

# Deeploy CV Project

Assignment : 2

Name: Vibha Narayan

Roll Number: 231141

ID : 224

Branch: PHY

Date of submission: 20/12/2024

Github Repository: 

---

## Answer 1

What I learned was that YOLOv5 was faster than SSD. SSD was capable of detecting more objects in one frame as compared to YOLOv5 and some objects like fan and charger were detected incorrect by the models as aeroplane and cell phone due to dataset limitations, model limitations like overlapping of objects which led to incorrect identification. Same goes with SSD300 Model it couldn't identify several objects like bed, notebooks because of model limitations.

Initially while using SSD 300 Model, I got an issue with loading the TensorFlow model using OpenCV's DNN module because the model or configuration file that I was using was not compatible with my OpenCV and also some training sets were missing from the model. So, later on I used different model and configuration file. The .caffemodel file provides weights pre-trained on datasets like Pascal VOC, enabling the model to detect a wide range of object classes without additional training.

SSD Model: 

YOLOv5 Model: 

## Solution Description

First the Model loads the pre-trained YOLOv5 model and then captures different sets of objects from the video file or using webcam. Using YOLO Model, each frame is processed to detect objects where objects are bounded by bounding boxes alongwith the object type and confidence scores.

**YOLO Model:**

- **Importing Libraries:**

**ultralytics:** Provides the implementation of the YOLO object detection models. Here, the YOLO class is used for model loading and inference.

**cv2:** OpenCV library for image and video processing, used to capture video frames and display the results.

- **Loading the YOLO Model:**

**YOLO('yolov5s.pt'):** Loads the pre-trained YOLOv5 model (yolov5s.pt), which is a smaller, faster model optimized for real-time use.

- **Capturing Video Input:**  
`cv2.VideoCapture(0):`  
 Initializes video capture.  
 0: Uses the default webcam as the video source. You can replace this with a file path (e.g., "video.mp4") to process a pre-recorded video.
- **Video Frame Processing Loop:**  
 A while loop processes the video frame-by-frame until it's stopped.  
`cap.read():`  
 Captures the next frame from the video source.  
 ret: Boolean indicating if the frame was successfully captured.  
 frame: The captured video frame.
- **Performing Object Detection**  
`model.predict(source=frame):`  
 Performs object detection on the current frame.  
 source=frame: Passes the captured frame to the YOLO model for inference.  
 save=False: Prevents saving the prediction results to a file.  
 results:  
 Contains the detection results, including object classes, confidence scores, and bounding box coordinates.
- **Annotating the Frame:**  
`results[0]:`  
 Accesses the first (and only) image result from the prediction.  
`.plot():`  
 Draws bounding boxes, labels, and confidence scores directly onto the frame.  
 annotated\_frame:  
 The frame with annotations added.
- **Displaying the Annotated Frame:**  
`cv2.imshow('YOLOv5 Detection', annotated_frame):`  
 Displays the annotated frame in a window titled "YOLOv5 Detection".
- **Breaking the Loop:**  
`cv2.waitKey(1):`  
 Waits for 1 millisecond for a key press.  
 & 0xFF: Ensures compatibility with 64-bit systems.  
 ord('q'): Checks if the 'q' key was pressed.  
 If 'q' is pressed, the loop breaks, and the program exits.
- **Releasing Resources:**  
`cap.release():`  
 Releases the video capture resource (webcam or video file). `cv2.destroyAllWindows():`  
 Closes all OpenCV windows.

## SSD Model:

Each frame from the video feed is preprocessed and passed through the SSD model. The model outputs bounding boxes, class IDs, and confidence scores for detected objects. The bounding boxes and labels are drawn on the frame for visualization.

- **Importing Libraries:**

cv2: OpenCV library, used for video capture, image preprocessing, and visualization.

numpy: For numerical operations such as scaling and working with arrays.

- **Loading the Model:**

MobileNetSSD\_deploy.caffemodel: Pre-trained weights of the MobileNet-SSD model.

