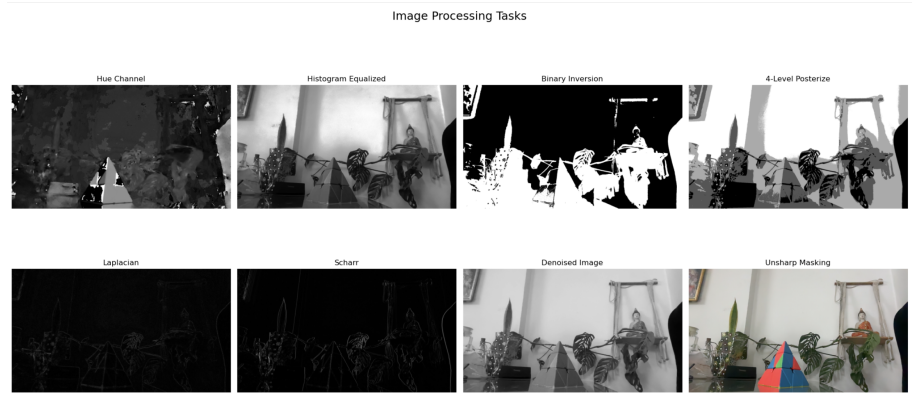
Assignment-2

Anurag More

June 2025

Task-1

In this assignment, I learned how to capture images using OpenCV and apply various image processing techniques like color space conversion (HSV, LAB), histogram equalization, thresholding, and posterization. I also explored edge detection using Laplacian and Scharr filters, noise removal with median filtering, and image sharpening with unsharp masking. Displaying all results in a 2x4 grid helped visualize and compare the transformations effectively as shown in Figure .



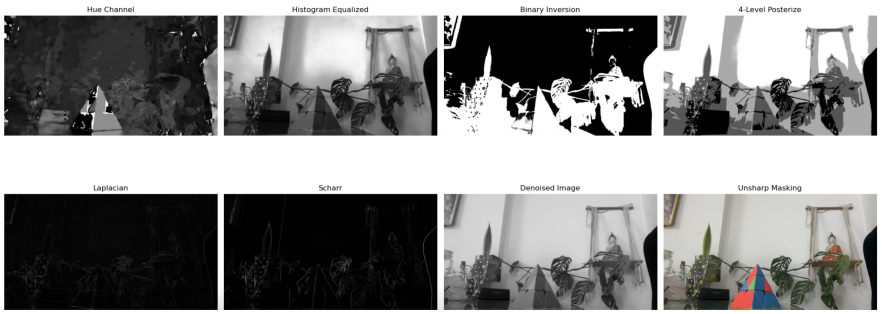
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def capture_image():
    cap = cv2.VideoCapture(0)  # Try 0, 1, 2...
    if not cap.isOpened():
        raise Exception("Webcam not accessible.")
    ret, frame = cap.read()
    cap.release()
    if not ret:
        raise Exception("Failed to capture image.")
    return cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # Convert to RGB



def hsv_channels(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    h, s, v = cv2.split(hsv)
    return h, s, v

def histogram_equalization(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    equalized = cv2.equalizeHist(gray)
    return gray, equalized

def binary_inversion(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    _, binary = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY_INV)
    return binary

def posterize(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    poster = np.floor(gray / 64) * 85
    return poster.astype(np.uint8)
```

```python
def edge_detection(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
    scharr = cv2.Scharr(gray, cv2.CV_64F, 1, 0)
    return np.abs(laplacian), np.abs(scharr)

def median_filter_denoising(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    noisy = gray.copy()
    # Add salt-and-pepper noise
    prob = 0.02
    mask = np.random.rand(*gray.shape)
    noisy[mask < prob] = 0
    noisy[mask > 1 - prob] = 255
    denoised = cv2.medianBlur(noisy, 3)
    return noisy, denoised

def unsharp_mask(img):
    blurred = cv2.GaussianBlur(img, (9, 9), 10.0)
    sharp = cv2.addWeighted(img, 1.5, blurred, -0.5, 0)
    return sharp

def lab_channels(img):
    lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    return l, a, b

img = capture_image()

fig, axes = plt.subplots(2, 4, figsize=(20, 10))
fig.suptitle("Image Processing Tasks", fontsize=18)


h, s, v = hsv_channels(img)
axes[0, 0].imshow(h, cmap='gray')
axes[0, 0].set_title("Hue Channel")
axes[0, 0].axis('off')
```

