Exercises No. 4			
Topic:	**Module 2.0: Feature Extraction and Object Detection**	**Week No.**	8-9
Course Code:	**CSST106**	**Term:**	1st Semester
Course Title:	**Perception and Computer Vision**	**Academic Year:**	2024-2025
Student Name	Maxyne Nuela Ignacio	**Section**	BSCS-4B
Due date		**Points**	

Object Detection and Recognition

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

Task:

HOG is a feature descriptor widely used for object detection, particularly for human detection. In this exercise, you will:

- Load an image containing a person or an object.
- Convert the image to grayscale.
- Apply the HOG descriptor to extract features.
- Visualize the gradient orientations on the image.
- Implement a simple object detector using HOG features.

Key Points:

- HOG focuses on the structure of objects through gradients.
- Useful for detecting humans and general object recognition.

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

Step 1: Install Libraries

```
[ ]  pip install opencv-python scikit-image matplotlib

Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-packages (0.24.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.26.4)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (1.13.1)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (3.4.2)
Requirement already satisfied: pillow>=9.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (10.4.0)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (2.36.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (2024.9.20)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (24.1)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image) (0.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

- This code installs three essential libraries for image processing and visualization.
- It installs OpenCV for image processing and computer vision tasks.
- It installs scikit-image for advanced image processing techniques.
- It installs Matplotlib for creating visualizations and plotting images and data.

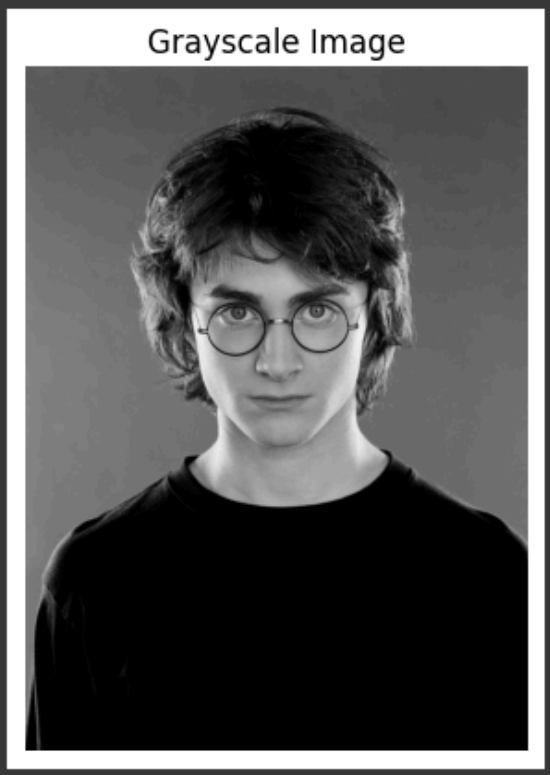
Step 2: Upload and Display the Image

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = 'image01.jpg'
image = cv2.imread(image_path)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')
plt.show()
```

- The code imports OpenCV and Matplotlib libraries for image processing and visualization.
- It loads an image from a specified file path using OpenCV's `imread` function.
- It displays the loaded image with Matplotlib after converting it from BGR to RGB color format for accurate colors.

Step 3: Convert the Image to Grayscale

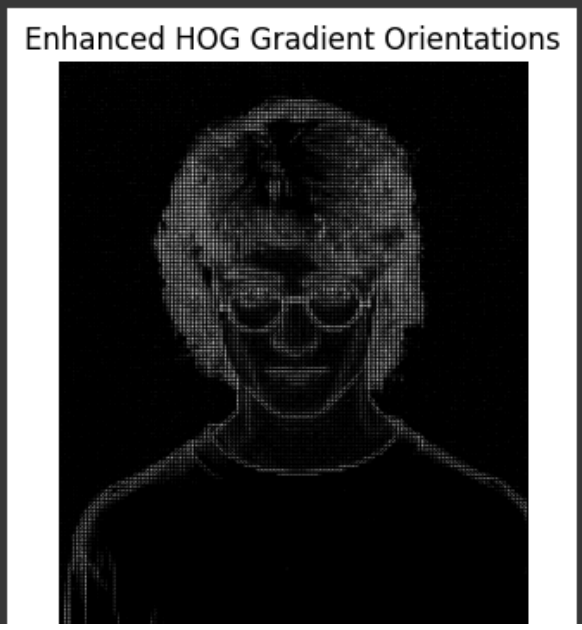
```
[ ]  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
     plt.imshow(gray_image, cmap='gray')
     plt.title("Grayscale Image")
     plt.axis('off')
     plt.show()
```



- This code converts the loaded color image to grayscale using OpenCV's `cvtColor` function.
- It displays the grayscale image with Matplotlib using a grayscale color map for proper shading.
- The image is shown without axis markings, and the title "Grayscale Image" is added for context.

