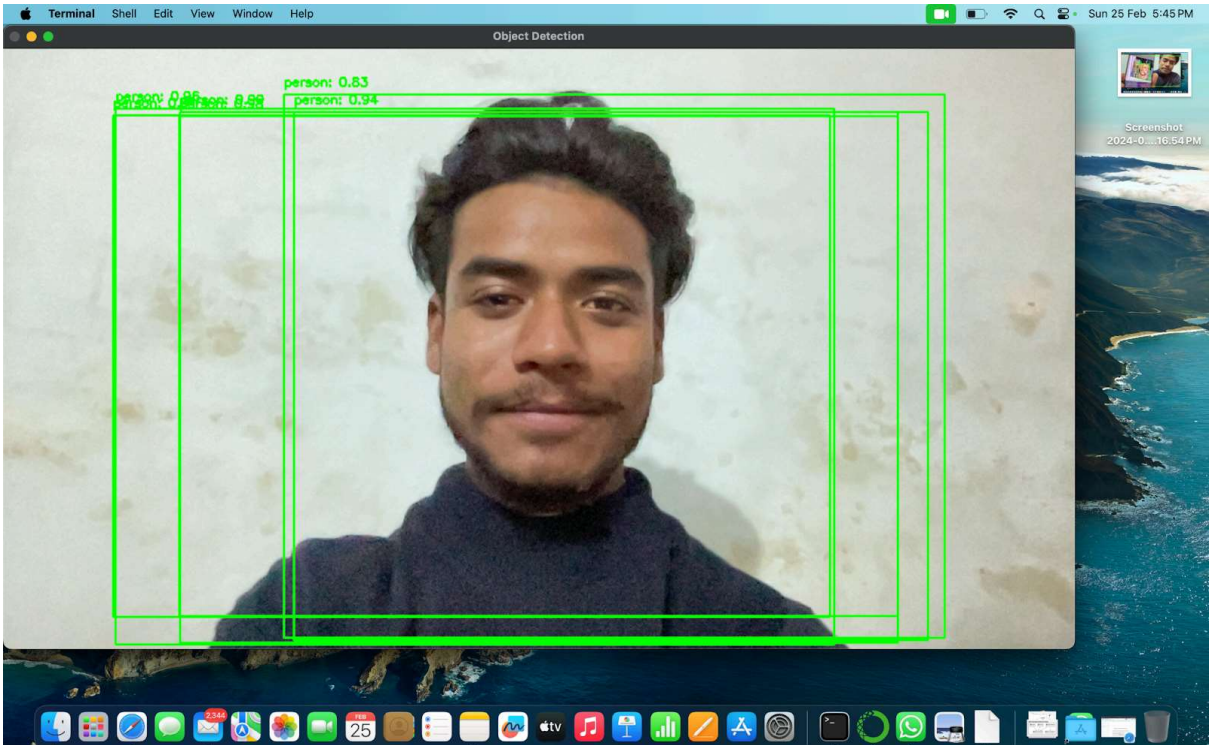
We have completed our first objective of collecting relevant data (COCO model of estimated size 6GB) which includes millions of object, images in different setup and preprocessed it and start implementing it using YOLO model which detect various objects and annotate it using bounding boxes around the object frame because it is a very big file so we haven’t implemented our project using this huge data set so we have only implemented using a small dataset which is preferred for YOLO and tested our model.