MobileNetSSD\_deploy.prototxt: Configuration file defining the model architecture.

cv2.dnn.readNetFromCaffe: Loads the model and its configuration into OpenCV's DNN module, enabling it to perform object detection.

- **Class Labels:**

The model can detect 20 classes (besides the background class) from the Pascal VOC dataset.

These labels are used to interpret the class IDs returned by the model.

- **Capturing Video:**

Captures video input from the webcam or a pre-recorded video file.

- **Processing Each Frame:**

cv2.dnn.blobFromImage:

Converts the frame into a format suitable for the SSD model. Parameters:

0.007843: Scaling factor ( $1/255$ ) to normalize pixel values to  $[0, 1]$ . (300, 300): Resizes the image to 300x300 pixels (required by SSD300). 127.5: Mean value for input normalization (shifts pixel values to  $[-1, 1]$ ).

net.setInput(blob): Sets the preprocessed image as input to the model. net.forward(): Performs forward propagation through the network to obtain predictions

- **Processing Detections:**

The output detections is a 4D array containing:

Index  $[0, 0, i, 2]$ : Confidence score for the detection.

Index  $[0, 0, i, 1]$ : Class ID of the detected object.

Index  $[0, 0, i, 3:7]$ : Bounding box coordinates normalized to  $[0, 1]$ .

Confidence Threshold ( $> 0.2$ ): Ensures only detections with confidence  $> 0.2$

Bounding Box Scaling:

The bounding box coordinates are scaled back to the frame's dimensions (width, height).

- **Drawing Bounding Boxes:**

cv2.rectangle: Draws a green bounding box around the detected object.

cv2.putText: Adds a label and confidence score above the bounding box.

- **Displaying the Frame:**

Displays the frame with detections. Pressing the q key exits the loop.

## Differences Between YOLO and SSD Model:

### YOLO Model:

YOLO divides the input image into a grid and predicts bounding boxes and class probabilities directly for each grid cell.

- Combines object localization and classification into a single step
- Faster due to its unified architecture.
- High speed can trade off accuracy. Struggles with small objects and overlapping objects.
- Requires fewer computational resources for training.

### SSD Model:

- Uses feature maps at different scales for detecting objects of various sizes.
- Each grid cell can predict multiple bounding boxes with different aspect ratios.
- Slightly slower than YOLO, especially in older versions, due to its reliance on multi-scale feature maps.
- Often achieves better accuracy for small objects and multi-scale detection due to its use of feature maps at different resolutions.
- Better suited for datasets with diverse object sizes.