Step 4: Apply the HOG Descriptor to Extract Features

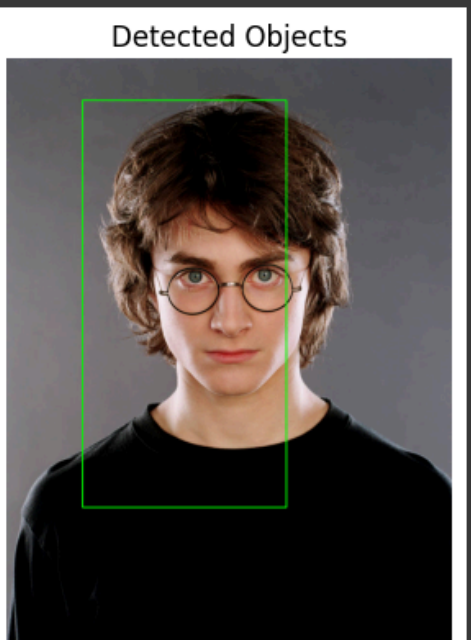
```
[ ]  from skimage import exposure
     import matplotlib.pyplot as plt

     # Compute HOG features and a visualization of gradients
     hog_features, hog_image = hog(
         gray_image,
         pixels_per_cell=(8, 8),
         cells_per_block=(2, 2),
         orientations=9,
         visualize=True,
         block_norm='L2-Hys'
     )

     # Increase contrast of the HOG image for better visibility
     hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

     # Display the enhanced HOG image
     plt.imshow(hog_image_rescaled, cmap='gray')
     plt.title("Enhanced HOG Gradient Orientations")
     plt.axis('off')
     plt.show()
```



- This code calculates HOG (Histogram of Oriented Gradients) features for the grayscale image and generates a visualization of the gradient orientations.
- It enhances the contrast of the HOG image using `rescale_intensity` to improve visibility of the gradient patterns.
- Finally, it displays the enhanced HOG image in grayscale with Matplotlib, showing gradient orientations without axis markings.

Step 5: Implement a Simple Object Detector Using HOG Features

```
[ ]  # Initialize HOG descriptor and SVM for human detection
     hog_detector = cv2.HOGDescriptor()
     hog_detector.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

     # Detect objects (people in this case)
     (rects, weights) = hog_detector.detectMultiScale(
         gray_image,
         winStride=(8, 8),
         padding=(16, 16),
         scale=1.05
     )

     # Draw bounding boxes around detected objects
     for (x, y, w, h) in rects:
         cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

     # Display the detection results
     plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
     plt.title("Detected Objects")
     plt.axis('off')
     plt.show()
```

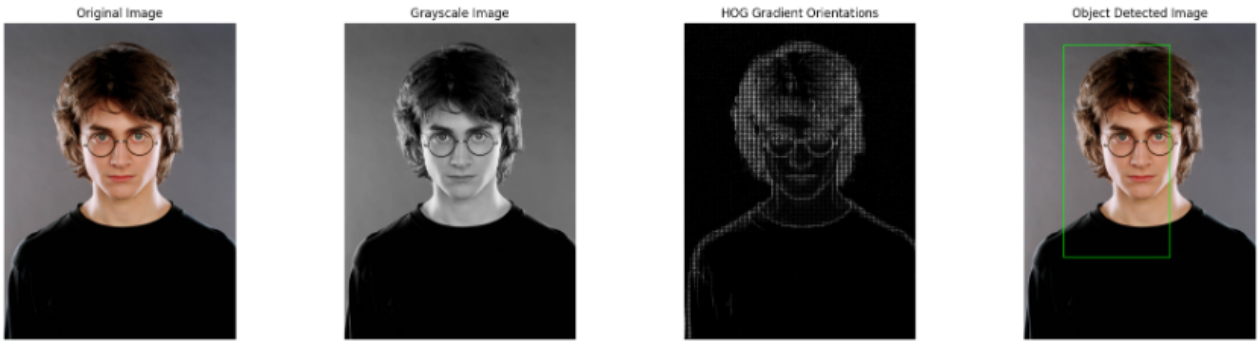


- The code initializes a HOG descriptor with a default SVM detector specifically trained to detect people in images.
- It uses the `detectMultiScale` method to detect people in the grayscale image, adjusting parameters for window stride, padding, and scaling.
- For each detected object, it draws a green bounding box on the original image and then displays the image with detection results using Matplotlib, without axis markings.

