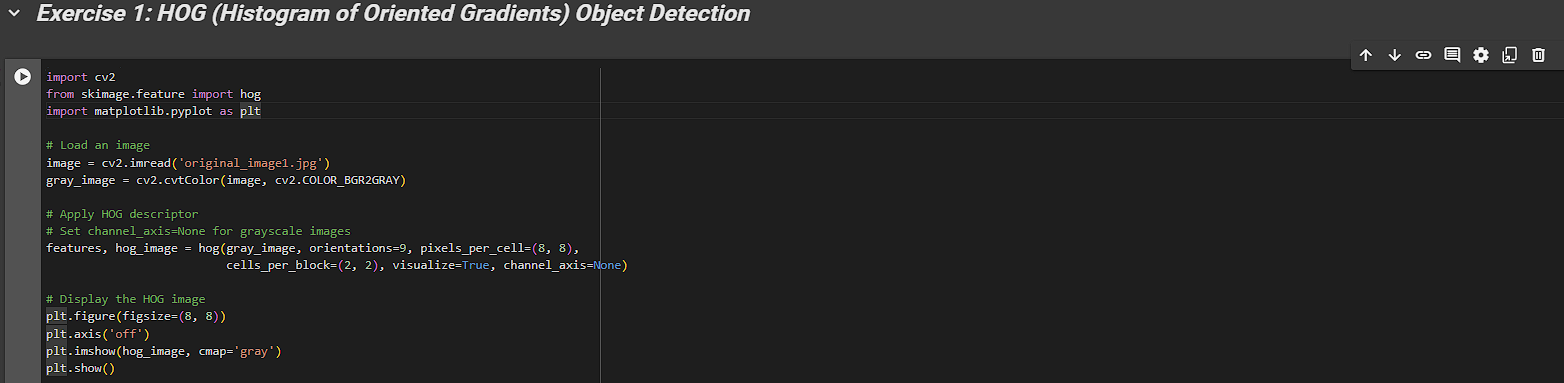
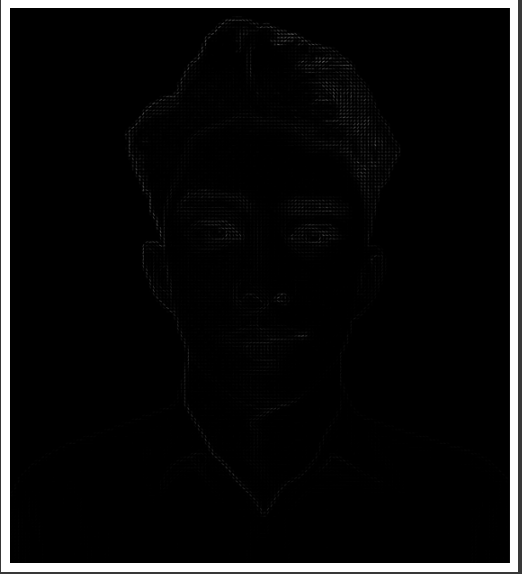
**Exercises No. 4**