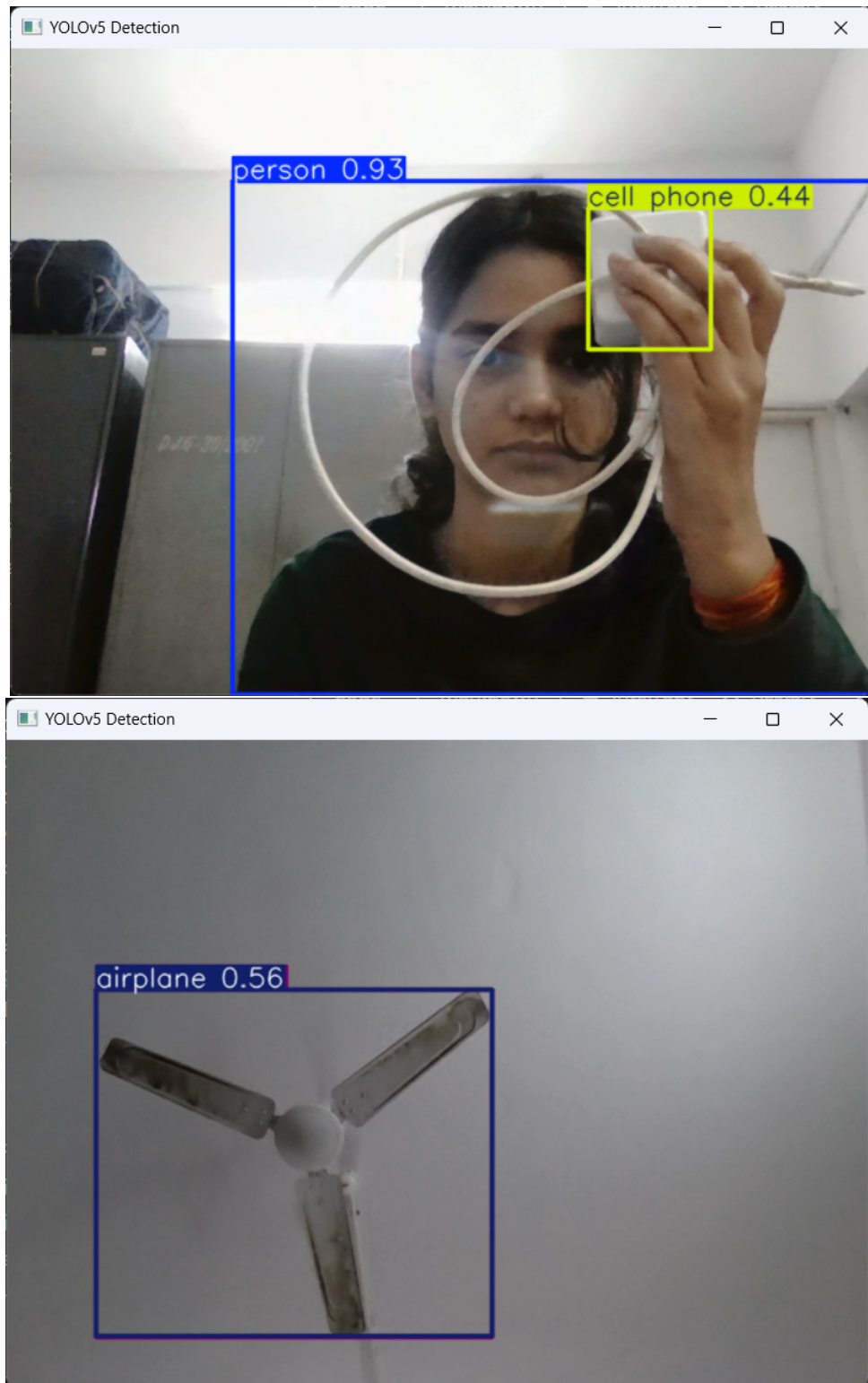


Figure 1: Image of Objects which Yolov5 didn't detect correctly

Link to Github Code: [Click here](#).

# Correct Code for Flag Problem in Assignment 1

Initially I tried to crop the area where the flag was present to focus on flag irrespective of background and dominance of red and white colour in the frame. It worked for Indonesian flag as I initially used Indonesian flag as the input but for Poland Flag the code still identified the flag as "Indonesia." This happened because the logic prioritized the red area over the white area in a way that doesn't fully account for the layout difference between the flags.

To fix this, I refined the logic further, ensuring it correctly checks the positions of red and white portions relative to each other in the image.

The code integrates color segmentation, contour-based detection, and geometric analysis for robust object detection. This method leverages the strengths of OpenCV for low-level image processing and Matplotlib for visualization, making it ideal for structured tasks like flag classification.

## Solution Description

Used Color based segmentation where first the image was converted from BGR to HSV to define precise color ranges for red and white then used Thresholding to convert binary masks for separating red and white colour.

Then used Contour detection where the largest contour in each color mask was chosen to focus on the most significant red and white areas followed by Bounding Box Computation which computes the smallest upright rectangle enclosing the largest red and white contours and lastly used geometric analysis where the relative positions of red and white colours led to identifying the flag as Indonesian or Poland.

- **Load and Convert the Image:**

The image is loaded in BGR format using OpenCV. It is then converted to RGB format for visualization using Matplotlib and to HSV format for easier color-based masking (since HSV separates color and intensity).

- **Define HSV Thresholds for Red and White:**

Red appears in two separate ranges in the HSV color space, so two masks (lower\_red1 and lower\_red2) are defined. White is defined as high value and low saturation, making it easy to mask using thresholds for bright, desaturated regions.

