Topic: Study and implementation of various object detection framework using python

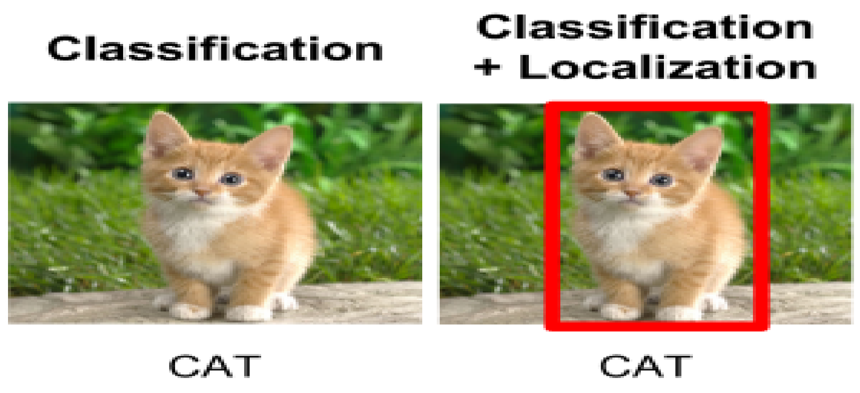
What is object detection?

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.

Understanding Object Detection

Object detection primarily aims to answer two critical questions about any image: "Which objects are present?" and "Where are these objects situated?" This process involves both object classification and localization:

- **Classification:** This step determines the category or type of one or more objects within the image, such as a dog, car, or tree.
- **Localization:** This involves accurately identifying and marking the position of an object in the image, typically using a bounding box to outline its location.



How Object Detection works?

The general working of object detection is:

1. **Input Image:** the object detection process begins with image or video analysis.

2. **Pre-processing:** image is pre-processed to ensure suitable format for the model being used.

3. **Feature Extraction:** CNN model is used as feature extractor, the model is responsible for dissecting the image into regions and pulling out features from each region to detect patterns of different objects.

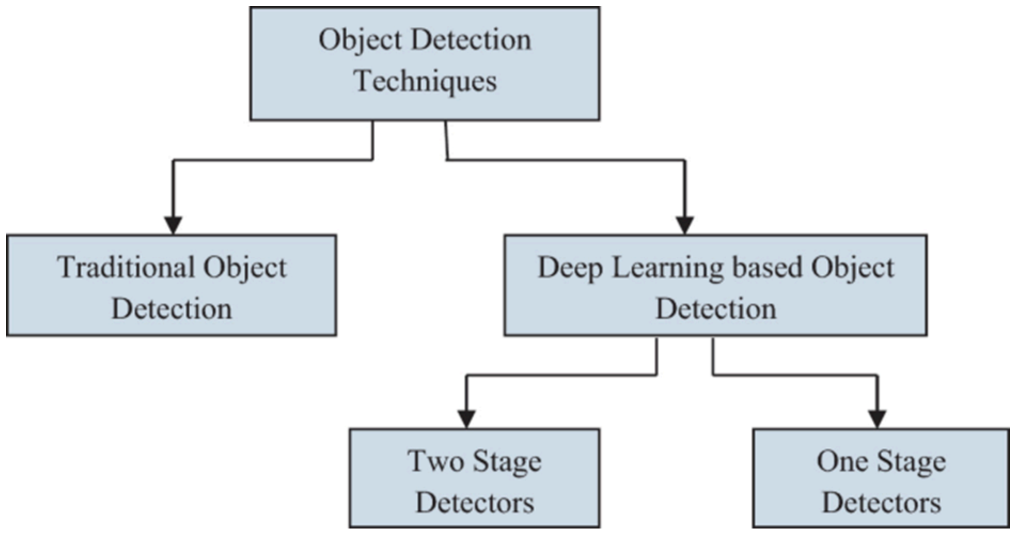
4. **Classification:** Each image region is classified into categories based on the extracted features. The classification task is performed using SVM or other neural network that computes the probability of each category present in the region.

5. **Localization:** Simultaneously with the classification process, the model determines the bounding boxes for each detected object. This involves calculating the coordinates for a box that encloses each object, thereby accurately locating it within the image.

6. **Non-max Suppression:** When the model identifies several bounding boxes for the same object, non-max suppression is used to handle these overlaps. This technique keeps only the bounding box with the highest confidence score and removes any other overlapping boxes.

