4B BAUTISTA EXER4   
  
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.

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

IMPORT LIBRARIES :

import cv2

from skimage.feature import hog

import matplotlib.pyplot as plt

PROCESS LOAD THE IMAGE AND CONVERT TO HOG .  
# load image

image = cv2.imread('anime.jpg')

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Ensure the image is explicitly 2-dimensional

gray\_image = gray\_image.squeeze()

#This line ensures the image is explicitly 2D by removing any redundant dimensions.

if gray\_image.ndim == 3 and gray\_image.shape[2] == 1:

   gray\_image = gray\_image[:, :, 0]

# Apply HOG descriptor

features, hog\_image = hog(gray\_image, orientations=9, pixels\_per\_cell=(8, 8),

                  cells\_per\_block=(2, 2), visualize=True, channel\_axis=None)

# Display the HOG image

plt.figure(figsize=(8, 8))

plt.axis('off')

plt.imshow(hog\_image, cmap='gray')

plt.title('HOG Image')

plt.show()

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

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.

Load the pretrained YOLO model .

!pip install ultralytics

Load the libraries to start the task .

import cv2

import numpy as np

from ultralytics import YOLO  # Import the YOLO class from the ultralytics package

from google.colab.patches import cv2\_imshow  # Import cv2\_imshow for displaying images in Colab

LOAD THE YOLO MODEL AND THE IMAGE SAMPLE .

# Load the pre-trained YOLO model

model = YOLO('yolov8n.pt')  # Use a pre-trained YOLOv8 model

# Load an image

image = cv2.imread('anime.jpg')

# Check if the image was loaded successfully

if image is None:

    raise FileNotFoundError("Image not found. Please check the image path.")

# Perform inference

results = model(image)

# Process results

for result in results:

    boxes = result.boxes  # Get the boxes from the detection results

    for box in boxes:

        x1, y1, x2, y2 = box.xyxy[0].numpy()  # Get the bounding box coordinates

        conf = box.conf[0].item()  # Get the confidence score

        class\_id = int(box.cls[0].item())  # Get the class ID

        if conf > 0.5:  # Filter out low confidence detections

            label = f"Class: {class\_id}, Confidence: {conf:.2f}"

            # Draw bounding box and label

            cv2.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 2)

            cv2.putText(image, label, (int(x1), int(y1) - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

# Display the image with detections using cv2\_imshow

#cv2\_imshow(image)  # Use cv2\_imshow instead of cv2.imshow()

# Additionally, using matplotlib to show the image

plt.figure(figsize=(10, 10))

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  # Convert BGR to RGB for proper color display

plt.axis('off')

plt.title('YOLO Detection')

plt.show()

OUTPUT : 0: 448x640 1 truck, 253.8ms

Speed: 2.4ms preprocess, 253.8ms inference, 1.7ms postprocess per image at shape (1, 3, 448, 640)

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.

# Install required packages if not already done

!pip install tensorflow opencv-python matplotlib

import tensorflow as tf

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load pre-trained SSD model

model = tf.saved\_model.load('ssd\_mobilenet\_v2\_320x320\_coco17\_tpu-8/saved\_model')

# load image

image\_path = 'anime.jpg'

image\_np = cv2.imread(image\_path)

input\_tensor = tf.convert\_to\_tensor(image\_np)

input\_tensor = input\_tensor[tf.newaxis, ...]

# Perform inference

detections = model(input\_tensor)

# Visualize the bounding boxes

# Access 'num\_detections' using bracket notation instead of .pop

for i in range(int(detections['num\_detections'][0])):

    if detections['detection\_scores'][0, i] > 0.5:

        # Get Bounding box coordinates

        ymin, xmin, ymax, xmax = detections['detection\_boxes'][0, i].numpy()

        (left, right, top, bottom) = (xmin \* image\_np.shape[1], xmax \* image\_np.shape[1],

                                      ymin \* image\_np.shape[0], ymax \* image\_np.shape[0])

        # Draw bounding box

        cv2.rectangle(image\_np, (int(left), int(top)), (int(right), int(bottom)), (0, 255, 0), 2)

# Import cv2\_imshow from google.colab.patches

from google.colab.patches import cv2\_imshow

# Display the image using cv2\_imshow

cv2\_imshow(image\_np)

cv2.waitKey(0)

cv2.destroyAllWindows()

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.

def display\_image(image, title):

    plt.figure(figsize=(10, 10))

    plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  # Convert BGR to RGB for proper color display

    plt.axis('off')

    plt.title(title)

    plt.show()

# Load the pre-trained YOLO model

model = YOLO('yolov8n.pt')  # Use a pre-trained YOLOv8 model

# Load an image for detection

image\_path = 'anime.jpg'

image\_yolo = cv2.imread(image\_path)

# Check if the image was loaded successfully

if image\_yolo is None:

    raise FileNotFoundError("Image not found. Please check the image path.")

# --------- YOLO Detection ---------

start\_time\_yolo = time.time()  # Start time for YOLO speed measurement

results = model(image\_yolo)  # Perform inference

end\_time\_yolo = time.time()  # End time for YOLO speed measurement

# Process YOLO results

for result in results:

    boxes = result.boxes  # Get the boxes from the detection results

    for box in boxes:

        x1, y1, x2, y2 = box.xyxy[0].numpy()  # Get the bounding box coordinates

        conf = box.conf[0].item()  # Get the confidence score

        class\_id = int(box.cls[0].item())  # Get the class ID

        if conf > 0.5:  # Filter out low confidence detections

            label = f"Class: {class\_id}, Confidence: {conf:.2f}"

            # Draw bounding box and label

            cv2.rectangle(image\_yolo, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 2)

            cv2.putText(image\_yolo, label, (int(x1), int(y1) - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

# Display YOLO detection results

display\_image(image\_yolo, 'YOLO Detection')

print(f"YOLO Processing Time: {end\_time\_yolo - start\_time\_yolo:.4f} seconds")

# --------- HOG-SVM Detection ---------

hog = cv2.HOGDescriptor()

hog.setSVMDetector(cv2.HOGDescriptor\_getDefaultPeopleDetector())

# Load the same image for HOG-SVM detection

image\_hog = cv2.imread(image\_path)

# Check if the image was loaded successfully

if image\_hog is None:

    raise FileNotFoundError("Image not found. Please check the image path.")

# Perform HOG detection

start\_time\_hog = time.time()  # Start time for HOG speed measurement

boxes, weights = hog.detectMultiScale(image\_hog, winStride=(8, 8), padding=(8, 8), scale=1.05)

end\_time\_hog = time.time()  # End time for HOG speed measurement

# Draw bounding boxes on the HOG-SVM image

for (x, y, w, h) in boxes:

    cv2.rectangle(image\_hog, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display HOG-SVM detection results

display\_image(image\_hog, 'HOG-SVM Detection')

print(f"HOG-SVM Processing Time: {end\_time\_hog - start\_time\_hog:.4f} seconds")

OUTPUT :

YOLO Processing Time: 0.2540 seconds  
HOG-SVM Processing Time: 0.2435 seconds