FINAL STEP FOR HOG: SIDE BY SIDE COMPARISON

```
[ ]  import cv2
     import matplotlib.pyplot as plt
     from skimage.feature import hog
     from skimage import exposure

     # Specify the path to your image
     image_path = 'image01.jpg'
     image = cv2.imread(image_path)

     # Check if the image was loaded successfully
     if image is None:
         print("Error: Unable to load image. Please check the image path.")
     else:
         # Convert the image to grayscale
         gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

         # Compute HOG features and visualize gradient orientations
         hog_features, hog_image = hog(
             gray_image,
             pixels_per_cell=(8, 8),
             cells_per_block=(2, 2),
             orientations=9,
             visualize=True,
             block_norm='L2-Hys'
         )

         # Enhance contrast of HOG image for better visibility
         hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

         # Initialize HOG descriptor and SVM for human detection
         hog_detector = cv2.HOGDescriptor()
         hog_detector.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

         # Detect objects (e.g., people) in the grayscale image
         (rects, weights) = hog_detector.detectMultiScale(
             gray_image,
             winStride=(8, 8),
             padding=(16, 16),
             scale=1.05
         )
```

```
# Draw bounding boxes around detected objects
detected_image = image.copy()
for (x, y, w, h) in rects:
    cv2.rectangle(detected_image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Convert BGR images to RGB for display in matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
detected_image_rgb = cv2.cvtColor(detected_image, cv2.COLOR_BGR2RGB)

# Display the images side by side
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

# Original Image
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')

# Grayscale Image
axes[1].imshow(gray_image, cmap='gray')
axes[1].set_title("Grayscale Image")
axes[1].axis('off')

# HOG Image
axes[2].imshow(hog_image_rescaled, cmap='gray')
axes[2].set_title("HOG Gradient Orientations")
axes[2].axis('off')

# Object Detected Image
axes[3].imshow(detected_image_rgb)
axes[3].set_title("Object Detected Image")
axes[3].axis('off')

plt.tight_layout()
plt.show()
```



- It loads an image from a specified path and checks if it was loaded successfully. If not, it prints an error message.
- The image is then converted to grayscale, and HOG (Histogram of Oriented Gradients) features are computed, displaying an enhanced visualization of gradient orientations.
- A HOG-based person detector, initialized with a pre-trained SVM, detects people in the grayscale image, and bounding boxes are drawn around each detected person.
- The code displays four images side-by-side using Matplotlib: the original image, grayscale image, HOG gradient visualization, and the image with detected objects marked by green bounding boxes.

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

Task:

YOLO is a deep learning-based object detection method. In this exercise, you will:

- Load a pre-trained YOLO model using TensorFlow.
- Feed an image to the YOLO model for object detection.
- Visualize the bounding boxes and class labels on the detected objects in the image.
- Test the model on multiple images to observe its performance.

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

Step 1: Install Necessary Packages

```
[ ]  pip install tensorflow opencv-python matplotlib
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflo
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensor
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorf
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from ten
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflo
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorf
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorf
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from ten
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensor
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotli
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotli
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplo
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunpar
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorfl
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorf
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensor
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from re
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-pack
```

- This command installs TensorFlow, a popular library for machine learning and deep learning, which provides tools for training and running neural networks.
- OpenCV-Python is installed, which includes the OpenCV library for computer vision, enabling image processing and object detection tasks.
- Matplotlib is added, a plotting library useful for visualizing data, images, and graphs in Python.

Coding and Result for YOLO Task

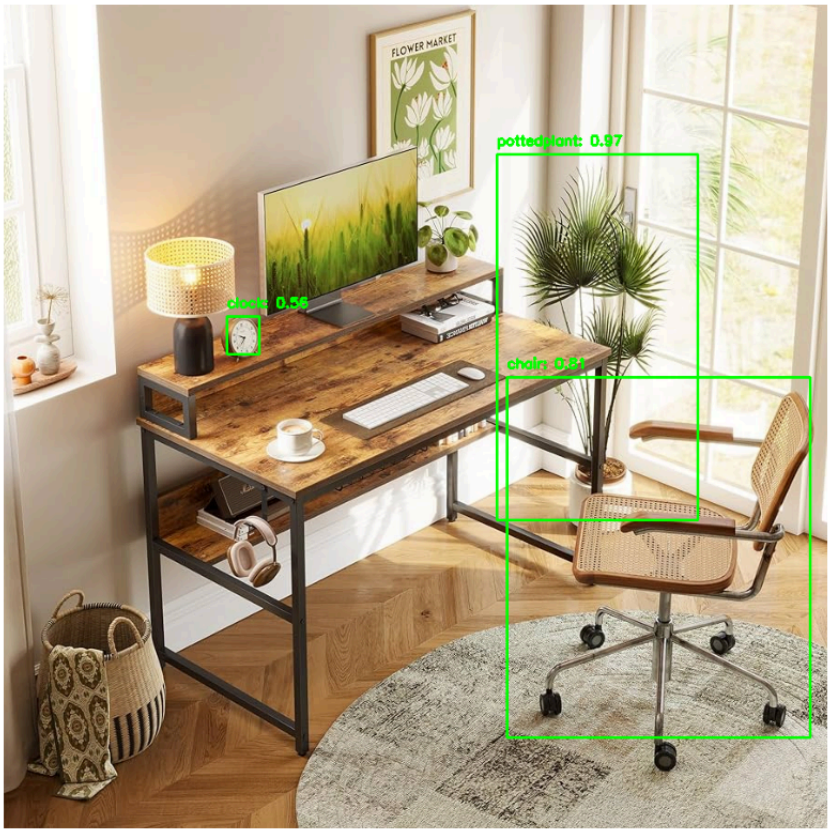
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Paths to your YOLO configuration, weight files, and COCO names
config_path = 'yolov3.cfg'       # Replace with the path to your YOLO config file
weights_path = 'yolov3.weights'  # Replace with the path to your YOLO weights file
names_path = 'coco.names'        # Replace with the path to your COCO names file