7. **Output:** The process ends with the original image being marked with bounding boxes and labels that illustrate the detected objects and their corresponding categories.

Objection Detection techniques?


Object Detection Techniques
Traditional Object Detection
Deep Learning based Object Detection
Two Stage Detectors
One Stage Detectors

- **Traditional Object Detection Technique:**

  Traditional object detection refers to early computer vision techniques used to locate and identify objects in images before deep learning became mainstream.

  It typically involves:

  Manually **extracting features** (like edges or corners),
  Using **classical algorithms** to search for objects,
  Applying **machine learning classifiers** to decide what those objects are.

  - **Algorithms used-**
  - HOG + SVM for Pedestrian Detection
  - Viola–Jones Face Detector

- **Deep-Learning based Object detection Techniques:**

  - **Two-Stage Detector**

    Stage 1: Region Proposal

      - The detector searches for areas in the image that might contain an object.
      - These are called Region Proposals

    Stage 2: Classification + Refinement

  - For each proposed region:
      - It **classifies** what the object is

      - It **refines** the bounding box to fit the object more accurately.

  - **One-stage Detectors**

    One-stage detectors predict bounding boxes over the images without the region proposal step. This process consumes less time and can therefore be used in real-time applications.

  - **Algorithms used-**
      - YOLO
      - SSD (Single Shot Detector)
      - RetinaNet

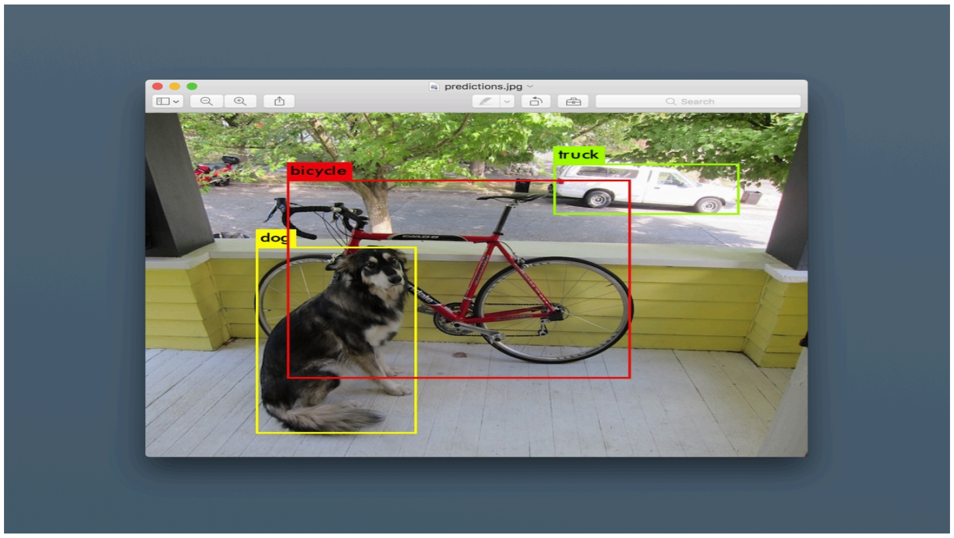
Models -

- YOLOv1 to YOLOv8
- SSD-MobileNetV2
- RetinaNet-101
- Efficient Det-D2

Algorithm used by us in this project-

YOLO:

YOLO (You Only Look Once) is a real-time object detection algorithm that has revolutionized the field of computer vision. It stands out due to its speed and accuracy, achieving both by processing an image in a single pass through a convolutional neural network. Instead of using separate stages for region proposal and classification like earlier methods, YOLO predicts bounding boxes and class probabilities simultaneously, making it exceptionally fast



Comparison of all YOLO versions-

Version	Year	Speed	Accuracy	Features
YOLO v1	2016	High	Low	No anchor boxes, basic CNN
YOLO v2	2017	High	Medium	Anchor boxes, better localization
YOLO v3	2018	Medium	High	Multi-scale detection, deeper backbone
YOLO v4	2020	Medium	Very High	Better training tricks, data augmentation
YOLO v5	2020	Very High	Very High	PyTorch, easy training, ready models
YOLO v6	2022	Very High	Very High	Optimized for real-world (industry) use
YOLO v7	2022	Highest	Very High	Efficient training and inference
YOLO v8	2023	Very High	Very High	Anchor-free, supports detection + segmentation

Reasons why not to use YOLOv1 to YOLOv7

1. **Lower Accuracy and Performance**
   Older versions (v1–v4) have less accurate object detection and slower processing compared to modern models like YOLOv8.