- **Create Masks for Red and White:**

cv2.inRange creates binary masks, identifying pixels within the specified HSV range. The red mask is formed by combining two masks (red\_mask1 and red\_mask2). A single white mask is created based on its defined range.

- **Detect Contours for Red and White Areas:**

Contours are boundaries of connected components in binary masks. Here, the largest connected regions of red and white are detected.

- **Identify the Largest Regions:**

To focus only on the most prominent areas of red and white, the largest contour is selected based on area using max.

- **Calculate the Bounding Boxes:**

Bounding boxes are calculated for the largest red and white contours, providing rectangular coordinates: (x, y, width, height).

- **Determine the Flag Based on Positions:**  
Indonesia's flag: The red area is above the white area. Poland's flag: The white area is above the red area. The Y-center of each box determines which color is above the other.
- **Visualize the Results:**  
Bounding boxes are drawn for visualization, allowing you to see the detected regions.



Figure 2: Wrong Output for Poland Flag when tried the first approach

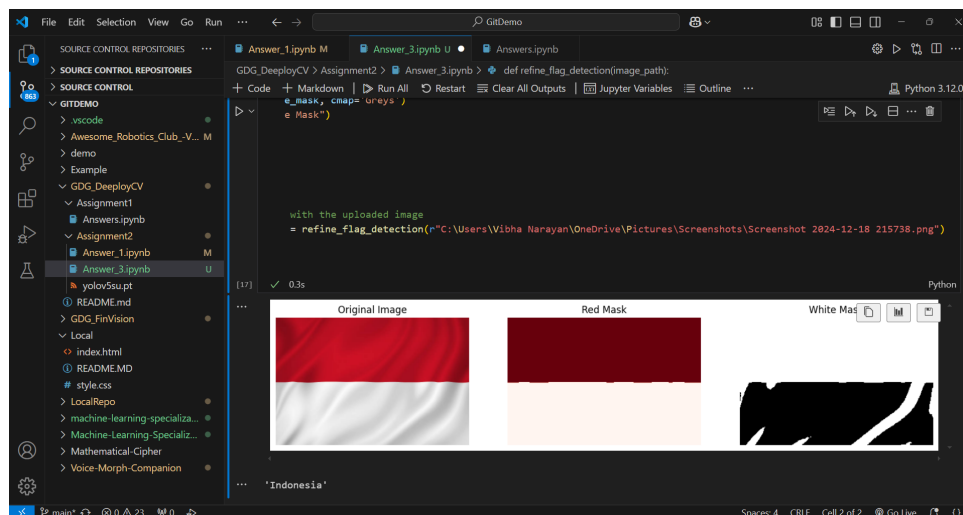


Figure 3: Indonesian Flag

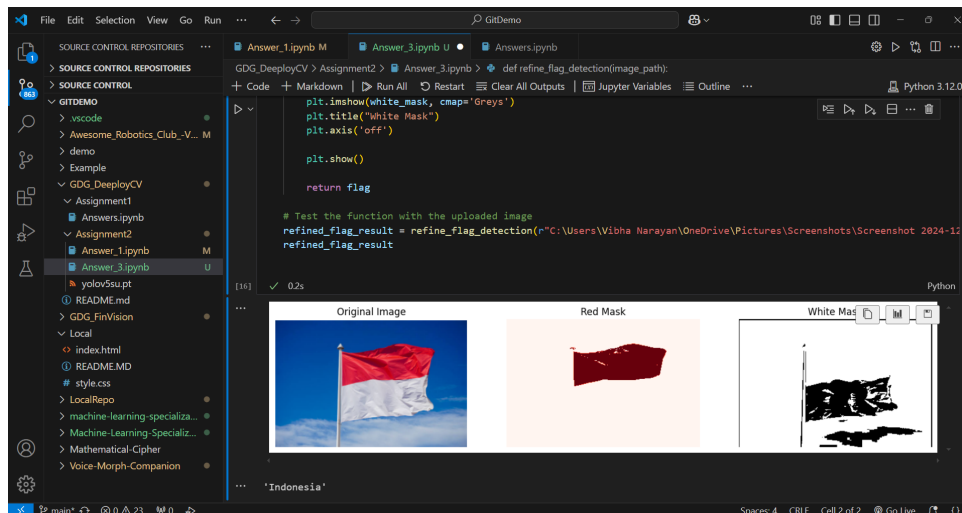


Figure 4: Indonesian Flag

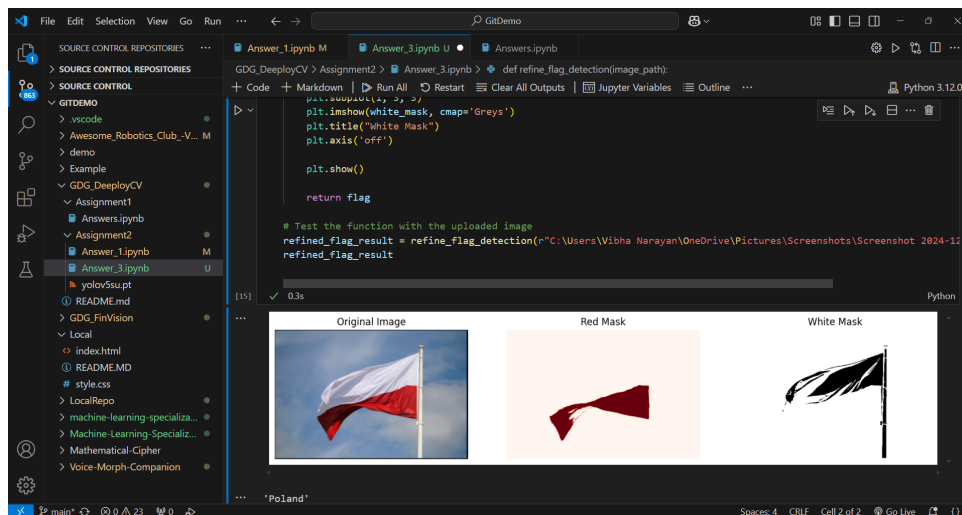


Figure 5: Poland Flag

Link to Github Code: [Click here](#).

## Answer 3

Yolov5 is not able to detect flags and neither Yolov8, it's detecting either the flag as bed, umbrella or kite.

## Solution Description

- **Importing Necessary Libraries:**
  - cv2: OpenCV for image processing.
