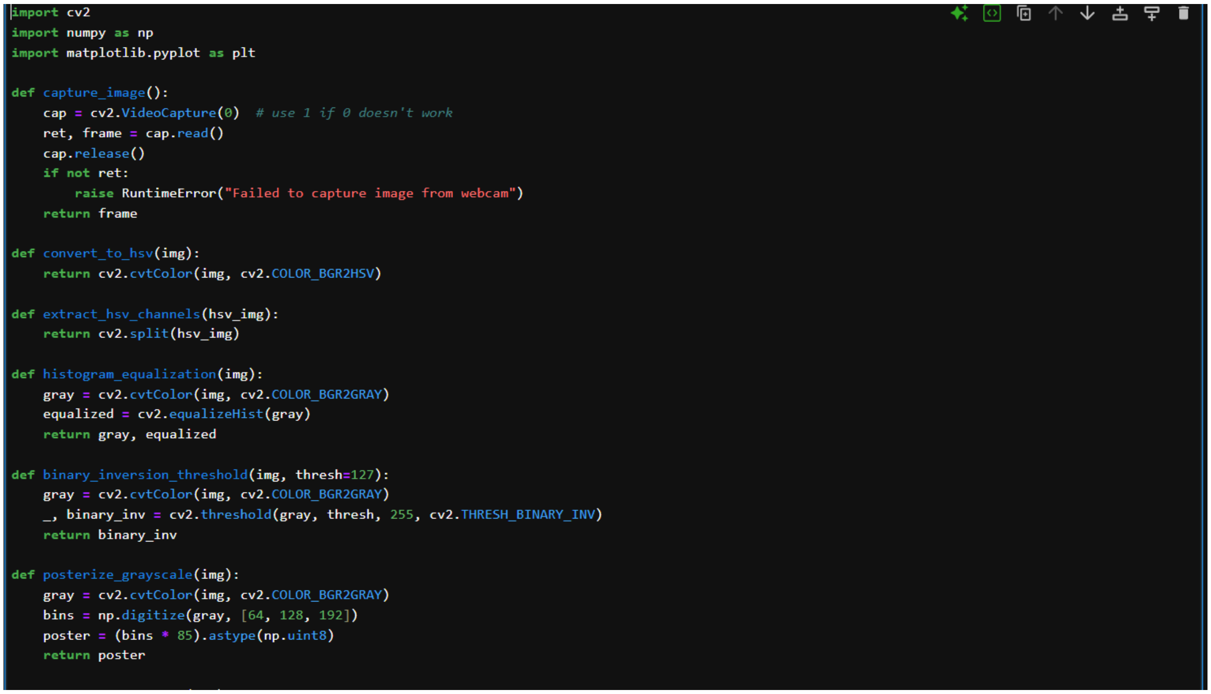
LEARNINGSThis assignment introduced me to the practical foundations of computer vision
deep learning-based object detection using OpenCV and YOLOv5

1. Image Processing with OpenCV (Task 1)
2. Object Detection with YOLOv5 (Task 2)
3. Custom Classification Logic (Task 3)- program to differentiate between the flags of Indonesia and Poland,
4. Deep Learning Concepts

TASK1



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

def capture_image():
    cap = cv2.VideoCapture(0)  # use 1 if 0 doesn't work
    ret, frame = cap.read()
    cap.release()
    if not ret:
        raise RuntimeError("Failed to capture image from webcam")
    return frame

def convert_to_hsv(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

def extract_hsv_channels(hsv_img):
    return cv2.split(hsv_img)

def histogram_equalization(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    equalized = cv2.equalizeHist(gray)
    return gray, equalized

def binary_inversion_threshold(img, thresh=127):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, binary_inv = cv2.threshold(gray, thresh, 255, cv2.THRESH_BINARY_INV)
    return binary_inv

def posterize_grayscale(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    bins = np.digitize(gray, [64, 128, 192])
    poster = (bins * 85).astype(np.uint8)
    return poster
```

Figure 1: Image 8

```python
def apply_edge_filters(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
    scharr = cv2.Scharr(gray, cv2.CV_64F, 1, 0)
    return np.uint8(np.absolute(laplacian)), np.uint8(np.absolute(scharr))

def add_salt_pepper_noise(img):
    noisy = img.copy()
    prob = 0.02
    noise = np.random.rand(*img.shape[:2])
    noisy[noise < prob] = 0
    noisy[noise > 1 - prob] = 255
    return noisy

def median_filter(img):
    return cv2.medianBlur(img, 5)