2. **Outdated Architecture**
   Versions before v5 are based on Darknet, which is harder to modify, lacks modern features, and has less community support now.

3. **Limited Features**
   YOLOv1 to v7 do not support instance segmentation or anchor-free detection, and most lack built-in tools for training, export, or deployment ease.

4. **No Anchor-Free Architecture**

   YOLOv8 simplifies training by removing anchor boxes; older versions still rely on them, which need careful tuning.

5. **Complex Setup and Training**

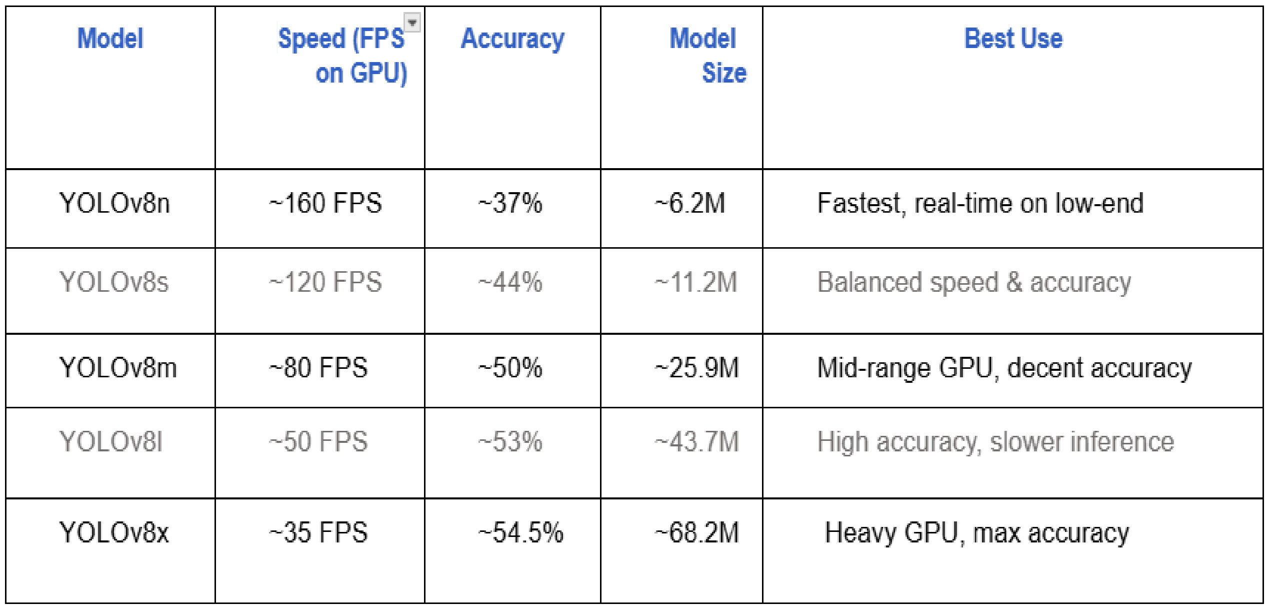
   Versions like YOLOv2 and v3 require manual configuration of anchors, input sizes, and training setups.

Yolo V8-

It offers cutting-edge performance in terms of accuracy and speed. Building upon the advancements of previous YOLO versions, YOLOv8 introduced new features and optimizations that make it an ideal choice for various object detection tasks in a wide range of applications.

Key Features of YOLOv8:

- **Advanced Backbone and Neck Architectures:** YOLOv8 employs state-of-the-art backbone and neck architectures, resulting in improved feature extraction and object detection performance.
- **Anchor-free Split Ultralytics Head:** YOLOv8 adopts an anchor-free split Ultralytics head, which contributes to better accuracy and a more efficient detection process compared to anchor-based approaches.
- **Optimized Accuracy-Speed Tradeoff:** With a focus on maintaining an optimal balance between accuracy and speed, YOLOv8 is suitable for real-time object detection tasks in diverse application areas.
- **Variety of Pre-trained Models:** YOLOv8 offers a range of pre-trained models to cater to various tasks and performance requirements, making it easier to find the right model for your specific use case.
- **Yolo v8 versions-**



Model	Speed (FPS on GPU)	Accuracy	Model Size	Best Use
YOLOv8n	~160 FPS	~37%	~6.2M	Fastest, real-time on low-end
YOLOv8s	~120 FPS	~44%	~11.2M	Balanced speed & accuracy
YOLOv8m	~80 FPS	~50%	~25.9M	Mid-range GPU, decent accuracy
YOLOv8l	~50 FPS	~53%	~43.7M	High accuracy, slower inference
YOLOv8x	~35 FPS	~54.5%	~68.2M	Heavy GPU, max accuracy

How Yolo V8 works-

1. Input Processing

The input image is resized (usually 640x640) and normalized so the model can process it efficiently.

