**Exer4**

**HOG (Histogram of Oriented Gradients) Feature Extraction**

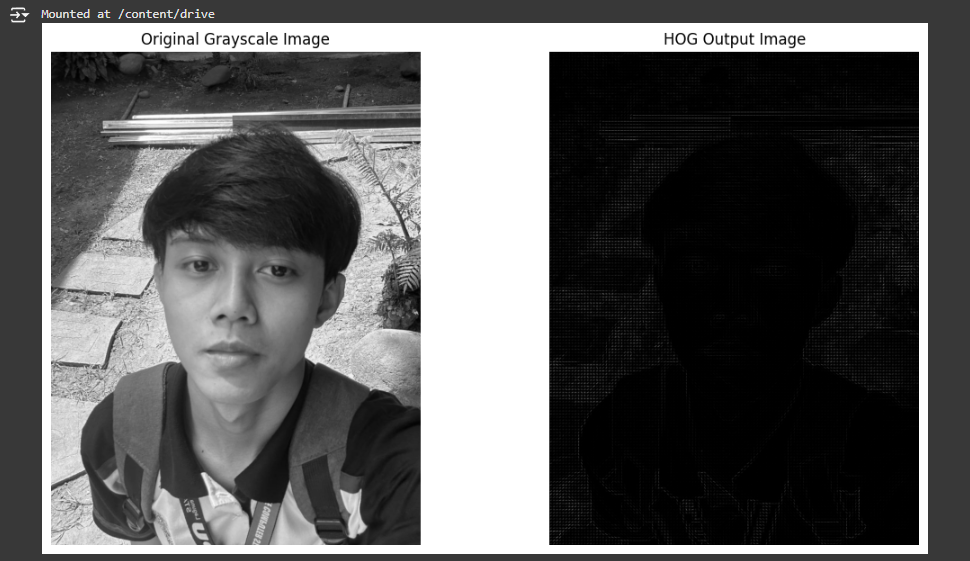
**Purpose**: To extract HOG features from an image and visualize the gradient orientations, which capture the structure and edges within the image.

**Process**:

1. **Mount Google Drive**: Allows access to files stored on Google Drive. This is essential if your image is saved on Google Drive.
2. **Load the Image**: Reads the specified image from Google Drive.
3. **Convert to Grayscale**: Converts the image to grayscale, as HOG operates on single-channel images.
4. **Apply HOG Descriptor**: Computes the HOG features, returning both the feature descriptor (HOG feature vector) and a visualization image of gradient orientations.
   * **Orientations**: Divides the gradient directions into 9 bins.
   * **Pixels per Cell**: Each cell in the image grid is 8x8 pixels.
   * **Cells per Block**: Each normalization block consists of a 2x2 cell grid.
   * **Block Normalization**: Uses the 'L2-Hys' method to normalize gradients within each block, making the feature descriptor more robust to lighting changes.
5. **Set Up Display**: Arranges two subplots to show the original grayscale image and the HOG visualization side by side.
6. **Display the Result**: Shows both the original grayscale image and the HOG visualization, illustrating the detected gradient structures in the image.

**Expected Output**: The output is a side-by-side comparison:

* **Left Image**: The original grayscale image.
* **Right Image**: The HOG visualization image, showing the gradient orientation and magnitude within each cell, highlighting edges and structural details in the image.



**YOLO Object Detection**

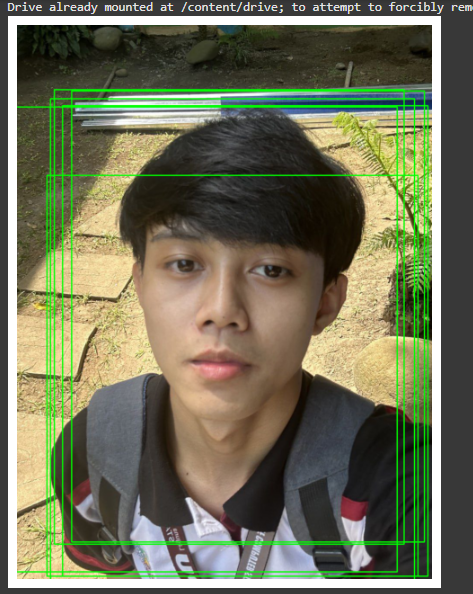
**Purpose**: To detect objects in an image using the YOLO (You Only Look Once) deep learning model, which can identify multiple objects in real time with high accuracy.