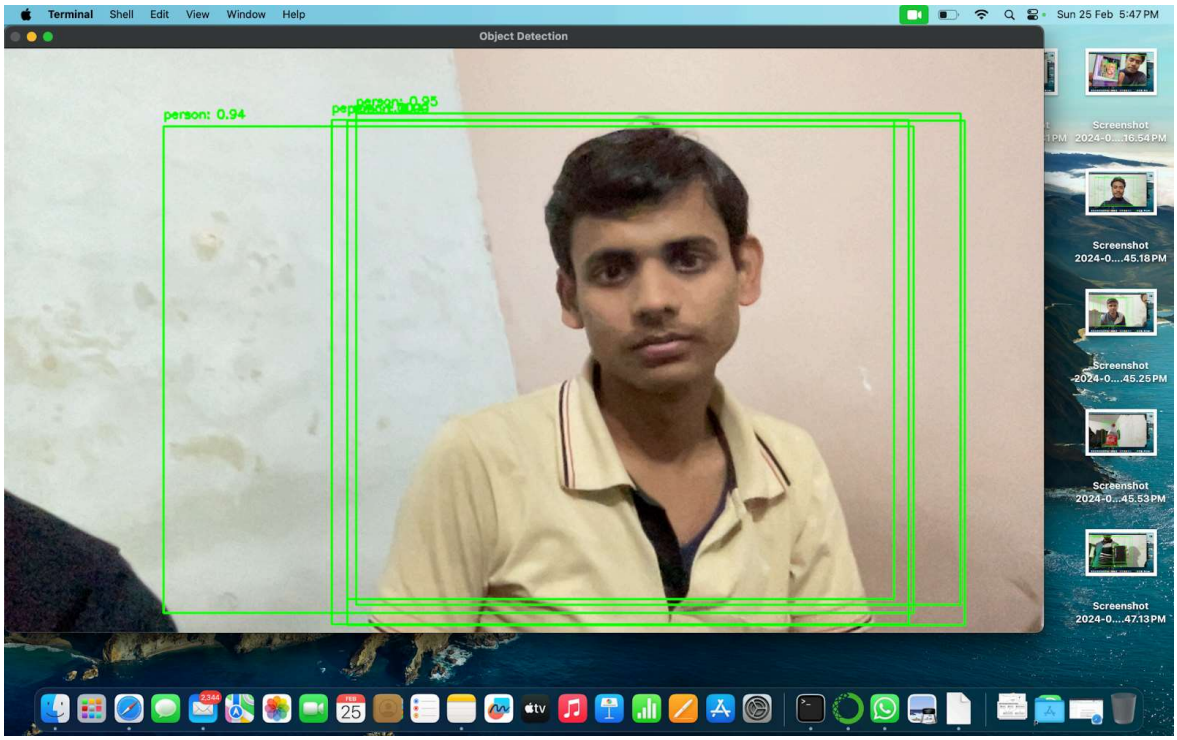
Our model is working properly with this small dataset but it can not detect a huge amount of object right now and also giving a big time complexity. As the project advances, further refinement and optimization efforts will be undertaken to ensure the system's effectiveness and reliability.