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

def convert_to_lab(img):
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    return cv2.split(lab)

# Main program
if __name__ == "__main__":
    img = capture_image()

    # 1. Convert to HSV
    hsv_img = convert_to_hsv(img)

    # Extract HSV channels
```

Figure 2: Image 1

```python
# Extract HSV channels
h, s, v = extract_hsv_channels(hsv_img)

# 2. Histogram equalization
gray_img, equalized_img = histogram_equalization(img)

# 3. Binary inversion thresholding
binary_inv_img = binary_inversion_threshold(img)

# 4. Posterize grayscale image
posterized_img = posterize_grayscale(img)

# 5. Edge detection filters
laplacian_img, scharr_img = apply_edge_filters(img)

# 6. Salt-and-pepper noise + median filter
noisy_img = add_salt_pepper_noise(img)
denoised_img = median_filter(noisy_img)

# 7. Unsharp masking for sharpening
sharpened_img = unsharp_mask(img)

# 8. Convert to LAB and split channels
l_channel, a_channel, b_channel = convert_to_lab(img)

# Plot results in 2x4 grid
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
axes = axes.flatten()

# 1 - HSV image (convert HSV to RGB for display)
axes[0].imshow(cv2.cvtColor(hsv_img, cv2.COLOR_HSV2RGB))
axes[0].set_title('HSV Image')
axes[0].axis('off')
```

Figure 3: Image 4

```
# 2,3,4 - Hue, Saturation, Value channels
axes[1].imshow(h, cmap='hsv')
axes[1].set_title('Hue Channel')
axes[1].axis('off')

axes[2].imshow(s, cmap='gray')
axes[2].set_title('Saturation Channel')
axes[2].axis('off')

axes[3].imshow(v, cmap='gray')
axes[3].set_title('Value Channel')
axes[3].axis('off')

# 5 - Histogram equalization: original grayscale
axes[4].imshow(gray_img, cmap='gray')
axes[4].set_title('Original Grayscale')
axes[4].axis('off')

# 6 - Histogram equalization: equalized image
axes[5].imshow(equalized_img, cmap='gray')
axes[5].set_title('Histogram Equalized')
axes[5].axis('off')

# 7 - Binary inversion threshold
axes[6].imshow(binary_inv_img, cmap='gray')
axes[6].set_title('Binary Inversion Threshold')
axes[6].axis('off')

# 8 - Posterized grayscale
axes[7].imshow(posterized_img, cmap='gray')
axes[7].set_title('Posterized Grayscale')
axes[7].axis('off')

plt.tight_layout()
```

Figure 4: Image 3

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

# Continuing from previous code (after plt.show())

fig2, axes2 = plt.subplots(2, 4, figsize=(20, 10))
axes2 = axes2.flatten()

# 1 - Laplacian filter output
axes2[0].imshow(laplacian_img, cmap='gray')
axes2[0].set_title('Laplacian Filter')
axes2[0].axis('off')

# 2 - Scharr filter output
axes2[1].imshow(scharr_img, cmap='gray')
axes2[1].set_title('Scharr Filter')
axes2[1].axis('off')

# 3 - Salt and pepper noisy image (show grayscale version)
axes2[2].imshow(cv2.cvtColor(noisy_img, cv2.COLOR_BGR2GRAY), cmap='gray')
axes2[2].set_title('Salt & Pepper Noise Added')
axes2[2].axis('off')

# 4 - Median filtered (denoised) image (grayscale)
axes2[3].imshow(cv2.cvtColor(denoised_img, cv2.COLOR_BGR2GRAY), cmap='gray')
axes2[3].set_title('Median Filtered (Denoised)')
axes2[3].axis('off')

# 5 - Unsharp masked (sharpened) image (show in RGB)
axes2[4].imshow(cv2.cvtColor(sharpened_img, cv2.COLOR_BGR2RGB))
axes2[4].set_title('Unsharp Masking (Sharpened)')
axes2[4].axis('off')

# 6 - LAB L channel
```

Figure 5: Image 2

```python
# 6 - LAB L channel
axes2[5].imshow(l_channel, cmap='gray')
axes2[5].set_title('LAB L Channel (Lightness)')
axes2[5].axis('off')

# 7 - LAB A channel
axes2[6].imshow(a_channel, cmap='gray')
axes2[6].set_title('LAB A Channel (Green-Red)')
axes2[6].axis('off')

# 8 - LAB B channel
axes2[7].imshow(b_channel, cmap='gray')
axes2[7].set_title('LAB B Channel (Blue-Yellow)')
axes2[7].axis('off')

plt.tight_layout()
plt.show()


    # For the rest of the tasks (Laplacian & Scharr, noise removal, sharpening, LAB channels),
    # you can create additional plots or print/save images as per your need.
```