```
gray, eq = histogram_equalization(img)
axes[0, 1].imshow(eq, cmap='gray')
axes[0, 1].set_title("Histogram Equalized")
axes[0, 1].axis('off')

binary = binary_inversion(img)
axes[0, 2].imshow(binary, cmap='gray')
axes[0, 2].set_title("Binary Inversion")
axes[0, 2].axis('off')

poster = posterize(img)
axes[0, 3].imshow(poster, cmap='gray')
axes[0, 3].set_title("4-Level Posterize")
axes[0, 3].axis('off')

lap, scharr = edge_detection(img)
axes[1, 0].imshow(lap, cmap='gray')
axes[1, 0].set_title("Laplacian")
axes[1, 0].axis('off')

axes[1, 1].imshow(scharr, cmap='gray')
axes[1, 1].set_title("Scharr")
axes[1, 1].axis('off')

noisy, denoised = median_filter_denoising(img)
axes[1, 2].imshow(denoised, cmap='gray')
axes[1, 2].set_title("Denoised Image")
axes[1, 2].axis('off')

sharp = unsharp_mask(img)
axes[1, 3].imshow(sharp)
axes[1, 3].set_title("Unsharp Masking")
axes[1, 3].axis('off')

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

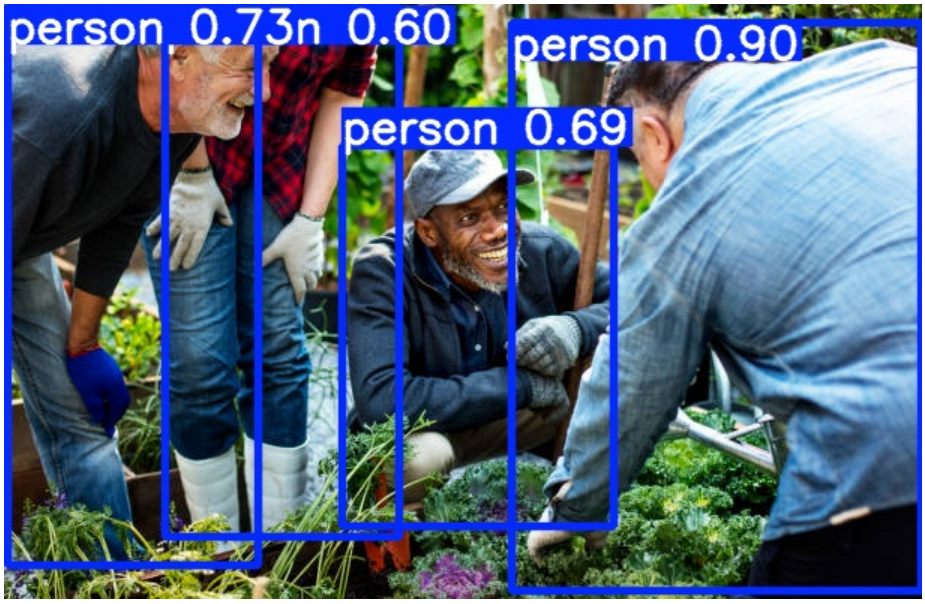
```
axes[1, 2].set_title("Denoised Image")
axes[1, 2].axis('off')

sharp = unsharp_mask(img)
axes[1, 3].imshow(sharp)
axes[1, 3].set_title("Unsharp Masking")
axes[1, 3].axis('off')

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

Image Processing Tasks



Task-2

In this task, I learned how to set up and use the YOLOv5 model for real-time object detection. I explored how YOLOv5 identifies and labels multiple objects in images or video streams efficiently. Implementing it on a custom video helped me understand the importance of frame-by-frame inference and how the model outputs confidence scores and class labels. This task also gave me practical experience in integrating deep learning models with OpenCV for real-time computer vision applications.