# Load class labels
with open(names_path, 'r') as f:
    class_labels = [line.strip() for line in f.readlines()]

# Load the YOLO network with OpenCV
net = cv2.dnn.readNetFromDarknet(config_path, weights_path)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

# Define a function for detecting objects using YOLO
def detect_objects(image):
    height, width = image.shape[:2]

    # Preprocess the image for YOLO
    blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
    net.setInput(blob)

    # Get the output layer names
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]

    # Run the forward pass
    outputs = net.forward(output_layers)

    # Initialize lists for detected bounding boxes, confidences, and class IDs
    boxes = []
    confidences = []
    class_ids = []
```

```
    # Process each output
    for output in outputs:
        for detection in output:
            scores = detection[5:]  # Skip the first 5 elements
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            # Filter out low confidence detections
            if confidence > 0.5:
                # Scale the bounding box coordinates to the original image size
                box = detection[0:4] * np.array([width, height, width, height])
                center_x, center_y, w, h = box.astype('int')

                # Calculate top-left corner coordinates
                x = int(center_x - (w / 2))
                y = int(center_y - (h / 2))

                # Append to the lists
                boxes.append([x, y, int(w), int(h)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    # Perform non-max suppression to remove redundant overlapping boxes
    indices = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)

    # Draw bounding boxes and labels
    for i in indices.flatten():
        x, y, w, h = boxes[i]
        label = f"{class_labels[class_ids[i]]}: {confidences[i]:.2f}"
        color = (0, 255, 0)  # Green color for bounding box
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        cv2.putText(image, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return image
```

```
# Test the model on multiple images
image_paths = ['image05.jpg', 'image04.jpg', 'image06.jpg']  # Update with your actual image file names
for path in image_paths:
    # Load the image and check if it's loaded correctly
    image = cv2.imread(path)
    if image is None:
        print(f"Error: Could not load image at path: {path}")
        continue

    # Perform object detection
    detected_image = detect_objects(image)

    # Convert to RGB for displaying with matplotlib
    detected_image_rgb = cv2.cvtColor(detected_image, cv2.COLOR_BGR2RGB)

    # Display the image with detected objects
    plt.figure(figsize=(10, 10))
    plt.imshow(detected_image_rgb)
    plt.axis('off')
    plt.show()
```


pottedplant: 0.91
book: 0.81
keyboard: 1.00
mouse: 0.94


chair: 0.67
chair: 0.81
pottedplant: 0.67
laptop: 1.00
book: 0.52
vase: 0.43



- The code imports essential libraries like OpenCV, NumPy, and Matplotlib, setting up the necessary tools for image processing and visualization.
- Paths to the YOLO configuration, weights, and class names are defined to load the pre-trained YOLO model for object detection.
- A function called `detect_objects` processes the input image through the YOLO network, retrieving bounding boxes, confidence scores, and class IDs for detected objects.
- Non-max suppression is applied to eliminate overlapping boxes, enhancing the clarity of the detection results by retaining only the best bounding boxes.
- The code tests the model on multiple images, displaying the results with detected objects highlighted and labeled using Matplotlib for clear visualization.

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

Task:

SSD is a real-time object detection method. For this exercise:

- Load an image of your choice.
- Utilize the TensorFlow Object Detection API to apply the SSD model.
- Detect objects within the image and draw bounding boxes around them.
- Compare the results with those obtained from the YOLO model.

Key Points:

- SSD is efficient in terms of speed and accuracy.
- Ideal for applications requiring both speed and moderate precision.

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