Figure 6: Image 9

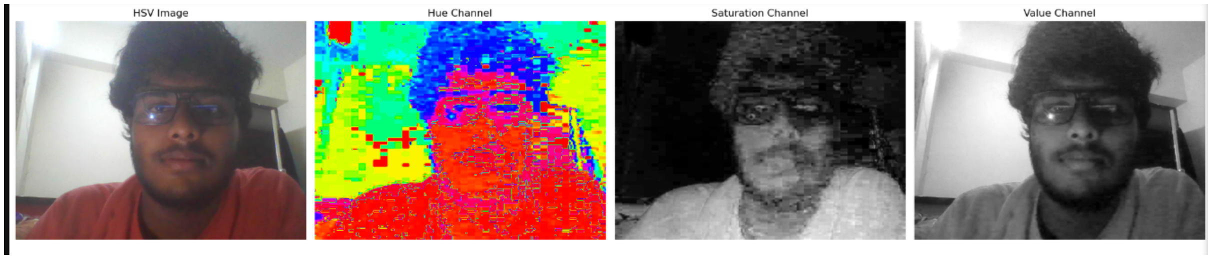


Figure 7: Image 12

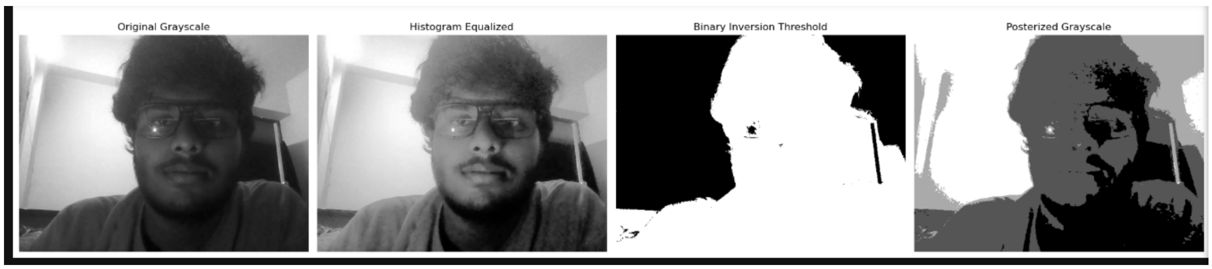


Figure 8: Image 11

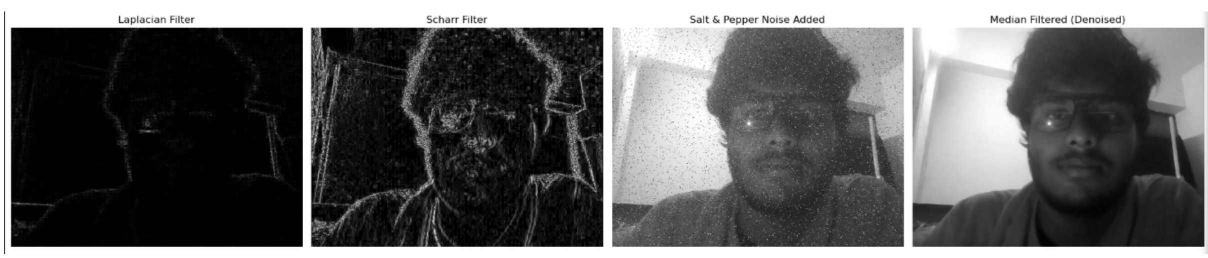


Figure 9: Image 13

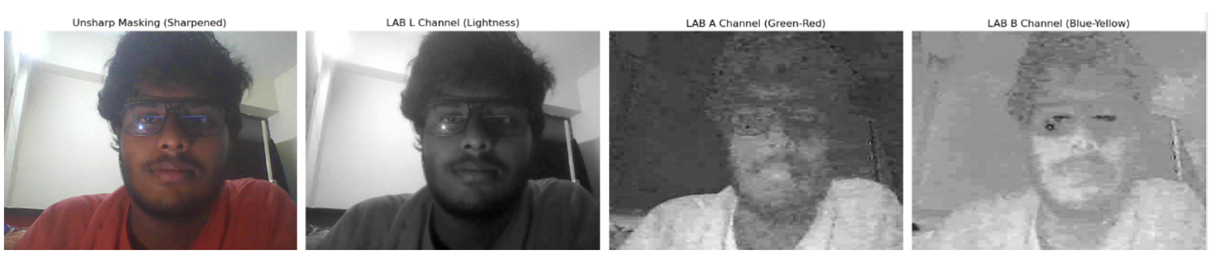


Figure 10: Image 7

TASK2

Figure 11: Image 10

```
480x640 1 person, 1 bottle, 1 toothbrush, 167.9ms
480x640 1 person, 1 bottle, 1 toothbrush, 183.1ms
480x640 1 bottle, 1 toothbrush, 182.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 164.2ms
480x640 1 person, 1 toothbrush, 181.8ms
480x640 1 person, 1 bottle, 1 toothbrush, 154.8ms
480x640 1 person, 1 bottle, 1 toothbrush, 166.9ms
480x640 1 person, 1 bottle, 1 toothbrush, 165.2ms
480x640 1 person, 1 bottle, 1 toothbrush, 158.5ms
480x640 1 person, 1 bottle, 1 toothbrush, 193.6ms
480x640 1 person, 1 bottle, 1 toothbrush, 176.8ms
480x640 1 person, 1 bottle, 1 toothbrush, 162.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 151.6ms
480x640 1 person, 1 bottle, 1 toothbrush, 190.4ms
480x640 1 person, 1 bottle, 1 toothbrush, 193.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 158.2ms