**Module 2.0: Feature Extraction and Object Detection**

**Exercise 1:** HOG (Histogram of Oriented Gradients) Object Detection

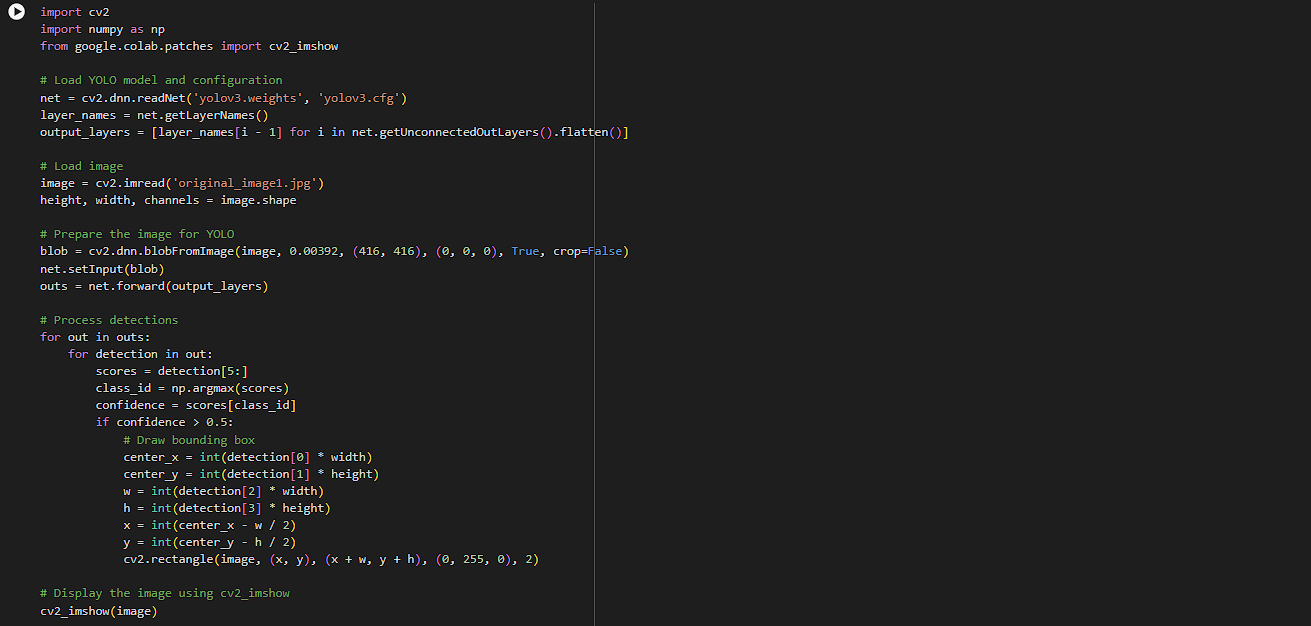
Code: 

****Output:

In the first task, I applied the Histogram of Oriented Gradients (HOG) descriptor to my image to extract and visualize its features. I started by downloading YOLOv3 weights and configuration files, although they are not used directly in this process.

I then imported necessary libraries, including OpenCV for image processing, hog from skimage.feature for HOG computation, and matplotlib.pyplot for displaying images. After that I loaded an image using OpenCV, converted it to grayscale, and applied the HOG descriptor to extract features and generate a visualization. Finally, we displayed the HOG image using Matplotlib. The resulting HOG image provides a clear representation of the gradient directions and magnitudes within the original image, highlighting its structural features. This process demonstrates how the HOG descriptor captures useful information for tasks like object detection and image classification.

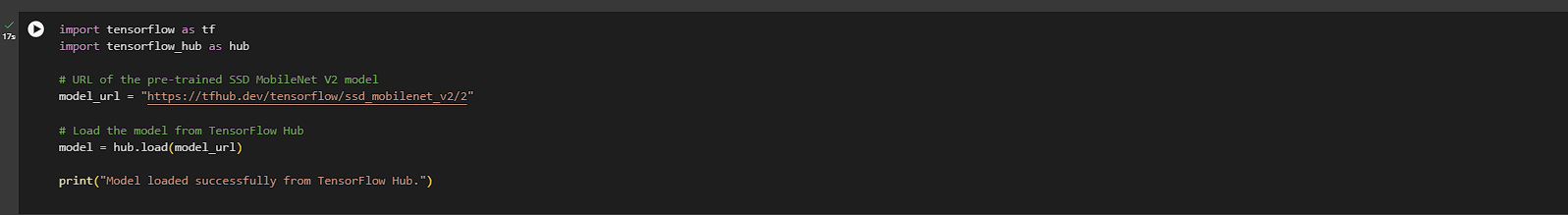
**Exercise 2:** YOLO (You Only Look Once) Object Detection