```
(yolov5-env) (base) anurag@Anurags-Laptop yolov5 % python detect.py --source detect.jpg --weights yolov5s.pt --conf 0
.4 --project runs/detect --name myoutput --exist-ok
/Users/anurag/yolov5/utils/general.py:32: UserWarning: pkg_resources is deprecated as an API. See https://setuptools.
pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slated for removal as early as 2025-11-30. Refrain
 from using this package or pin to Setuptools<81.
  import pkg_resources as pkg
detect: weights=['yolov5s.pt'], source=detect.jpg, data=data/coco128.yaml, imgsz=[640, 640], conf_thres=0.4, iou_thre
s=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_format=0, save_csv=False, save_conf=False, save_c
rop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs
/detect, name=myoutput, exist_ok=True, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False, v
id_stride=1
YOLOv5 🚀 v7.0-420-g0c99ce80 Python-3.12.7 torch-2.7.1 CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
image 1/1 /Users/anurag/yolov5/detect.jpg: 416x640 4 persons, 44.4ms
Speed: 1.1ms pre-process, 44.4ms inference, 3.3ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/myoutput
```

Task-3

In this task, I learned how to analyze an image by segmenting it into regions and using color detection techniques to distinguish between two very similar flags — Indonesia and Poland. The challenge was to handle variations in lighting, orientation, and scale, which required converting the image to HSV color space and examining dominant colors in different parts of the flag. This task improved my skills in color-based image segmentation and robust classification using simple yet effective image processing methods.

Code Link:https://github.com/xanuragx-0/Flag-Detection

```python
import cv2
import numpy as np
from ultralytics import YOLO


1 usage
def crop_flag(image_path):
    model = YOLO('yolov5su.pt')

    # Load image
    img = cv2.imread(image_path)

    # Run detection
    results = model(img)

    # Get detections
    for result in results:
        boxes = result.boxes
        if boxes is not None and len(boxes) > 0:
            # Get first detection
            box = boxes[0]
            x1, y1, x2, y2 = box.xyxy[0].cpu().numpy().astype(int)
            cropped_flag = img[y1:y2, x1:x2]
            return cropped_flag
    print("No object detected, using full image")
    return img

```

```python
def identify_flag(flag_image):
    # Convert to HSV
    hsv = cv2.cvtColor(flag_image, cv2.COLOR_BGR2HSV)

    # Define red and white color ranges in HSV
    red_lower = np.array([0, 50, 50])
    red_upper = np.array([10, 255, 255])

    red_lower2 = np.array([170, 50, 50])
    red_upper2 = np.array([180, 255, 255])

    # Create masks
    mask_red1 = cv2.inRange(hsv, red_lower, red_upper)
    mask_red2 = cv2.inRange(hsv, red_lower2, red_upper2)
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)

    # Split image into top and bottom halves
    height = flag_image.shape[0]
    top_half_red = mask_red[:height // 2]
    bottom_half_red = mask_red[height // 2:]

    # Count red pixels in each half
    red_top = cv2.countNonZero(top_half_red)
    red_bottom = cv2.countNonZero(bottom_half_red)

    # Classify based on red position
    if red_top > red_bottom:
        return "Indonesian Flag"
    else:
        return "Polish Flag"
```

```python
def main():
    image_path = "poland.png"  # Replace with your image path