OUTPUT SCREEN

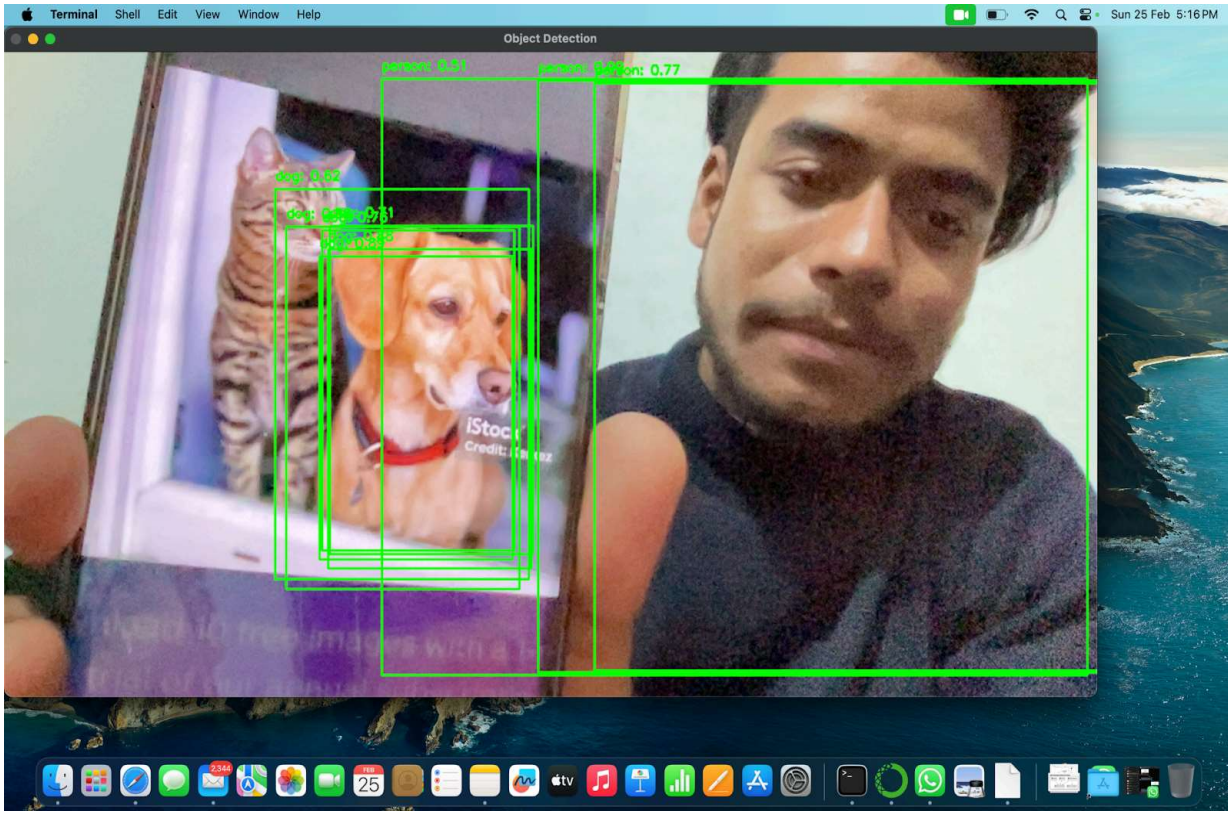
1. Detection of Person :