2. Backbone

YOLOv8 uses a CSPNet-based backbone with C2f modules to extract useful features like edges, shapes, and textures from the image.

3. Neck

The neck combines features from different layers to help the model detect objects of all sizes — small, medium, and large.

4. Anchor-Free Head

Instead of using anchor boxes, YOLOv8 directly predicts the center, width, height, class, and confidence of objects. This makes detection faster and more accurate.

5. Post-processing (Non-Maximum Suppression)

It removes duplicate or overlapping boxes, keeping only the best ones with the highest confidence.

Cameras Used-

1. Intel RealSense D435i:

- **Integrated IMU with stereo depth sensing**
  - The D435i combines 3D depth data with 6DoF motion sensing, allowing real-time tracking of movement and orientation.

- **Supports SLAM and better point-cloud alignment**
  - With IMU data, it improves spatial awareness and enables mapping or localization even when the camera is in motion.

- **Global shutter sensors for accurate motion capture**
  - Captures sharp depth images even during fast movement or in low-light environments, avoiding distortion.

- **Wide field of view for reduced blind spots**
  - Covers a larger area with a single camera, making it ideal for robots or drones to navigate complex spaces.

- **SDK 2.0 for synchronized data and easy integration**
  - Intel RealSense SDK provides time-aligned IMU and depth streams, simplifying development and calibration in applications.

2. OAK-D Pro:

- **Multiple Camera Variants**
  - Offers IMX378 (12MP Auto/Fixed Focus) and OV9782 (1MP Fixed Focus); Auto-Focus for close-range, Fixed-Focus for vibration-heavy use.

- **Advanced Stereo Depth Sensing**
  - 7.5 cm baseline with 70 cm–12 m range, achieving ≤2% error under 4 m; optimized for high-accuracy 3D perception.

- **IR & Night Vision Capabilities**
  - Integrated IR dot projector (Belago 1.1) and IR LED enable reliable depth in low-light or no-light environments.

- **Integrated IMU (9-Axis BNO085)**
  - Provides motion tracking data critical for robotics, SLAM, and spatial AI applications**.**

- **Durability & Power Efficiency**
  - IP66 water-resistant body, -20°C to 50°C temp range, total power use up to 7.5W; connects via USB-C with external power support.

3. OAK-D Pro Wide

- **Wide FOV Stereo + RGB Vision**
    - Combines wide-angle active stereo depth and high-resolution color camera to deliver human-like perception, ideal for robotic vision and multi-camera reduction.

- **Low-Light & Texture-Less Scene Performance**
    - Equipped with IR dot projector and IR LED for reliable depth sensing in low-light or texture-less environments where passive stereo fails.

- **On-Device AI & Vision Processing**
    - Built on RVC2 architecture with 4 TOPS (1.4 TOPS AI), enabling real-time neural network inference and advanced computer vision functions directly on the camera.
- **Robust Build for Industrial Use**
    - Features aluminum housing, Gorilla Glass front, IP-rated durability, and mounting options (tripod + VESA), suitable for field and factory environments.

- **Advanced Tracking, Mapping & Encoding**
    - Supports 2D/3D object tracking, SLAM, RGB-depth alignment, and high-quality video encoding (4K/30FPS), making it ideal for AMRs and automation.