```
!pip install tensorflow
!pip install tensorflow-hub
```

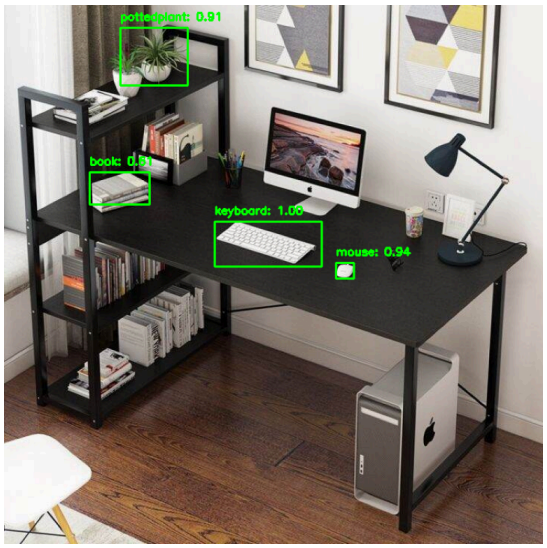
```
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tenso
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorfl
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflo
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from ast
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->ten
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->te
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->t
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fr
```

- The code uses `pip` to install TensorFlow, a powerful library for building and training machine learning models, particularly deep learning networks.
- It also installs TensorFlow Hub, which provides a repository of pre-trained models that can be easily integrated into TensorFlow applications.
- These installations enable developers to enhance advanced machine learning capabilities and access a variety of ready-to-use models for various tasks.

Coding and Result for SSD TASK

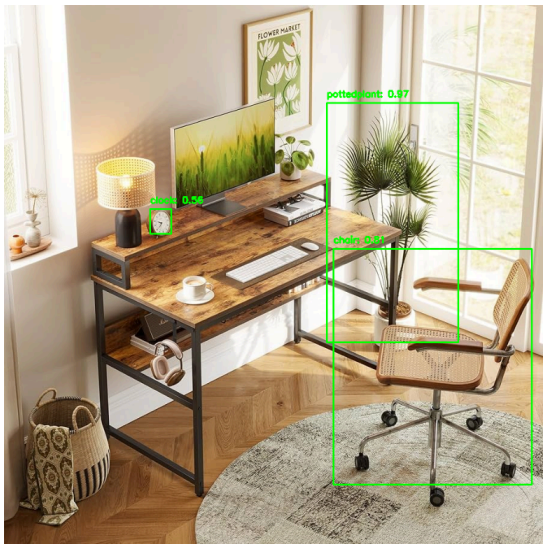
```
[ ]  # Import Libraries.
     import tensorflow as tf
     import tensorflow_hub as hub
     import cv2
     import numpy as np
     import matplotlib.pyplot as plt

     # Load the SSD model from TensorFlow Hub.
     ssd_model = hub.load("https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2")

     def detect_objects(image_path):
         # Load image.
         image = cv2.imread(image_path)
         if image is None:
             raise ValueError(f"Image not found at the specified path: {image_path}")

         # Convert to RGB.
         image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         height, width = image.shape[:2]

         # Simulate YOLO detection (placeholder).
         # Here, add your YOLO model detection code if available.
         # If not, for demonstration, you can return the original image.
         return image_rgb  # Replace this with actual YOLO detection logic if you have it.

     def detect_objects_ssd(image_path):
         # Load image.
         image = cv2.imread(image_path)
         if image is None:
             raise ValueError(f"Image not found at the specified path: {image_path}")

         # Convert to RGB.
         image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         height, width = image.shape[:2]

         # Preprocess image for SSD model.
         input_tensor = tf.image.resize(image_rgb, (300, 300))
         input_tensor = tf.cast(input_tensor, dtype=tf.uint8)  # Convert to uint8.
         input_tensor = tf.expand_dims(input_tensor, axis=0)  # Add batch dimension.
```

```
[ ]          if detection_scores[i] >= confidence_threshold:
                 box = detection_boxes[i]
                 y_min, x_min, y_max, x_max = int(box[0] * height), int(box[1] * width), int(box[2] * height),

                 # Draw bounding box and label.
                 cv2.rectangle(image_rgb, (x_min, y_min), (x_max, y_max), (0, 255, 0), 3)
                 label = f"Class {detection_classes[i]}: {detection_scores[i]:.2f}"
                 cv2.putText(image_rgb, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0),

         return image_rgb

     # Multiple images to test SSD and compare with YOLO.
     image_paths = [
         'image05.jpg',
         'image06.jpg',
         'image04.jpg'
     ]

     # Visualization.
     plt.figure(figsize=(14, 12))

     for i, path in enumerate(image_paths, 1):
         # YOLO detection.
         yolo_detected_image = detect_objects(path)