480x640 1 person, 1 bottle, 1 toothbrush, 171.3ms
480x640 1 person, 1 bottle, 1 toothbrush, 181.6ms
480x640 1 person, 1 bottle, 1 toothbrush, 198.4ms
480x640 1 person, 1 bottle, 1 toothbrush, 166.8ms
480x640 1 person, 1 bottle, 1 toothbrush, 181.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 163.2ms
480x640 1 person, 1 bottle, 1 toothbrush, 165.8ms
480x640 1 person, 1 bottle, 1 toothbrush, 151.7ms
480x640 1 person, 1 bottle, 1 toothbrush, 161.9ms
480x640 1 person, 1 bottle, 1 toothbrush, 162.2ms
480x640 1 person, 1 bottle, 1 toothbrush, 173.1ms
480x640 1 person, 1 bottle, 1 toothbrush, 181.9ms
480x640 1 person, 1 bottle, 1 toothbrush, 210.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 153.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 154.0ms
480x640 1 person, 1 bottle, 1 toothbrush, 165.7ms
480x640 1 person, 1 bottle, 1 toothbrush, 176.3ms
480x640 1 person, 1 bottle, 1 toothbrush, 159.6ms
480x640 1 person, 1 bottle, 1 toothbrush, 159.2ms
480x640 1 person, 1 bottle, 1 toothbrush, 164.4ms
480x640 1 person, 1 bottle, 1 toothbrush, 193.6ms
```

