

**LIVE**



**OBJECT**



**DETECTION**

To design and configure a YOLOv3 object detection model by utilizing its architecture and parameters to achieve real-time object detection for 80 predefined classes or a custom dataset. This includes defining convolutional layers, shortcuts, and YOLO detection layers to enable multi-scale detection with high accuracy and efficiency

**Project Description:**

This project focuses on implementing **real-time object detection** using the YOLOv3 (You Only Look Once, version 3) algorithm. YOLOv3 is a state-of-the-art deep learning model known for its speed and accuracy in detecting and classifying multiple objects within an image or video feed. The primary goal of the project is to process live video input and detect objects by drawing bounding boxes and labeling them with their respective class names and confidence scores. **Key components of the project include: 1. Loading Pre-trained YOLOv3 Model:**

**The model is pre-trained on the COCO (Common Objects in Context) dataset, which includes 80 classes of everyday objects. The project uses the yolov3.weights file for weights, the yolov3.cfg file for architecture, and the coco.names file for object labels.**

**2.Input Preprocessing:**

**The input video frames are resized and converted into a format compatible with YOLOv3. This involves creating a blob representation of the image to normalize and optimize it for detection.**

**3.YOLO Architecture:**

**The YOLOv3 configuration file (yolov3.cfg) defines a series of convolutional layers, shortcuts (residual connections), and detection layers. This architecture enables the network to detect objects at three different scales, enhancing its performance on objects of varying sizes.**

**4. Output Processing:**

**The output from the YOLO model consists of bounding boxes, class probabilities, and confidence scores for detected objects. NonMaximum Suppression (NMS) is applied to eliminate redundant detections and retain the most accurate bounding boxes.**

**5.Real-Time Detection:**

**The processed output is displayed in real-time by drawing bounding boxes around detected objects and labeling them with their names and confidence percentages. The detection is interactive, and users can terminate the process by pressing the 'q' key.**

**6.Applications:**

* **Surveillance and security systems.**
* **Autonomous vehicles for obstacle detection.**
* **Retail and inventory management.** o **Smart cameras and interactive media**

**systems.**

**Modules Involved:**

**This project is built upon various software and hardware modules that work together to achieve real-time object detection. Below is an overview of the key modules involved:**

1. **OpenCV (Open Source Computer Vision Library)**  **Purpose:** 
   * + **Captures live video streams from the webcam.**
     + **Handles image processing tasks like resizing, blob creation, and displaying results with bounding boxes and labels.**
   * **Key Functions:** 
     + **cv2.VideoCapture(): Captures live video.** o **cv2.dnn.blobFromImage(): Prepares the input image for YOLOv3.** o **cv2.rectangle(), cv2.putText(): Draws bounding boxes and labels.**

1. **YOLOv3 Deep Learning Model** 
   * **Purpose:** 
     + **Detects objects in the input video feed by predicting bounding boxes, class probabilities, and confidence scores.**
   * **Components:** 
     + **Weight File (yolov3.weights): Pre-trained weights**

**of the YOLOv3 model.** o **Configuration File (yolov3.cfg): Defines the architecture of the YOLOv3 network.** o **Labels File (coco.names): Contains the class names (e.g., person, car, dog).**

* + **Key Operations:** 
    - **Processes input through convolutional layers to identify objects at different scales.**

1. **DNN Module (Deep Neural Network) from OpenCV**  **Purpose:**

o **Provides support for deep learning frameworks and model integration.**

* + **Key Functions:** 
    - **cv2.dnn.readNet(): Loads the pre-trained YOLOv3 model and its configuration.** o **net.setInput(): Sets the preprocessed frame as the input to the model.** o **net.forward(): Performs inference to detect objects.**

1. **Non-Maximum Suppression (NMS)** 
   * **Purpose:** 
     + **Filters overlapping bounding boxes to retain the most confident detection for each object.**
   * **Key Functions:** 
     + **cv2.dnn.NMSBoxes(): Applies NMS to reduce duplicate detections.**

1. **Live Video Processing Module** 
   * **Purpose:** 
     + **Continuously processes frames from the webcam feed in real-time.**
   * **Key Operations:** 
     + **Frame reading, resizing, and visualization.**
     + **Interactive control for stopping the detection loop.**

1. **Hardware Support Module** 
   * **Purpose:** 
     + **Provides real-time video feed via webcam or other video capture devices.**
   * **Hardware Required:** 
     + **Webcam (internal or external).**
     + **GPU (optional for improved performance).**

1. **User Interaction Module** 
   * **Purpose:** 
     + **Allows the user to terminate the process interactively.**
   * **Key Feature:** 
     + **Press 'q' to exit the detection loop and release resources.**

**Project Workflow Live Object Detection Using YOLOv3:**

The workflow of the project follows a systematic pipeline that includes data input, preprocessing, model inference, and output visualization. Below is the step-by-step flow:

1. **Input Data** 
   * **Description**:
     + The system captures real-time video frames from a webcam or other video source.
   * **Steps**:
     + Initialize the webcam using OpenCV's cv2.VideoCapture() function. o Continuously fetch video frames for processing.