The projects we have completed –

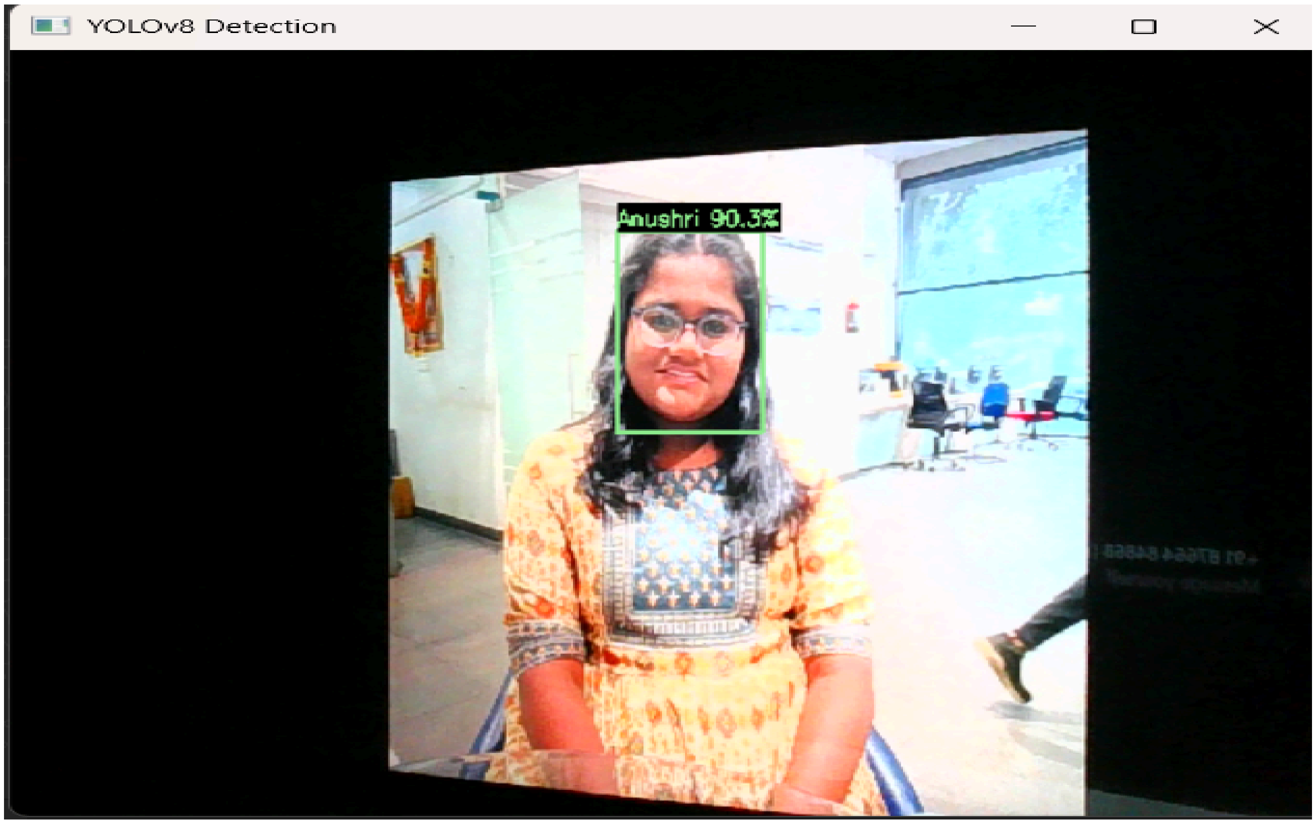
Project 1: Real-time person & object detection of (Ankita, Hriday, Anushri, Aditya) using YOLO v8 .