**Process**:

1. **Import Libraries**:
   * Imports necessary libraries including OpenCV for image processing, NumPy for numerical operations, and Matplotlib for displaying the image.
   * Mounts Google Drive in Google Colab to access the required YOLO files and image if stored there.
2. **Mount Google Drive**:
   * drive.mount('/content/drive'): Mounts Google Drive to access the weights, configuration files, and image if they are stored in Google Drive.
3. **Set Paths for YOLO Files**:
   * **weights\_path**: Path to the YOLO weights file (yolov3.weights).
   * **config\_path**: Path to the YOLO configuration file (yolov3.cfg).
4. **Load YOLO Model**:
   * net = cv2.dnn.readNet(weights\_path, config\_path): Loads the YOLO model with the specified weights and configuration file, enabling object detection.
5. **Get Output Layer Names**:
   * Retrieves the names of the output layers required to obtain the detection results.
   * **Output Layers**: YOLO’s architecture has several layers, but only the output layers contain detection information.
6. **Load and Prepare the Image**:
   * Loads the specified image from Google Drive using OpenCV.
   * Converts the image dimensions (height, width, channels) for use in YOLO.
   * Creates a blob from the image, which resizes and normalizes it to meet YOLO’s input requirements:
     + **blob = cv2.dnn.blobFromImage(...)**: Converts the image into a blob with specific parameters:
       - **1/255.0**: Normalizes pixel values to [0, 1].
       - **(416, 416)**: Resizes the image to 416x416 pixels, YOLO’s default input size.
       - **swapRB=True**: Switches the red and blue channels to RGB.
7. **Run Forward Pass for Object Detection**:
   * **net.forward(output\_layers)**: Runs a forward pass through the YOLO network using the preprocessed image to get detection results.
8. **Loop Through Each Detection**:
   * Iterates over each detection result:
     + **Detection Scores**: Extracts the confidence scores for each object class.
     + **Class ID**: Identifies the class with the highest score as the detected object class.
     + **Confidence**: Retrieves the highest confidence score for the detected class.
     + **Confidence Threshold**: Only keeps detections with a confidence score above 0.5 (50%) to reduce false positives.
9. **Draw Bounding Boxes**:
   * Converts the normalized coordinates of the bounding box to actual pixel coordinates.
   * **cv2.rectangle(...)**: Draws a green bounding box around the detected object.
10. **Convert Image Format**:
    * Converts the image from BGR to RGB color format to display it correctly using Matplotlib.
11. **Display the Image**:
    * Displays the image with detected objects highlighted with bounding boxes.

**Expected Output**:

* The output is the original image displayed with bounding boxes around detected objects, each representing an object identified by YOLO.
* The bounding boxes are drawn in green, marking the detected objects with high confidence (above 50%).



**SSD MobileNet Object Detection with TensorFlow Hub**

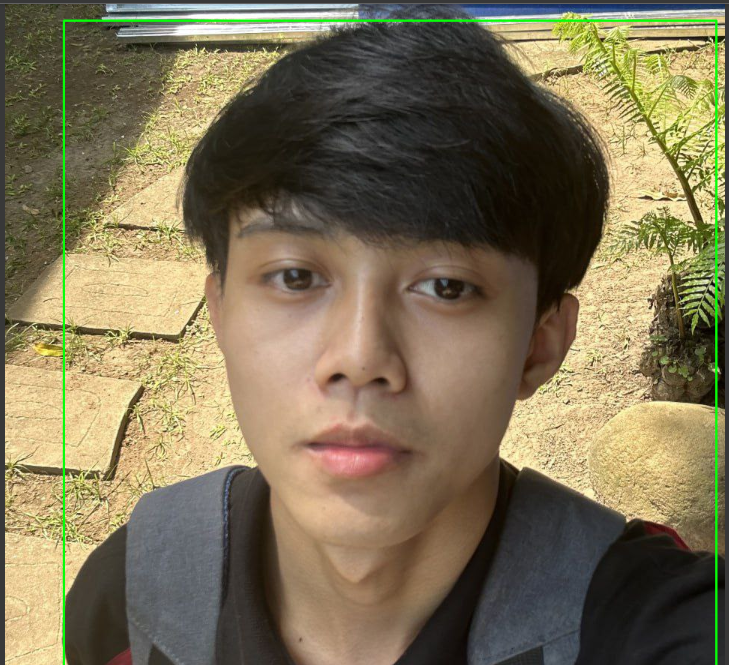
**Purpose**: To detect objects in an image using the SSD MobileNet V2 model from TensorFlow Hub, which provides efficient and accurate object detection trained on the COCO dataset.