    # Task 1: Crop flag using YOLOv5
    cropped_flag = crop_flag(image_path)

    # Task 2: Classify Indonesian vs Polish
    result = identify_flag(cropped_flag)

    print(f"Result: {result}")

    cv2.imwrite( filename: "cropped_flag.jpg", cropped_flag)


if __name__ == "__main__":
    main()
```

```
/Users/anurag/yolov5-env/bin/python /Users/anurag/PycharmProjects/pythonProject4/model_d.py

0: 416x640 (no detections), 73.5ms
Speed: 2.9ms preprocess, 73.5ms inference, 2.2ms postprocess per image at shape (1, 3, 416, 640)
No object detected, using full image
Result: Polish Flag

Process finished with exit code 0
```

Task-4

This paper presents a detailed technical and practical overview of YOLOv5, one of the most influential and widely used real-time object detection models. The authors, Rahima Khanam and Muhammad Hussain, begin by situating YOLOv5 within the broader evolution of the YOLO (You Only Look Once) family, which has revolutionized object detection by enabling real-time processing through a single-stage architecture. They highlight how YOLOv5, released in 2020 by Ultralytics, marks a significant leap forward from its predecessors, YOLOv3 and YOLOv4, by focusing on improved usability, flexibility, and deployment efficiency, especially on edge devices. One of the most notable changes is the migration of the YOLO architecture from the Darknet framework to PyTorch, which has made the model more accessible to a wider range of researchers and practitioners due to PyTorch's popularity and robust ecosystem.

The paper delves into the core architectural innovations that distinguish YOLOv5. The backbone of the model is built using Cross Stage Partial (CSP) modules, which split feature maps into two paths—one processed through dense blocks and the other passed directly—resulting in more efficient feature reuse and reduced computational redundancy, inspired by DenseNet architectures. For feature aggregation, YOLOv5 employs a Path Aggregation Network (PANet) neck, which enhances the integration of multi-scale features and improves detection performance, especially for small objects. The authors explain that YOLOv5's training process is enhanced by robust data augmentation techniques, most notably mosaic augmentation, which combines four images into a single training sample to increase data diversity and improve the model's ability to detect objects of varying sizes and orientations. The loss function used in YOLOv5 is a weighted combination of classification loss, objectness loss, and localization loss, optimized to maximize mean average precision (mAP) while balancing detection accuracy and localization quality.

YOLOv5's efficiency is further boosted by its use of 16-bit floating-point precision (FP16) for both training and inference, significantly reducing memory usage and speeding up computation on compatible hardware. The model also employs K-means clustering and genetic algorithms to generate anchor box dimensions tailored to specific datasets, which improves performance on custom object detection tasks where object scales and aspect ratios may differ from standard benchmarks. The authors note that YOLOv5 is available in several variants—n, s, m, l, and x—each offering a different balance between speed, accuracy, and computational requirements, making it suitable for deployment across a wide spectrum of hardware platforms, from resource-constrained IoT devices to high-performance servers.

The paper also discusses the practical aspects of working with YOLOv5, such as its use of a simple PyTorch TXT annotation format that is compatible with popular labeling tools like Roboflow, LabelImg, and CVAT. The inclusion of YAML configuration files for class labels and model settings simplifies the process of training on custom datasets. The authors highlight the extensive ecosystem of third-party integrations for automated compilation, quanti-

zation, experiment tracking, and dataset management, which further enhances YOLOv5's usability and adoption in both academic research and industrial applications.

In their analysis, the authors emphasize YOLOv5's architectural efficiency, versatility, and training innovations as key factors behind its widespread adoption. The model's high mAP scores, low inference times, and ease of use make it a leading choice for real-time object detection, particularly in scenarios with deployment constraints. The paper concludes by positioning YOLOv5 as a versatile, efficient, and accessible model that is well-suited for a wide range of computer vision tasks. While the paper provides a thorough technical overview and practical guidance, it could be further strengthened by empirical comparisons with newer YOLO versions and more case studies of real-world edge deployments. Overall, the paper is a valuable resource for understanding the design and practical strengths of YOLOv5 in the current landscape of object detection.