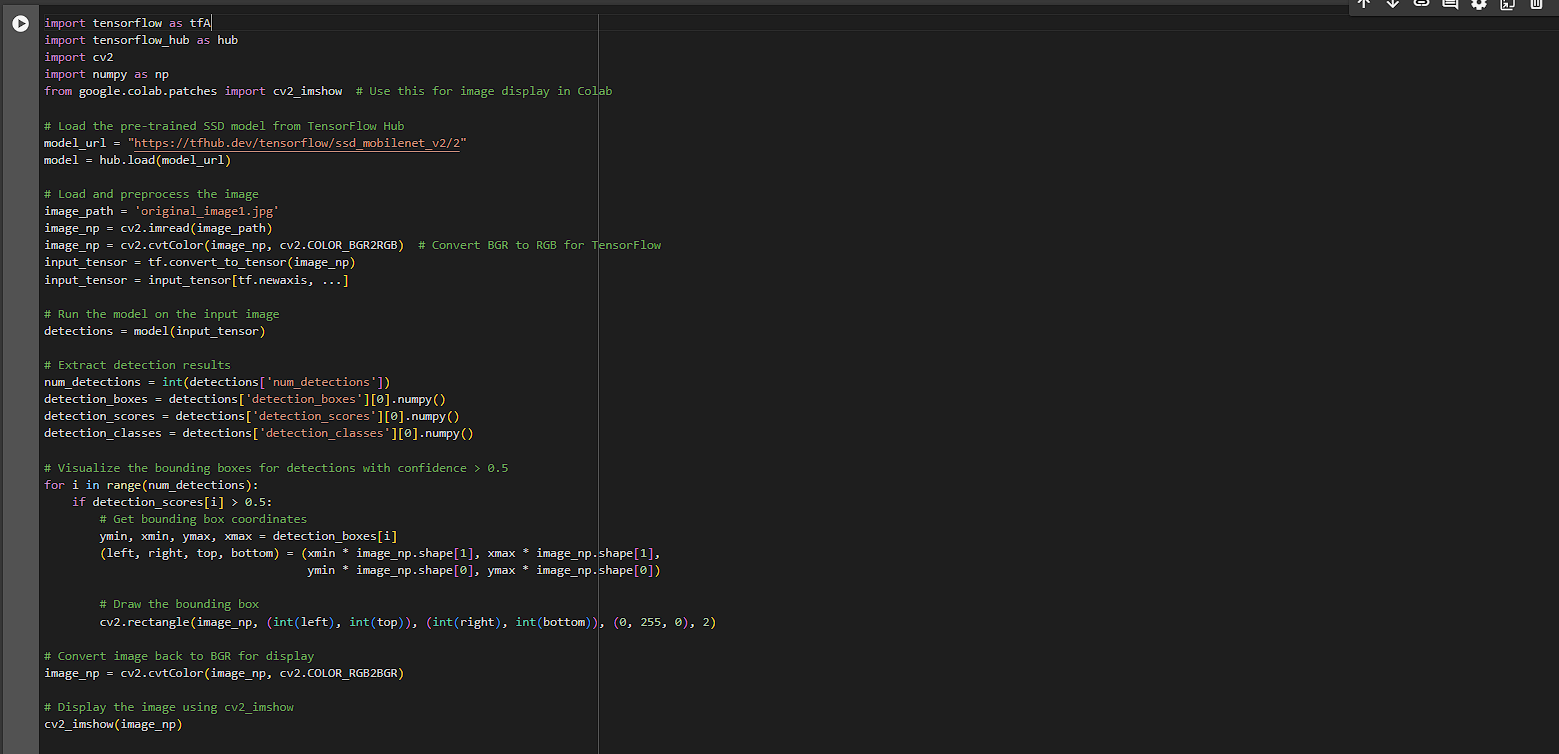
Code: 

Output:

In this task, I implemented YOLOv3 for object detection on my image. We began by importing necessary libraries, including OpenCV for image processing, NumPy for numerical operations, and cv2\_imshow from Google Colab for displaying images. We loaded the YOLO model with its weights and configuration file, retrieved the layer names, and determined the output layers. After loading an image, we obtained its dimensions and prepared it for YOLO by creating a blob from the image, which we then set as the input to the network. The forward pass of the network produced detection outputs, which we processed to identify objects with a confidence score greater than 0.5. For each detected object, we calculated the bounding box coordinates and drew rectangles around the objects on the image. Finally, we displayed the annotated image using cv2\_imshow. This process demonstrates the application of YOLOv3 for real-time object detection.