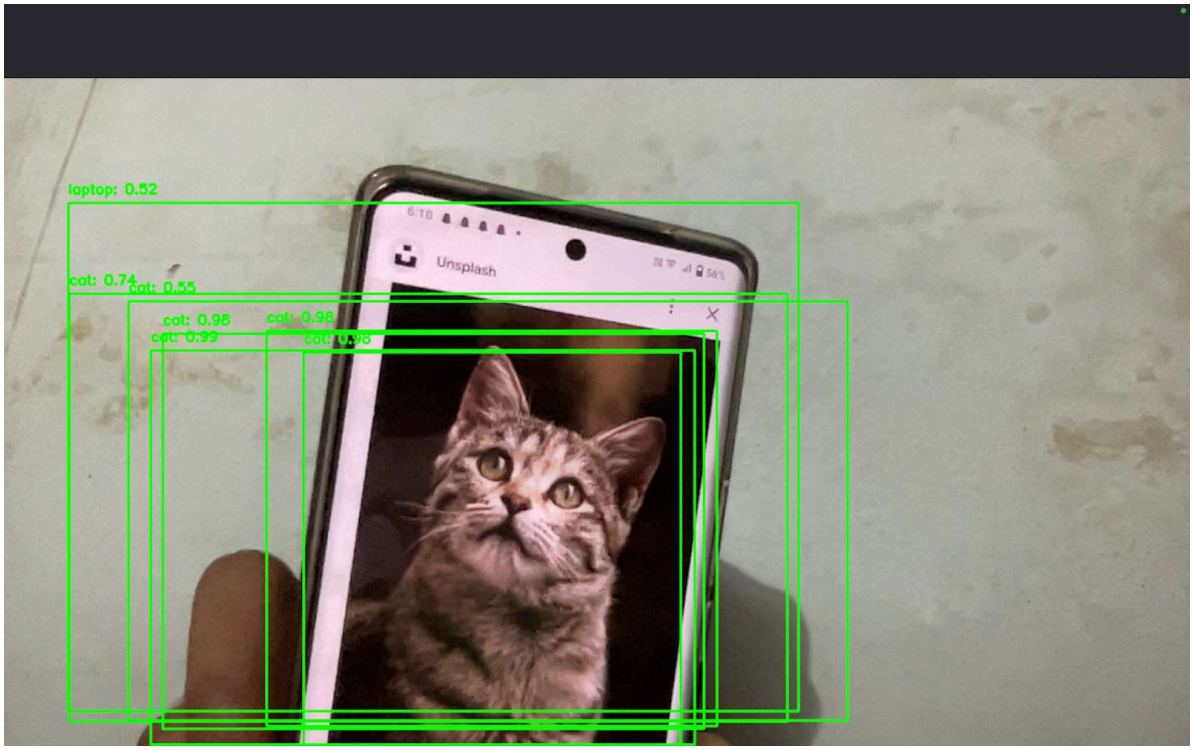




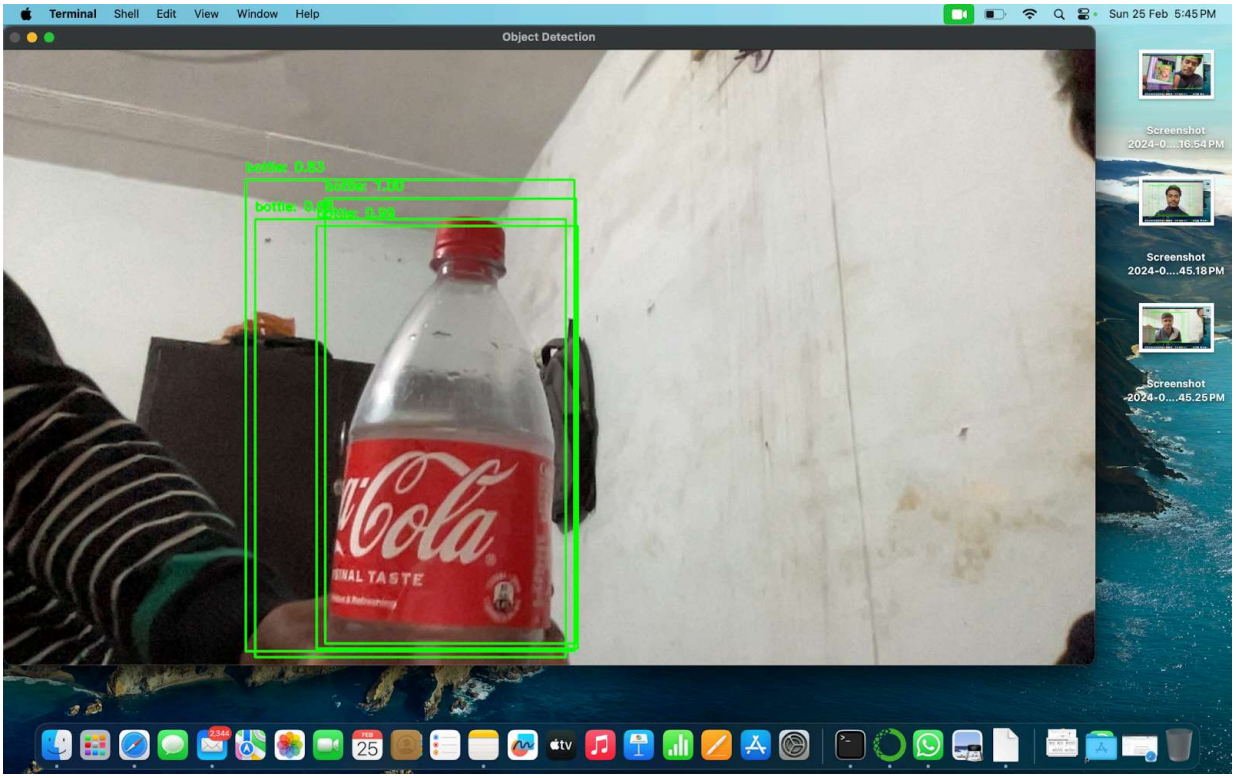

Terminal Shell Edit View Window Help
Sun 25 Feb 5:47 PM
Object Detection
person: 0.94
Screenshot 2024-0....16.54 PM
Screenshot 2024-0....45.18 PM
Screenshot 2024-0....45.25 PM
Screenshot 2024-0....45.53 PM
Screenshot 2024-0....47.13 PM