         # SSD detection.
         ssd_detected_image = detect_objects_ssd(path)

         # Display YOLO result.
         plt.subplot(len(image_paths), 2, 2 * i - 1)
         plt.imshow(yolo_detected_image)
         plt.axis('off')
         plt.title(f"Test Image {i} (YOLO Detection)")

         # Display SSD result.
         plt.subplot(len(image_paths), 2, 2 * i)
         plt.imshow(ssd_detected_image)
         plt.axis('off')
         plt.title(f"Test Image {i} (SSD Detection)")

     plt.tight_layout()
     plt.show()
```


pottedplant: 0.91
book: 0.91
keyboard: 1.00
mouse: 0.94


chair: 0.91
pottedplant: 0.83
book: 0.52


Class 67: 0.73: 0.60


pottedplant: 0.97
chair: 0.58


Class 64: 0.63
Class 64
0.50
0.72

- The code imports necessary libraries for handling object detection, image processing, and visualization.
- It loads the SSD MobileNet v2 model from TensorFlow Hub to perform SSD-based object detection.
- A placeholder function simulates YOLO detection, returning the original RGB image to allow for future YOLO integration.
- The `detect_objects_ssd` function preprocesses images, applies SSD detection, and draws bounding boxes and labels for objects above a confidence threshold.
- The code tests detection on multiple images, displaying SSD and placeholder YOLO results side-by-side with Matplotlib for easy comparison.

Exercise 4: Traditional vs. Deep Learning Object Detection Comparison

Task:

Compare traditional object detection (e.g., HOG-SVM) with deep learning-based methods (YOLO, SSD):

- Implement HOG-SVM and either YOLO or SSD for the same dataset.
- Compare their performances in terms of accuracy and speed.
- Document the advantages and disadvantages of each method.

Key Points:

- Traditional methods may perform better in resource-constrained environments.
- Deep learning methods are generally more accurate but require more computational power.

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

Coding and Result for HOG-SVM and SDD

```
# Import Libraries
import cv2
import matplotlib.pyplot as plt
import time
import tensorflow as tf
import tensorflow_hub as hub

# Load the SSD model from TensorFlow Hub
ssd_model = hub.load("https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2")

# Initialize HOG descriptor with default people detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
```

```
# Define a mapping from class indices to labels (for SSD)
class_labels = {0: "Person", 1: "Bicycle", 2: "Car", 3: "Motorcycle", 4: "Airplane",
                5: "Bus", 6: "Train", 7: "Truck", 8: "Boat", 9: "Traffic Light",
                10: "Fire Hydrant", 11: "Stop Sign", 12: "Parking Meter",
                13: "Bench", 14: "Bird", 15: "Cat", 16: "Dog", 17: "Horse",
                18: "Sheep", 19: "Cow", 20: "Elephant", 21: "Bear",
                22: "Zebra", 23: "Giraffe", 24: "Backpack", 25: "Umbrella",
                26: "Handbag", 27: "Tie", 28: "Suitcase", 29: "Frisbee",
                30: "Skis", 31: "Snowboard", 32: "Sports Ball", 33: "Kite",
                34: "Baseball Bat", 35: "Baseball Glove", 36: "Skateboard",
                37: "Surfboard", 38: "Tennis Racket", 39: "Bottle",
                40: "Wine Glass", 41: "Cup", 42: "Fork", 43: "Knife",
                44: "Spoon", 45: "Bowl", 46: "Banana", 47: "Apple",
                48: "Sandwich", 49: "Orange", 50: "Broccoli", 51: "Carrot",
                52: "Hot Dog", 53: "Pizza", 54: "Donut", 55: "Cake",
                56: "Chair", 57: "Couch", 58: "Potted Plant", 59: "Bed",
                60: "Dining Table", 61: "Toilet", 62: "TV", 63: "Laptop",
                64: "Mouse", 65: "Remote", 66: "Keyboard", 67: "Cell Phone",
                68: "Microwave", 69: "Oven", 70: "Toaster", 71: "Sink",
                72: "Refrigerator", 73: "Book", 74: "Clock", 75: "Vase",
                76: "Scissors", 77: "Teddy Bear", 78: "Hair Drier", 79: "Toothbrush"}

# HOG-SVM Detection Function
def detect_objects_hog(image_path):
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"Image not found at the specified path: {image_path}")

    # Convert image to grayscale
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Start timing
    start_time = time.time()

    # Detect people in the image using HOG-SVM
    (rects, weights) = hog.detectMultiScale(image_gray, winStride=(4, 4), padding=(8, 8), scale=1.05)