**Exercise 3:** SSD (Single Shot MultiBox Detector) with TensorFlow

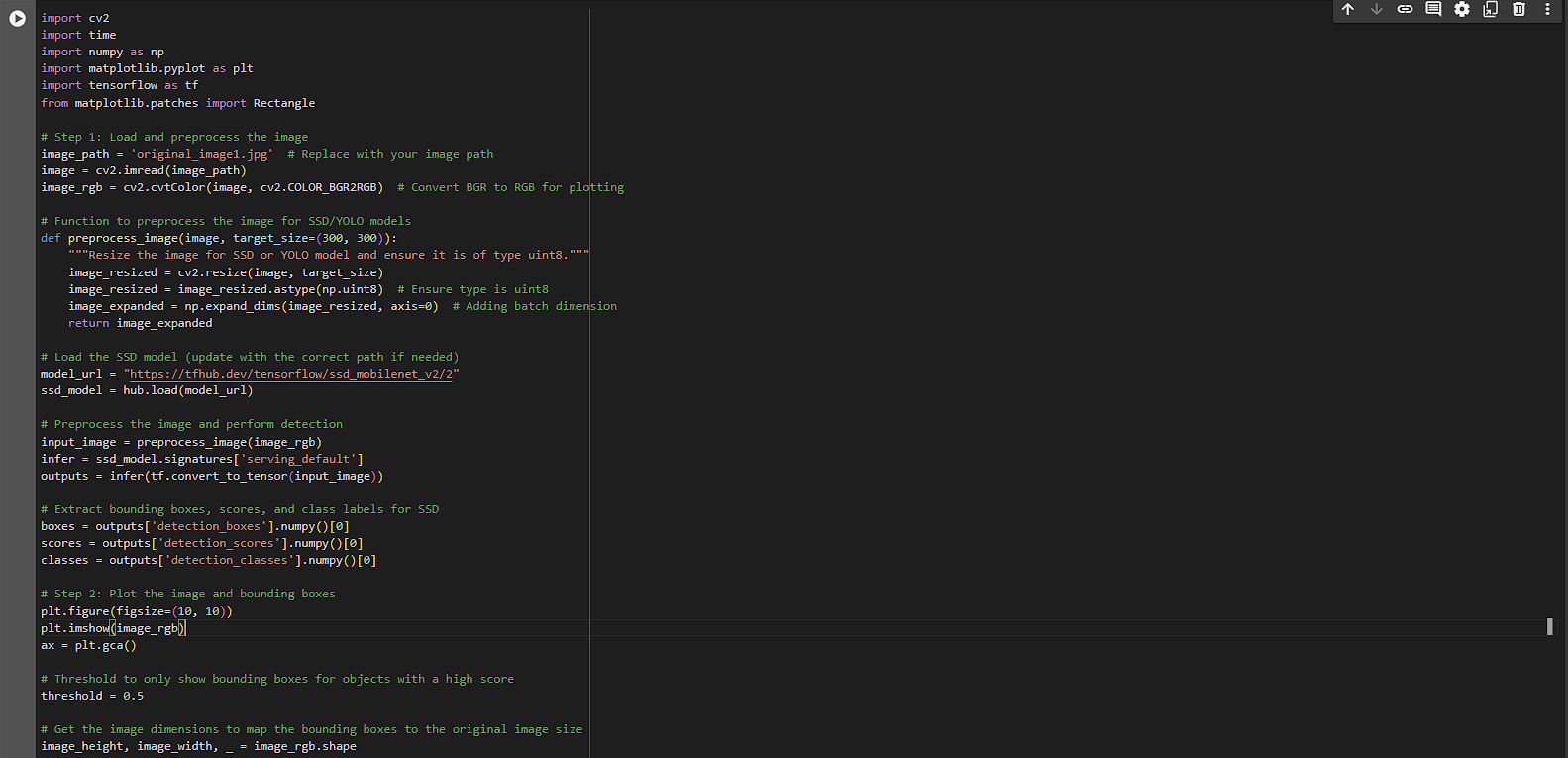
Code: 