Figure 12: Image 6

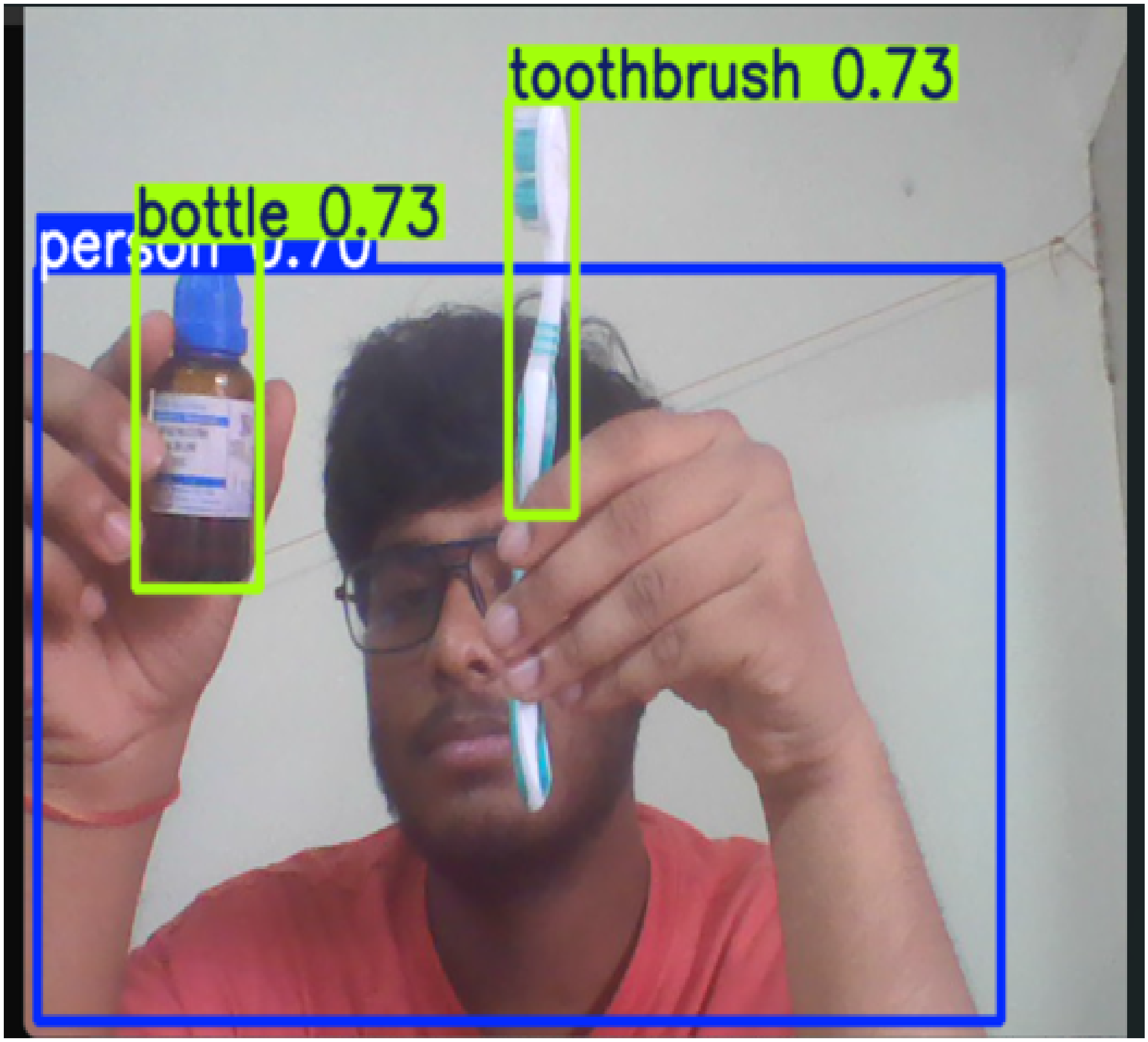


Figure 13: Image 5

article graphicx float array booktabs

TASK3

```
[10]: import cv2
import numpy as np
import os

def rotate_image(image, angle):
    (h, w) = image.shape[:2]
    center = (w//2, h//2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h), borderMode=cv2.BORDER_REPLICATE)
    return rotated

def create_test_set(base_image_path, output_dir='test_set', num_images=100):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    base_img = cv2.imread(base_image_path)
    if base_img is None:
        print("Error: Could not read the base image.")
        return

    base_img = cv2.resize(base_img, (300, 200))

    for i in range(num_images):
        angle = np.random.uniform(-180, 180)
        rotated = rotate_image(base_img, angle)

        # Optionally, add some noise or brightness change here for more variation

        output_path = os.path.join(output_dir, f"flag_rotated_{i+1}.jpg")
        cv2.imwrite(output_path, rotated)

    print(f"Created {num_images} rotated images in '{output_dir}' folder.")
```

Figure 14: Image 6

```python
    for i in range(num_images):
        angle = np.random.uniform(-180, 180)
        rotated = rotate_image(base_img, angle)

        # Optionally, add some noise or brightness change here for more variation

        output_path = os.path.join(output_dir, f"flag_rotated_{i+1}.jpg")
        cv2.imwrite(output_path, rotated)

    print(f"Created {num_images} rotated images in '{output_dir}' folder.")

# Example usage: generate 100 rotated images of Indonesia flag
create_test_set('indonesia_flag.png', output_dir='indonesia_test_set', num_images=100)

# Similarly, generate for Poland flag
create_test_set('poland_flag.png', output_dir='poland_test_set', num_images=100)
```

```
Created 100 rotated images in 'indonesia_test_set' folder.
Created 100 rotated images in 'poland_test_set' folder.
```

[3]:

```python
import cv2
import numpy as np

def is_red(hsv_pixel):
    h, s, v = hsv_pixel
    # Red can be around 0 or 180 in hue in HSV (wraps around)
    return ((h <= 10 or h >= 170) and s > 100 and v > 50)

def is_white(hsv_pixel):
    h, s, v = hsv_pixel
    # White: low saturation, high brightness
    return (s < 30 and v > 200)

def identify_flag(image_path):
```