    # End timing
    end_time = time.time()
    hog_time = end_time - start_time
```

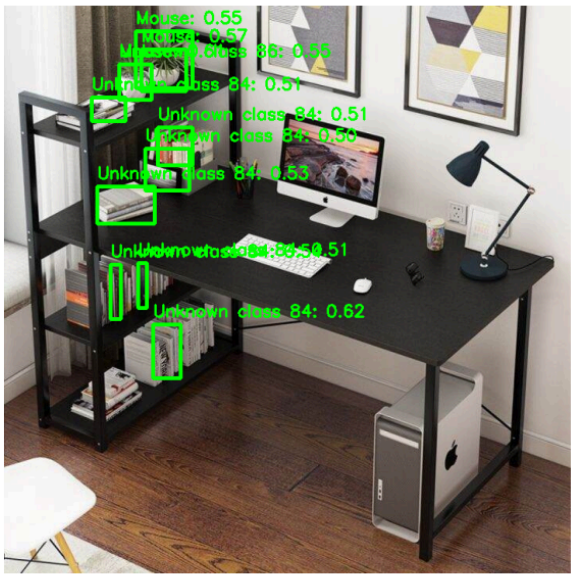
```
    # Draw bounding boxes and labels for HOG-SVM
    for (x, y, w, h) in rects:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(image, "Person", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    print(f"HOG-SVM detection completed in {hog_time:.4f} seconds for {image_path}.")
    return image, hog_time

# SSD Detection Function
def detect_objects_ssd(image_path):
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"Image not found at the specified path: {image_path}")

    # Convert to RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width = image.shape[:2]

    # Resize image to 300x300, expected by SSD model
    input_tensor = tf.image.resize_with_pad(image_rgb, 300, 300)
    input_tensor = tf.cast(input_tensor, dtype=tf.uint8)
    input_tensor = tf.expand_dims(input_tensor, axis=0)
```

```
# Start timing
start_time = time.time()

# Run SSD model
detections = ssd_model(input_tensor)

# End timing
end_time = time.time()
ssd_time = end_time - start_time

# Extract detection results
detection_boxes = detections['detection_boxes'][0].numpy()
detection_scores = detections['detection_scores'][0].numpy()
detection_classes = detections['detection_classes'][0].numpy().astype(int)

# Set a confidence threshold
confidence_threshold = 0.5

# Draw bounding boxes on the image
for i in range(len(detection_scores)):
    if detection_scores[i] >= confidence_threshold:
        box = detection_boxes[i]
        y_min, x_min, y_max, x_max = (int(box[0] * height), int(box[1] * width),
                                      int(box[2] * height), int(box[3] * width))
        cv2.rectangle(image_rgb, (x_min, y_min), (x_max, y_max), (0, 255, 0), 3)
        class_index = detection_classes[i]

        # Check if the class index is in our dictionary
        if class_index in class_labels:
            label = f"{class_labels[class_index]}: {detection_scores[i]:.2f}"
        else:
            label = f"Unknown class {class_index}: {detection_scores[i]:.2f}"
            print(f"Warning: Detected class index {class_index} is not in class_labels")

        cv2.putText(image_rgb, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255,

print(f"SSD detection completed in {ssd_time:.4f} seconds for {image_path}.")
return image_rgb, ssd_time
```

```
# Multiple images to test HOG-SVM and SSD
image_paths = [
    'image04.jpg',
    'image05.jpg'
]

# Visualization
plt.figure(figsize=(14, 12))

for i, path in enumerate(image_paths, 1):
    # HOG-SVM detection
    hog_detected_image, hog_time = detect_objects_hog(path)

    # SSD detection
    ssd_detected_image, ssd_time = detect_objects_ssd(path)

    # Display HOG-SVM result
    plt.subplot(len(image_paths), 2, 2 * i - 1)
    plt.imshow(cv2.cvtColor(hog_detected_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f"HOG-SVM Detection - Time: {hog_time:.4f} sec")

    # Display SSD result
    plt.subplot(len(image_paths), 2, 2 * i)
    plt.imshow(ssd_detected_image)
    plt.axis('off')
    plt.title(f"SSD Detection - Time: {ssd_time:.4f} sec")