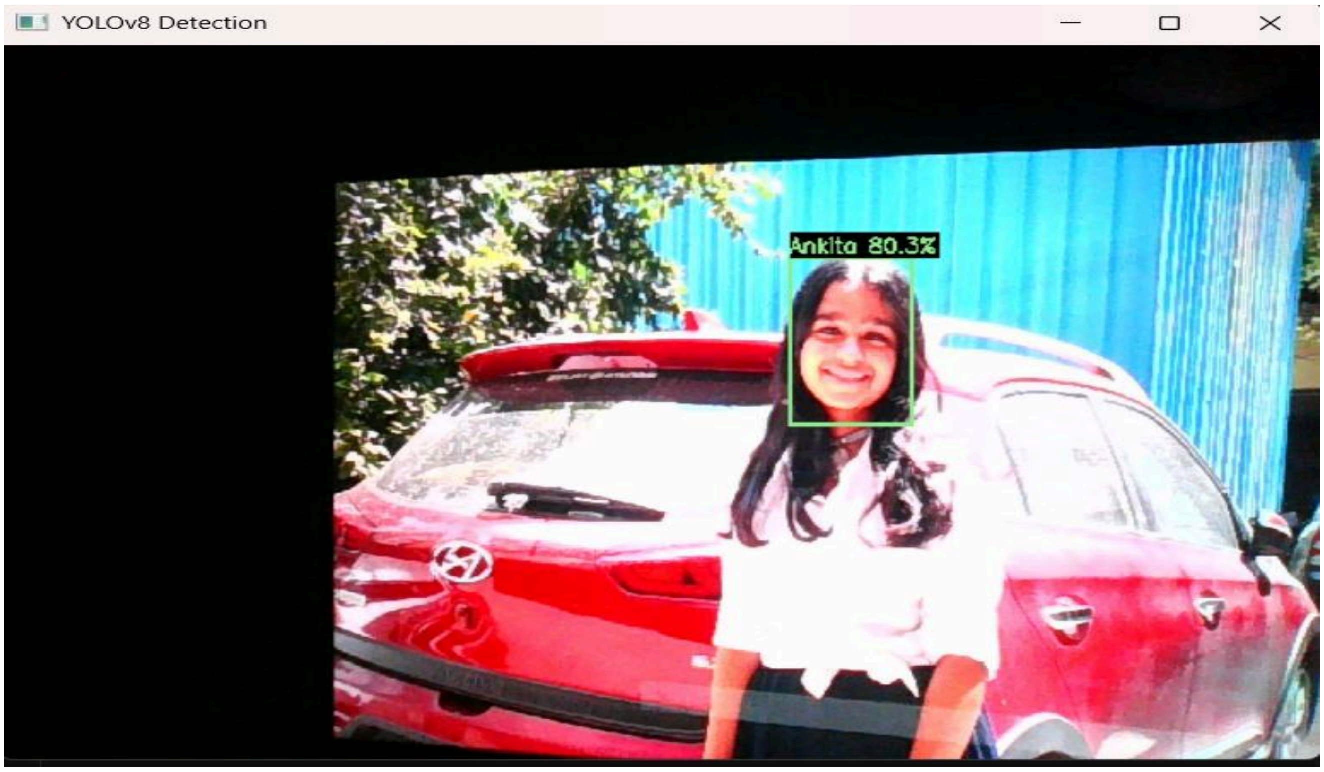
- **CODE-**

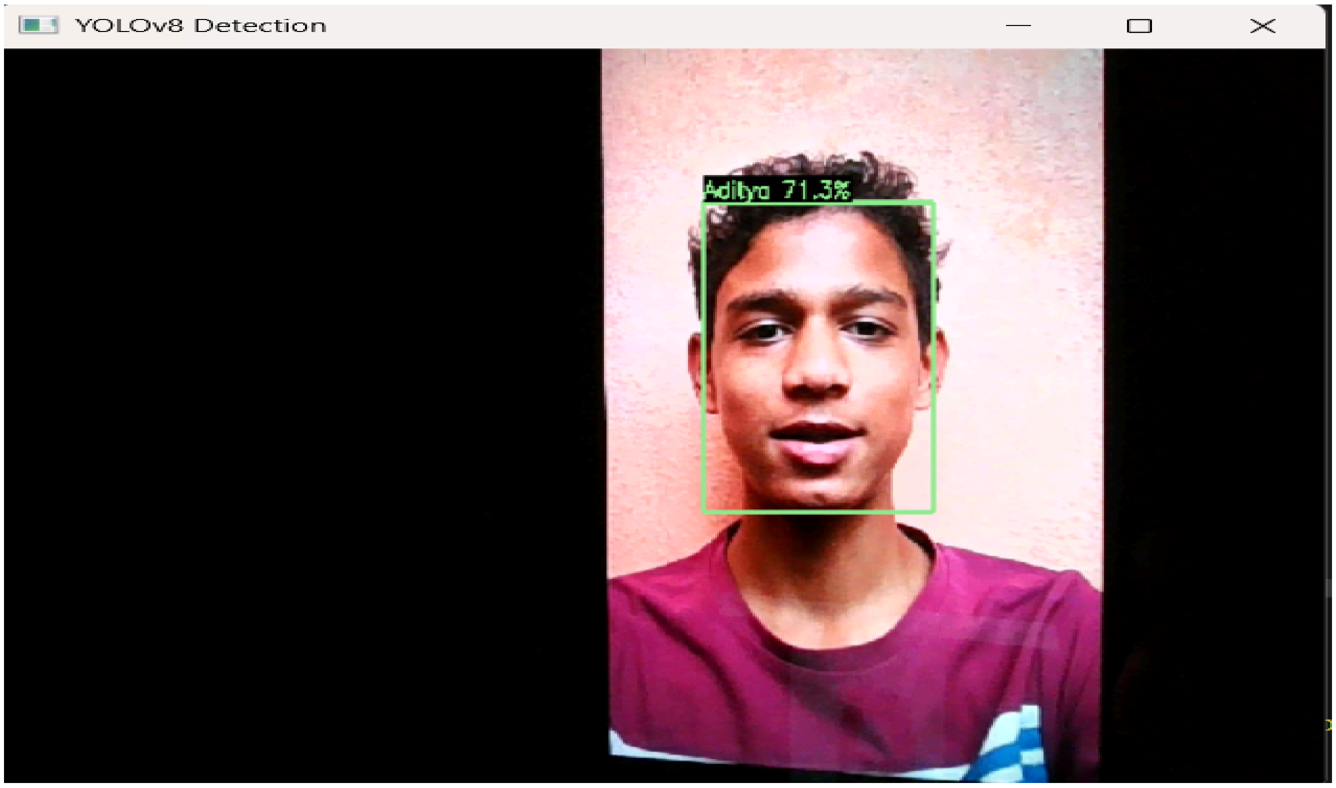
```
import cv2
import time
from ultralytics import YOLO

# Load model
model = YOLO(r"C:\Users\Anushrii\Downloads\person
detection-20250719T153046Z-1-001\person
detection\runs\detect\train\weights\best.pt")
model.conf = 0.5
names = model.names

# Open webcam
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
if not cap.isOpened():
   raise SystemExit("❌ Cannot access webcam")

# Window setup
WIN = "YOLOv8 Detection"
cv2.namedWindow(WIN, cv2.WINDOW_NORMAL)
time.sleep(1)
print("✅ Press 'q' to quit")

# Detection loop
while True:
   ok, frame = cap.read()
   if not ok:
      break
   results = model(frame, verbose=False)[0]
   for box in results.boxes:
      x1, y1, x2, y2 = map(int, box.xyxy[0])
      conf = float(box.conf)
      cls = int(box.cls)
```