  - numpy: Used for numerical operations like manipulating bounding box data.
  - matplotlib.pyplot: Used to visualize images with bounding boxes.
  - YOLO: The Ultralytics library to load and use the YOLOv8 model for object



detection.

- **Defining the Function:**

This function takes the file path of an image as input, detects objects using YOLOv8, and determines if the image represents a flag (Indonesia or Poland).

- **Load the YOLOv8 Model:**

Loads the YOLOv8 model. yolov8n.pt is the pre-trained YOLOv8 Nano model. If you have a custom-trained model for flags, you can replace 'yolov8n.pt' with the path to your custom model.

- **Load and Preprocess the Images:**

cv2.imread: Reads the image from the given path.

cv2.cvtColor: Converts the image from BGR (default OpenCV format) to RGB for compatibility with YOLO and visualization.

- **Perform Object Detections:**

model.predict: Runs the YOLOv8 model on the input image.

The results object contains predictions, including bounding boxes, class indices, and confidence scores.

- **Extract Bounding Boxes and Class Labels:**

x1, y1, x2, y2: Coordinates of the bounding box (top-left and bottom-right corners).

cls: Class index (e.g., 0 for "red", 1 for "white").

conf: Confidence score (e.g., how confident the model is about the detection).

- **Filter and Categorize Regions:**

Filters detections based on confidence (conf > 0.2 is ignored). Categorizes bounding boxes into red regions and white regions based on the class index.

- **Handle Missing Regions:**

If either red regions or white regions are not detected, the function returns a message indicating the flag is unrecognized.