1. **Preprocessing** 
   * **Description**:
     + Prepare the input video frame for the YOLOv3 deep learning model.  **Steps**:
2. **Resize Frame**:

Resize the input frame to the required dimensions (e.g., 608x608).

1. **Blob Creation**:

Use cv2.dnn.blobFromImage() to create a 4D blob for model input. This blob includes resizing, scaling, and mean subtraction.

1. **Set Input**:

Pass the blob to the YOLOv3 model using net.setInput().

**3. Model Inference**  **Description**:

o Perform object detection by passing the preprocessed frame through the YOLOv3 network.  **Steps**:

1. **Load YOLOv3 Network**:
   * Load the model configuration (yolov3.cfg) and weights (yolov3.weights).
   * Use cv2.dnn.readNet() for this task.
2. **Forward Pass**:
   * Pass the input through the YOLOv3 network using net.forward() to generate predictions.
   * Obtain detection data such as bounding boxes, class IDs, and confidence scores.

**4. Post-Processing**  **Description**:

o Refine the model’s output to display accurate detections.  **Steps**:

1. **Filter Low-Confidence Detections**:
   * Apply a confidence threshold to retain high-confidence predictions.
2. **Non-Maximum Suppression (NMS)**:
   * Use cv2.dnn.NMSBoxes() to remove overlapping bounding boxes and retain the most confident predictions.
3. **Assign Labels**:
   * Map detected class IDs to human-readable labels using the coco.names file.

**5. Visualization**

* **Description**:

o Display detected objects on the video feed with bounding boxes and labels.

* **Steps**:

1. **Draw Bounding Boxes**:
   * Use cv2.rectangle() to draw bounding boxes around detected objects.
2. **Add Labels and Confidence**:
   * Display the class label and confidence score using cv2.putText().
3. **Show Video Feed**:
   * Display the video with detections using cv2.imshow().

1. **User Interaction**  **Description**:

o Provide an interface for the user to control the application.

* + **Steps**: o Monitor for user input to exit the application (e.g., pressing the 'q' key).
    - Use cv2.waitKey() for interactive control.

1. **Termination and Resource Release**  **Description**:

o Gracefully exit the program and release all resources.

* + **Steps**:
    - Release the webcam using cv2.VideoCapture().release(). o Close all OpenCV windows with cv2.destroyAllWindows().

**How to Run the Project: Steps to Run the Live Object Detection Project**

Follow the steps below to successfully execute the live object detection system using

YOLOv3.

1. **Setup Environment**

**Install Required Libraries**

* + Ensure Python is installed (preferably Python 3.7+).
  + Install the required libraries using the following command:

bash Copy code pip install opencv-python opencv-python-headless numpy

If GPU acceleration is required, install the appropriate CUDA toolkit, cuDNN, and OpenCV with GPU support.

**Download YOLO Files**

* + Download the following YOLOv3 model files:

o **Configuration File**: yolov3.cfg o **Weights File**: yolov3.weights o **Class Names File**: coco.names

* + Place these files in a designated directory (e.g., model\_files).

1. **Prepare Project Directory**

Organize the project directory as follows:

plaintext Copy code project\_directory/ ├── model\_files/

│ ├── yolov3.cfg

│ ├── yolov3.weights

│ ├── coco.names

├── main.py

1. **Project Code:**

**Create main.py**

Paste the following code into a file named main.py: **Python Code** import cv2 import numpy as np

# Load YOLO

net =

cv2.dnn.readNet(r"C:\Users\Srinu\Downloads\yolov3.weights",r"C:\Users\Srinu\ Downloads\yolov3.cfg") with open(r"C:\Users\Srinu\Downloads\coco.names", "r") as f:

classes = [line.strip() for line in f.readlines()]

# Layer names to get the output layer layer\_names = net.getLayerNames() output\_layers = [layer\_names[i - 1] for i in

net.getUnconnectedOutLayers().flatten()]

# Set up the video capture

cap = cv2.VideoCapture(0)

# Loop through frames while cap.isOpened(): ret, frame = cap.read() if not ret: break

# Prepare the frame for YOLO height, width, channels = frame.shape

blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True,

crop=False)

net.setInput(blob) outs = net.forward(output\_layers)