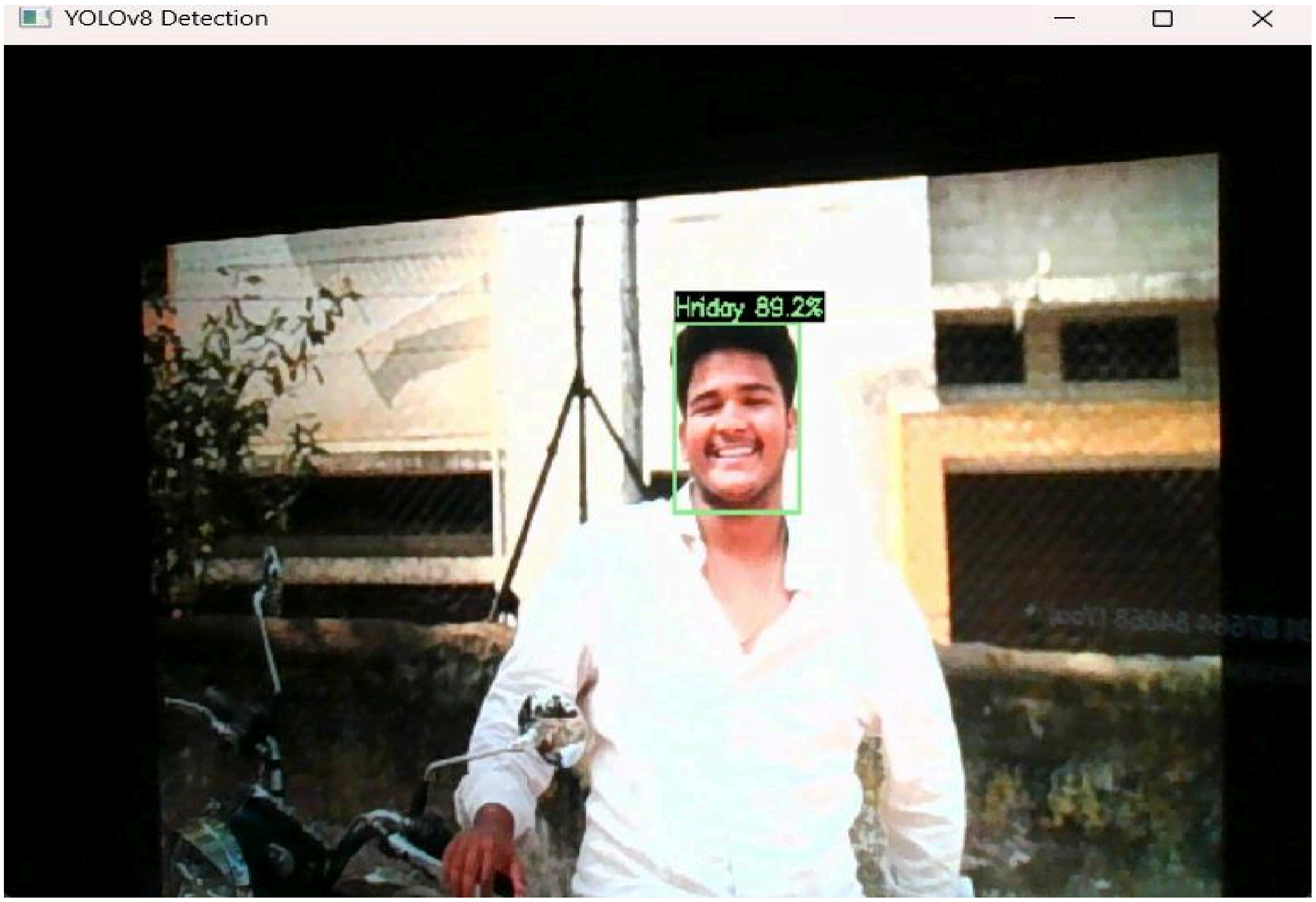
```
        label = f"{names[cls]} {conf * 100:.1f}%"
        color = (144, 238, 144)  # Sage green

        # Box
        cv2.rectangle(frame, (x1, y1), (x2, y2), color, 3)

        # Label background
      (tw, th), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 1.0, 3)
        cv2.rectangle(frame, (x1, y1 - th - 10), (x1 + tw, y1), (0, 0, 0), -1)

        # Label text with outline
        cv2.putText(frame, label, (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX,
1.0, (0, 0, 0), 5)
        cv2.putText(frame, label, (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX,
1.0, color, 2)

    cv2.imshow(WIN, frame)

    # Exit
    if cv2.getWindowProperty(WIN, cv2.WND_PROP_VISIBLE) < 1:
        break
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Cleanup
cap.release()
cv2.destroyAllWindows()
```