- **Determine the Largest Regions:**

Finds the largest red and white regions by calculating the area of each bounding box.

- **Compute Centers and Decide Flag Type:**

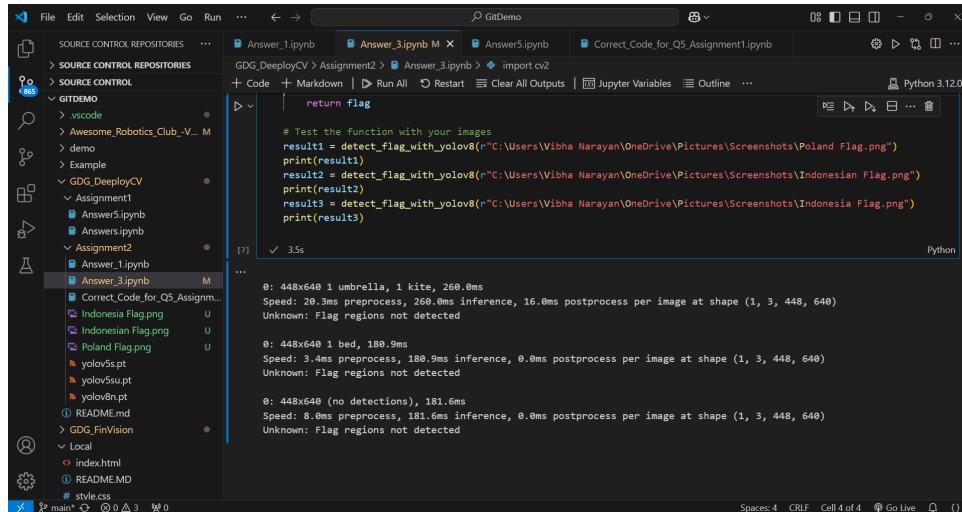
Computes the Y-coordinate of the center of the largest red and white regions. If red is higher (smaller Y-coordinate), it's Indonesia's flag. Otherwise, it's Poland's flag.

- **Drawing Bounding Boxes:**

Draws bounding boxes around the detected red and white regions:

Red boxes: Drawn in blue.

White boxes: Drawn in green.



The screenshot shows a Jupyter Notebook titled 'Answer\_3.ipynb' in a VS Code editor. The notebook is running a Python script that tests the YOLOv8 flag detection function on three images: 'Poland Flag.png', 'Indonesian Flag.png', and 'Indonesia Flag.png'. The output shows that the function correctly identifies flags in the first two images but fails to detect them in the third image, 'Indonesia Flag.png'. The output also includes performance metrics for each image, such as speed and inference time.

```
# Test the function with your images
result1 = detect_flag_with_yolov8(r"C:\Users\Vibha Narayan\OneDrive\Pictures\Screenshots\Poland Flag.png")
print(result1)
result2 = detect_flag_with_yolov8(r"C:\Users\Vibha Narayan\OneDrive\Pictures\Screenshots\Indonesian Flag.png")
print(result2)
result3 = detect_flag_with_yolov8(r"C:\Users\Vibha Narayan\OneDrive\Pictures\Screenshots\Indonesia Flag.png")
print(result3)
```

0: 448x640 1 umbrella, 1 kite, 260.0ms  
Speed: 20.3ms preprocess, 260.0ms inference, 16.0ms postprocess per image at shape (1, 3, 448, 640)  
Unknown: Flag regions not detected

0: 448x640 1 bed, 188.9ms  
Speed: 3.4ms preprocess, 188.9ms inference, 0.0ms postprocess per image at shape (1, 3, 448, 640)  
Unknown: Flag regions not detected

0: 448x640 (no detections), 181.6ms  
Speed: 8.0ms preprocess, 181.6ms inference, 0.0ms postprocess per image at shape (1, 3, 448, 640)  
Unknown: Flag regions not detected

Figure 6: Wrong detection for Flags using YOLOv8

Link to Github Code: [Click here.](#)