# Parse YOLO's output

class\_ids = [] confidences = []

boxes = []

for out in outs: for detection in out:

scores = detection[5:] class\_id = np.argmax(scores) confidence = scores[class\_id]

if confidence > 0.5: # Adjust confidence threshold as needed # Object detected

center\_x = int(detection[0] \* width) center\_y = int(detection[1] \* height) w = int(detection[2] \* width) h = int(detection[3] \* height)

# Rectangle coordinates x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h]) confidences.append(float(confidence)) class\_ids.append(class\_id)

# Suppress overlapping boxes

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

# Draw bounding boxes and labels for i in range(len(boxes)): if i in indexes: x, y, w, h = boxes[i] label = str(classes[class\_ids[i]]) confidence = confidences[i]

color = (0, 255, 0) # Color for the bounding box

cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

cv2.putText(frame, f"{label} {int(confidence \* 100)}%", (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, color, 2)

# Display the output frame

cv2.imshow("Live Object Detection", frame)

# Press 'q' to break the loop if cv2.waitKey(1) & 0xFF == ord('q'): break

# Release resources cap.release()

cv2.destroyAllWindows()

1. **Run the Project**

**Execute the Script**

* + Open a terminal in the project directory and run:

bash Copy code python -m main.py

* + The webcam feed will open, and the detected objects will appear with bounding boxes and labels.

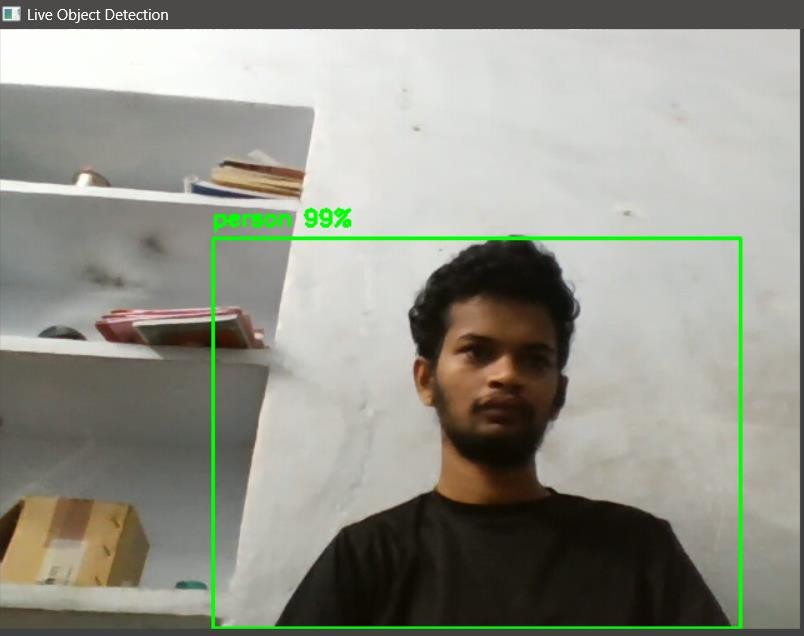
**Exit the Application**

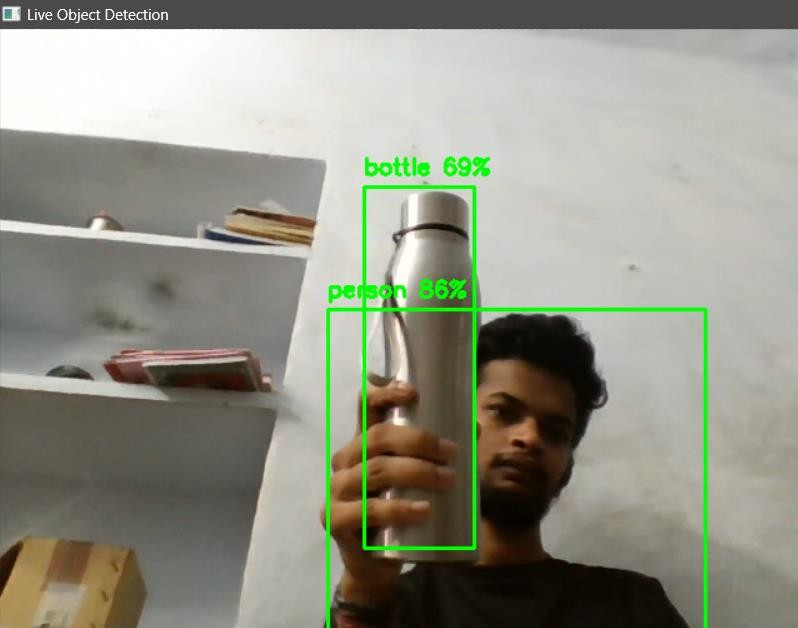
* + Press the q key to close the application and release resources.