plt.tight_layout()
plt.show()
```

HOG-SVM Detection - Time: 1.7325 sec

SSD Detection - Time: 3.7432 sec



HOG-SVM Detection - Time: 2.4496 sec

SSD Detection - Time: 0.0861 sec

- Imports essential libraries and loads SSD MobileNet v2 model from TensorFlow Hub
- Sets up HOG-SVM descriptor to detect people and prepares SSD class labels
- Runs HOG-SVM detection, drawing boxes around people and logging the time taken
- Uses SSD to detect objects, resizing images and labeling results based on confidence
- Displays side-by-side detection results for each method, showing processing times for easy comparison

Submission Instructions:

1. **Repository Setup**:
   - Create or navigate to the GitHub repository named CSST106-Perception and Computer Vision.
   - Inside the repository, create a folder specifically for this assignment (e.g., Exercise-2).
2. **Submission Format**:
   - **Processed Images**: Save and upload all images generated by your code (e.g., keypoint detection, feature matching, homography results) to the Exercise-2 folder in the repository.
   - **Code**: Save your Python scripts or Jupyter Notebook files containing your implementation and upload them to the same folder.
   - **Documentation**: Write a brief document (README.md or PDF) explaining your approach, observations, and results for each task. This document should provide enough detail for someone reviewing your work to understand your process.
   - **Folder Organization**:
     - Ensure that all content is properly labeled and structured for easy navigation.
     - You might have subfolders like:
       - images/ (for processed images)
       - code/ (for Python code or Jupyter notebooks)
       - documentation/ (for explanations, README.md, etc.)

3. **Filename Format**:
   - For all files (processed images, code, and documentation), use the following filename format: **[SECTION-BERNARDINO-EXER4]** 4D-BERNARDINO-EXER4

   Example filenames:
   - 4D-BERNARDINO-EXER4.py (for Python script)
   - 4D-BERNARDINO-EXER4.ipynb (for Jupyter Notebook)
   - 4D-BERNARDINO-EXER4-keypoints.jpg (for processed images)
   - 4D-BERNARDINO-EXER4-documentation.md (for documentation)

4. **Penalties**:
   - **Incorrect Filename**: If the filename format is not followed, there will be a 5-point deduction.
   - **Late Submission**: There will be a 5-point deduction per day for late submissions.
   - **Cheating/Plagiarism**: Any evidence of cheating or plagiarism will result in additional penalties according to the university's academic integrity policies.

By adhering to these submission guidelines and formatting, you will avoid penalties and ensure your work

is well-received.

Rubric for Exercise 4: Object Detection and Recognition

Criteria	Excellent (90-100%)	Good (75-89%)	Satisfactory (60-74%)	Needs Improvement (0-59%)
Exercise 1: HOG Object Detection	Accurate implementation of HOG with clear visualization of gradient orientations. Code is efficient, well-structured, and well-commented.	Minor issues in visualization or code optimization. Code mostly correct.	Basic HOG implementation but lacks clarity in visualization or explanation.	Incorrect HOG implementation, poor visualization, or missing explanations.
Exercise 2: YOLO Object Detection	Correct use of the YOLO model with precise bounding boxes and labels. Thorough explanation of results.	Minor flaws in bounding box accuracy or explanation. Mostly correct implementation.	Basic detection with unclear explanation or minor accuracy issues.	Incorrect YOLO implementation, poor results, or inadequate explanation.
Exercise 3: SSD with TensorFlow	Accurate object detection with SSD, clear comparison with YOLO. Code is well-documented and efficient.	Minor flaws in detection or comparison. Code works but could be optimized.	Basic SSD implementation with minimal comparison and explanation.	Incorrect detection or missing implementation and analysis.
Exercise 4: Traditional vs. Deep Learning Comparison	Comprehensive comparison with clear documentation, including advantages and disadvantages of each method. In-depth analysis of performance.	Comparison is mostly correct but lacks some details. Adequate explanation of strengths and weaknesses.	Basic comparison, lacks depth in analysis and documentation.	Poor or missing comparison, with incorrect or incomplete explanation.
Code Quality	Code is efficient, follows best practices, and is well-commented and organized.	Code works but could be optimized. Comments are present but minimal.	Code is functional but lacks structure or detailed comments.	Code is incorrect, poorly structured, or lacks comments.
Visualization of Results	Visuals are clear, well-labeled, and easy to interpret. Proper use of bounding boxes and labels.	Minor issues with visualization formatting or clarity.	Basic visuals present, but lack labels or are unclear.	Poor or missing visualization, with no clear output or interpretation.
Documentation	Comprehensive documentation explaining the approach, process, and observations for each task. Well-organized and detailed README or PDF.	Documentation is mostly correct but could be more detailed.	Basic documentation present but lacks depth or clarity.	Poor or missing documentation, fails to explain process or results.
Folder Organization and Naming	All files are correctly named and organized according to submission instructions. Proper use of folders and subfolders.	Mostly follows the naming and organization requirements with minor errors.	Basic organization present but does not fully adhere to the specified format.	Poor or missing organization; incorrect file names and structure.
Submission Format	All required files (processed images, code, documentation) are submitted and correctly formatted.	Most files are submitted with minor formatting issues.	Basic submission present but some files are missing or incorrectly formatted.	Incomplete submission with significant missing files or incorrect format.