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		**Section**	
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.

Sample Code:

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

# Load an image
image = cv2.imread('path_to_image.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 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=-1)

# Display the HOG image
plt.figure(figsize=(8, 8))
plt.axis('off')
plt.imshow(hog_image, cmap='gray')
plt.show()
```

Key Points:

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

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.

Sample Code:

```
import cv2
import numpy as np

# Load YOLO model and configuration
net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# Load image
image = cv2.imread('path_to_image.jpg')
height, width, channels = image.shape

# Prepare the image for YOLO
blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layers)
```

```python
# Process detections
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            # Draw bounding box
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the image
cv2.imshow('YOLO Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Key Points:

- YOLO is fast and suitable for real-time object detection.
- It performs detection in a single pass, making it efficient for complex scenes.

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.

Sample Code:

```
import tensorflow as tf
import cv2

# Load pre-trained SSD model
model = tf.saved_model.load('ssd_mobilenet_v2_coco/saved_model')

# Load image
image_path = 'path_to_image.jpg'
image_np = cv2.imread(image_path)
input_tensor = tf.convert_to_tensor(image_np)
input_tensor = input_tensor[tf.newaxis, ...]

# Run the model
detections = model(input_tensor)

# Visualize the bounding boxes
for i in range(int(detections.pop('num_detections'))):
    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)

# Display the image
cv2.imshow('SSD Detection', image_np)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Key Points:

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

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.

Sample Code Snippet for HOG-SVM:

```
# Assuming HOG features extracted as in Exercise 1
from sklearn import svm

# Train SVM with HOG features (sample dataset required)
clf = svm.SVC()
clf.fit(features, labels)  # features and labels should be prepared accordingly

# Predict and evaluate
prediction = clf.predict(new_image_features)
```

Key Points:

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

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.