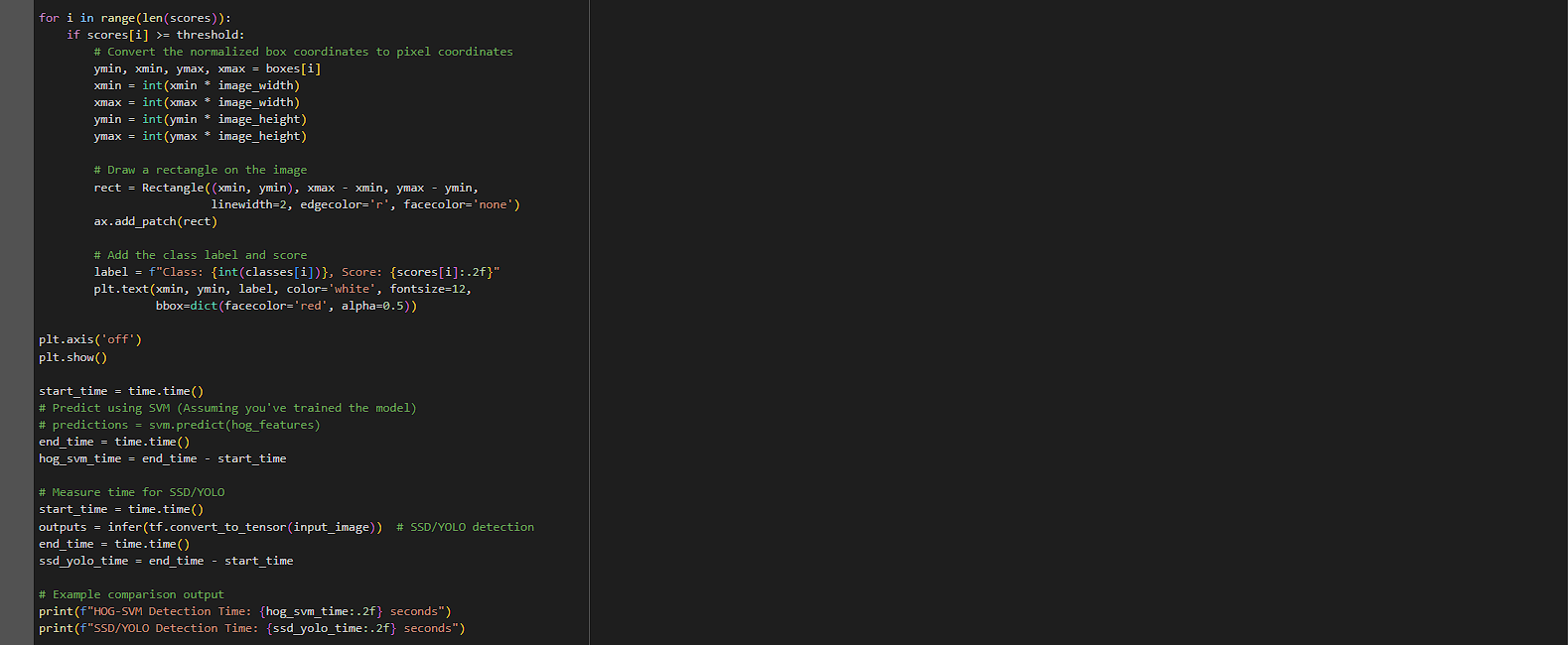
****

Output:

I used a pre-trained SSD MobileNet V2 model for object detection .First, I imported TensorFlow, TensorFlow Hub, OpenCV, NumPy, and cv2\_imshow for image display in Google Colab. I loaded the SSD MobileNet V2 model from TensorFlow Hub using its URL. After loading the model, we read an image and converted its color format from BGR to RGB, as required by TensorFlow then prepared the image as an input tensor and ran the model on this input. Finally, I converted the image back to BGR format for display and used cv2\_imshow to visualize the annotated image. This activity demonstrates the use of SSD MobileNet V2 for efficient object detection.

**Exercise 4:** Traditional vs. Deep Learning Object Detection Comparison

Code: 



Output:

In the last task, we performed object detection on an image using a pre-trained SSD MobileNet V2 model from TensorFlow Hub and measured the detection time for both HOG-SVM and SSD/YOLO models. We started by importing the necessary libraries, including OpenCV for image processing, TensorFlow for model inference, Matplotlib for visualization, and NumPy for numerical operations. We loaded an image, converted it to RGB format, and defined a preprocessing function to resize the image and prepare it for the SSD model.

Next, we loaded the SSD model and preprocessed the input image. We ran the model on the preprocessed image to obtain the detection outputs, including bounding boxes, scores, and class labels. We then visualized the detection results by plotting the image with bounding boxes around detected objects that had a confidence score above a defined threshold (0.5). Bounding boxes were drawn using Matplotlib's Rectangle patches, and class labels with scores were annotated on the image.

Finally, we measured the detection times for both HOG-SVM (assuming a pre-trained SVM model for HOG features) and SSD/YOLO models. This involved running the SSD model inference and recording the time taken for detection. The results were printed to compare the performance of the two detection methods. This activity demonstrates the process of loading a pre-trained object detection model, preprocessing the input, performing detection, visualizing the results, and comparing the detection times for different methods.