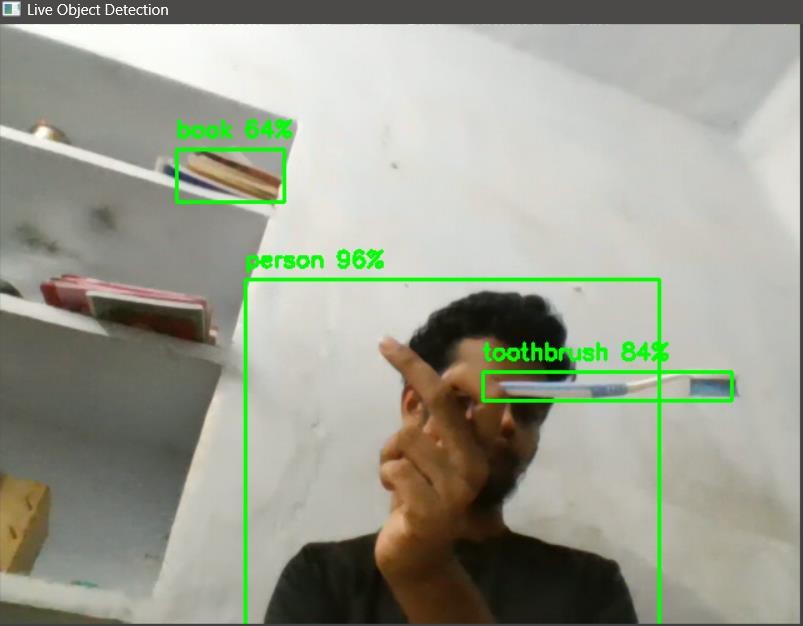
1. **Troubleshooting** 
   * If the script fails:
     + Check the file paths for yolov3.cfg, yolov3.weights, and coco.names. o Verify your OpenCV installation.
     + Ensure the webcam is properly connected and accessible.

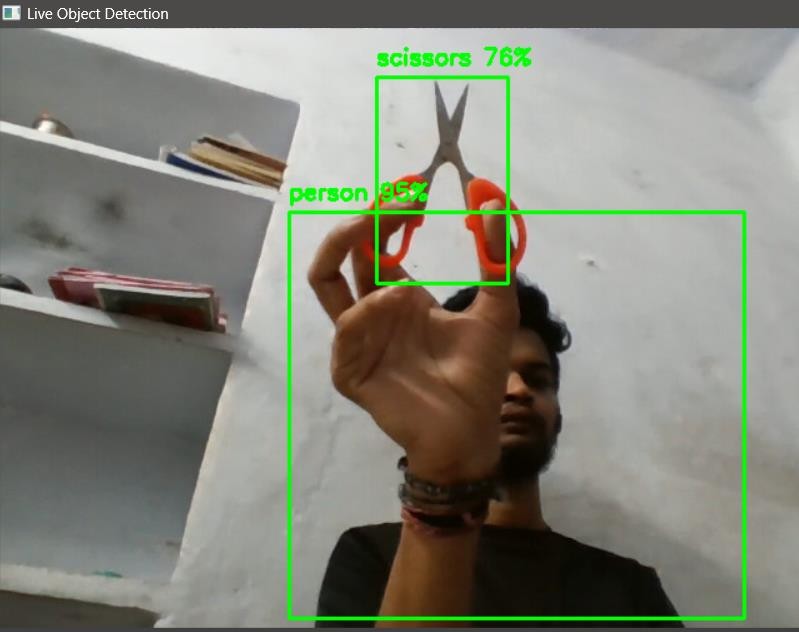
This setup ensures you can execute live object detection using YOLOv3 with ease.

**Output:**









**Conclusion:**

In this project, we have successfully implemented a **Live Object Detection System** using the YOLOv3 (You Only Look Once) model. YOLOv3 is an advanced deep learning model that allows real-time object detection with high accuracy.

The system utilizes the following components:

* **YOLOv3 Architecture**: The core model that performs the detection by classifying objects in an image frame and drawing bounding boxes around them.
* **OpenCV**: Used to handle video feed processing and visualization of results.
* **DNN (Deep Neural Network) Module**: OpenCV's DNN module loads the YOLOv3 weights and configuration for inference.
* **Non-Maximum Suppression (NMS)**: Used to filter out overlapping bounding boxes and retain the best detection.

The implementation allows real-time detection of 80 different object classes from a webcam feed. Once the system is running, it continuously processes frames, detects objects, and visualizes the results with bounding boxes and labels. This can be applied to various practical applications such as surveillance, autonomous vehicles, and retail analytics.

With the provided code and setup steps, you can run the system locally and make further customizations based on specific use cases, such as modifying the confidence threshold, adjusting the detection classes, or enhancing the model with other advanced techniques like transfer learning.

This project demonstrates the potential of YOLOv3 in real-time object detection, leveraging deep learning and computer vision technologies to achieve robust and efficient results.