**Process**:

1. **Import Necessary Libraries**:
   * Imports TensorFlow and TensorFlow Hub for model loading and inference.
   * Uses OpenCV for image processing and displays images with cv2\_imshow, which is compatible with Google Colab.
   * Mounts Google Drive to access images stored there.
2. **Mount Google Drive**:
   * drive.mount('/content/drive'): Allows access to files stored on Google Drive, such as the image for object detection.
3. **Load SSD MobileNet Model**:
   * hub.load(model\_url): Loads the pre-trained SSD MobileNet V2 model from TensorFlow Hub. This model is trained on the COCO dataset and can detect 90 common objects.
   * **Model URL**: https://tfhub.dev/tensorflow/ssd\_mobilenet\_v2/2.
4. **Load and Preprocess the Image**:
   * **Load Image**: Reads the image specified from Google Drive.
   * **Convert to RGB**: Converts the image from BGR (OpenCV format) to RGB, as TensorFlow models typically expect images in RGB format.
   * **Convert to Tensor**: Transforms the image into a TensorFlow tensor and adds a batch dimension to match the input shape expected by the model.
5. **Run Object Detection**:
   * detector(input\_tensor): Runs inference on the image and returns detection results, which include bounding boxes, scores, and class labels.
   * **Bounding Boxes**: Contain normalized coordinates for each detected object’s position in the image.
   * **Confidence Scores**: Provide the confidence level of each detection.
6. **Draw Bounding Boxes**:
   * Loops over the detection results, only drawing boxes for detections with a confidence score greater than 0.5 (50%).
   * **Coordinate Transformation**: Converts bounding box coordinates from normalized values (0–1) to actual pixel values based on the image dimensions.
   * **Draw Bounding Box**: Uses cv2.rectangle to draw a green bounding box around each detected object.
7. **Display the Image**:
   * **Convert to BGR**: Converts the image back to BGR format for OpenCV display.
   * **Display with cv2\_imshow**: Uses cv2\_imshow to show the image with detected objects, as cv2.imshow is not supported in Colab.

**Expected Output**:

* The output is the original image displayed with bounding boxes around objects detected by the SSD MobileNet model. Bounding boxes are drawn in green and highlight the locations of detected objects in the image with high confidence (above 50%).



### ****Object Detection using SSD MobileNet with TensorFlow****

**Purpose**: To detect objects in an image using the SSD MobileNet model and visualize bounding boxes with class labels and confidence scores. Additionally, it measures and compares the processing time of SSD MobileNet with a placeholder HOG-SVM model.

**Process**:

1. **Import Necessary Libraries**:
   * Imports libraries for image processing, visualization, object detection, and time measurement.
   * Mounts Google Drive to access images and models stored there.
2. **Mount Google Drive**:
   * drive.mount('/content/drive'): Mounts Google Drive to access files, such as the image and model if they are stored there.
3. **Load and Preprocess the Image**:
   * **Load Image**: Reads the specified image using OpenCV.
   * **Convert to RGB**: Converts the image from BGR (default in OpenCV) to RGB for compatibility with TensorFlow models.
   * **Preprocess Function**:
     + preprocess\_image(image, target\_size=(300, 300)): Resizes the image to a standard input size (300x300 pixels) and adds a batch dimension, making it compatible with SSD/YOLO models.
4. **Load the SSD MobileNet Model**:
   * **Model Path**: Specifies the path to the pre-trained SSD MobileNet model in TensorFlow SavedModel format.
   * ssd\_model = tf.saved\_model.load(ssd\_model\_path): Loads the model for inference.
   * infer = ssd\_model.signatures['serving\_default']: Sets up the model's detection function.
5. **Run Object Detection**:
   * **Model Inference**: Runs the SSD MobileNet model on the preprocessed image to obtain detections.
   * **Extract Results**:
     + **Bounding Boxes**: Retrieves normalized bounding box coordinates.
     + **Scores**: Retrieves confidence scores for each detected object.
     + **Class Labels**: Retrieves class labels for detected objects.
6. **Display Image with Bounding Boxes**:
   * **Plot Image**: Displays the original image with bounding boxes and labels.
   * **Loop through Detections**:
     + Filters detections by a confidence threshold (0.5) to keep high-confidence predictions.
     + Converts normalized bounding box coordinates to pixel coordinates based on the image dimensions.
     + **Draw Bounding Box**: Draws a red bounding box around each detected object with Rectangle.
     + **Add Label**: Adds text indicating the class ID and confidence score for each detected object.
   * **Display**: Hides the axis and shows the processed image with bounding boxes.
7. **Time Measurement**:
   * **HOG-SVM Detection Time**: Placeholder code simulates the time measurement for HOG-SVM detection (replace this with actual HOG-SVM model code if available).
   * **SSD Detection Time**: Measures the time taken by SSD MobileNet to perform object detection on the image.
   * Displays both times for comparison.
8. **Display Timing Results**:
   * Prints the detection time for both HOG-SVM (placeholder) and SSD MobileNet, allowing a performance comparison.

**Expected Output**:

1. **Visual Output**:
   * The image is displayed with bounding boxes around detected objects, each labeled with the class ID and confidence score.
2. **Timing Results**:
   * The terminal displays the time taken for object detection by both HOG-SVM (placeholder) and SSD MobileNet. This provides insight into the efficiency of SSD MobileNet compared to traditional HOG-SVM-based detection.

### Example of Expected Output

* **Image with Bounding Boxes**: The image should display bounding boxes in red around detected objects (e.g., cars, people) with text annotations showing the class ID and confidence score.
* **Timing Comparison**: Printed values for detection times of HOG-SVM and SSD MobileNet, which highlight the efficiency of SSD MobileNet for object detection.