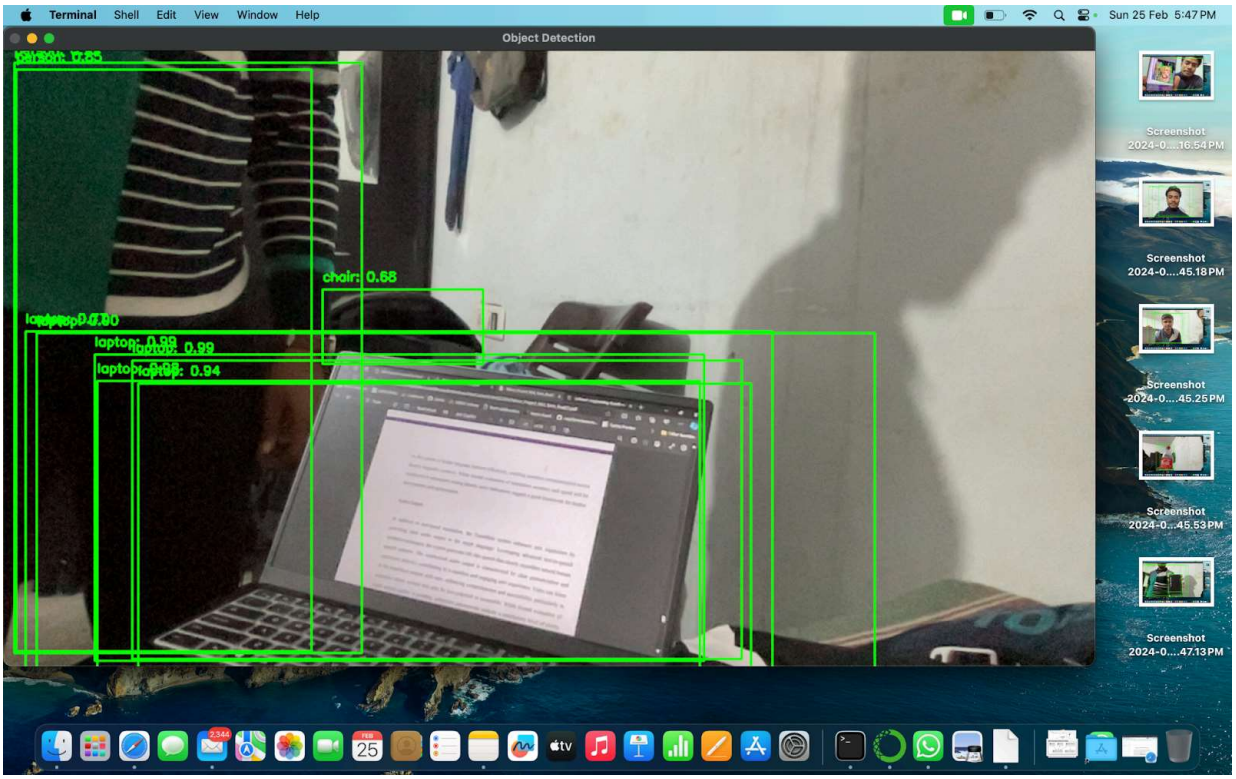
2. Detection of Animal





3. Detection of Object





References

1. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real Time Object Detection with Region Proposal Networks." In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015.
2. Joseph Redmon and Santosh Divvala. "YOLO9000: Better, Faster, Stronger." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
3. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, and Scott Reed. "SSD: Single Shot MultiBox Detector." In Proceedings of the European Conference on Computer Vision (ECCV), 2016.
4. Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN." In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.
5. Mingxing Tan, Ruoming Pang, Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020.