- **Output -**






YOLOv8 Detection
Aditya 71.3%


YOLOv8 Detection
Hriday 89.2%

- **Cameras used in this projects –**

1. **Laptop Camera Specifications**

- **Laptop Manufacturer: Microsoft (System Name: DESKTOP-D5JVH5A)**
- **Camera Name: Integrated Webcam (USB Video Device)**
- **Camera Manufacturer: Microsoft**
- **Camera Resolution: 1280 × 720 (HD)**
- **Frame Rate: ~30 FPS (suitable for real-time detection)**
- **Connection Type: Built-in USB Interface**

2.

Project 2: Real-time object detection using YOLO v8 .

- **CODE-**

Conclusion-

This project focused on the implementation of various object detection frameworks using Python, with a primary emphasis on deep learning techniques and one-stage detectors. After exploring several algorithms, we selected YOLOv8 due to its high accuracy and real-time processing speed, which makes it ideal for practical applications. YOLOv8's advanced capabilities allowed us to perform fast and precise detection, which is essential for dynamic environments and real-time monitoring systems. We also trained the YOLOv8 model using Label Studio, a powerful data labeling tool, to detect specific individuals—Ankita, Hriday, Anushri, and Aditya —demonstrating the model's flexibility in recognizing custom classes.

To support our object detection pipeline, we utilized high-performance cameras provided by our industry partner. The Intel RealSense D435i, featuring an integrated IMU and stereo depth sensing, delivered accurate SLAM support and point cloud alignment. The camera's SDK 2.0 ensured synchronized data collection and smooth integration into our workflow. Additionally, the OAK-D Pro provided advanced stereo depth sensing, IR night vision, and energy efficiency, while the OAK-D Pro Wide offered a wide field of view, strong performance in low-light or texture-less environments, and on-device AI processing. Together, the combination of cutting-edge models and powerful hardware enabled the successful creation of a reliable, real-time object detection system tailored for real-world use cases.