Figure 15: Image 1

```
Cut this cell (X) y_flag(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (300, 200))
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    top_half = hsv[0:100, :, :]
    bottom_half = hsv[100:200, :, :]

    # Count red and white pixels in top half
    top_red = np.sum([is_red(pix) for row in top_half for pix in row])
    top_white = np.sum([is_white(pix) for row in top_half for pix in row])

    # Count red and white pixels in bottom half
    bottom_red = np.sum([is_red(pix) for row in bottom_half for pix in row])
    bottom_white = np.sum([is_white(pix) for row in bottom_half for pix in row])

    # Decision logic
    if top_red > top_white and bottom_white > bottom_red:
        return "Indonesia"
    elif top_white > top_red and bottom_red > bottom_white:
        return "Poland"
    else:
        return "Cannot Determine"

# # Example usage
# flag = identify_flag('flag_image.jpg')
# print("Detected flag:", flag)
```

```
[4]: import os

def evaluate_test_set(test_folder, true_label):
    images = [f for f in os.listdir(test_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    total = len(images)
```

Figure 16: Image 9

```python
import os

def evaluate_test_set(test_folder, true_label):
    images = [f for f in os.listdir(test_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    total = len(images)
    correct = 0

    for img_name in images:
        img_path = os.path.join(test_folder, img_name)
        pred = identify_flag(img_path)
        print(f"{img_name}: Predicted={pred}, Expected={true_label}")
        if pred.lower() == true_label.lower():
            correct += 1

    accuracy = (correct / total) * 100 if total > 0 else 0
    print(f"\nAccuracy on {total} images from '{test_folder}': {accuracy:.2f}%")

# Example usage:
evaluate_test_set('indonesia_test_set', 'Indonesia')
print("\n")
evaluate_test_set('poland_test_set', 'Poland')
```

Figure 17: Image 7

```
flag_rotated_87.jpg: Predicted=Poland, Expected=Indonesia
flag_rotated_88.jpg: Predicted=Poland, Expected=Indonesia
flag_rotated_89.jpg: Predicted=Indonesia, Expected=Indonesia
flag_rotated_9.jpg: Predicted=Cannot Determine, Expected=Indonesia
flag_rotated_90.jpg: Predicted=Poland, Expected=Indonesia
flag_rotated_91.jpg: Predicted=Indonesia, Expected=Indonesia
flag_rotated_92.jpg: Predicted=Poland, Expected=Indonesia
flag_rotated_93.jpg: Predicted=Indonesia, Expected=Indonesia
flag_rotated_94.jpg: Predicted=Indonesia, Expected=Indonesia
flag_rotated_95.jpg: Predicted=Poland, Expected=Indonesia
flag_rotated_96.jpg: Predicted=Indonesia, Expected=Indonesia
flag_rotated_97.jpg: Predicted=Poland, Expected=Indonesia
flag_rotated_98.jpg: Predicted=Indonesia, Expected=Indonesia
flag_rotated_99.jpg: Predicted=Indonesia, Expected=Indonesia

Accuracy on 100 images from 'indonesia_test_set': 57.00%
```

Figure 18: Image 2

```
flag_rotated_85.jpg: Predicted=Indonesia, Expected=Poland
flag_rotated_86.jpg: Predicted=Poland, Expected=Poland
flag_rotated_87.jpg: Predicted=Indonesia, Expected=Poland
flag_rotated_88.jpg: Predicted=Poland, Expected=Poland
flag_rotated_89.jpg: Predicted=Indonesia, Expected=Poland
flag_rotated_9.jpg: Predicted=Poland, Expected=Poland
flag_rotated_90.jpg: Predicted=Poland, Expected=Poland
flag_rotated_91.jpg: Predicted=Indonesia, Expected=Poland
flag_rotated_92.jpg: Predicted=Poland, Expected=Poland
flag_rotated_93.jpg: Predicted=Indonesia, Expected=Poland
flag_rotated_94.jpg: Predicted=Poland, Expected=Poland
flag_rotated_95.jpg: Predicted=Poland, Expected=Poland
flag_rotated_96.jpg: Predicted=Poland, Expected=Poland
flag_rotated_97.jpg: Predicted=Poland, Expected=Poland
flag_rotated_98.jpg: Predicted=Indonesia, Expected=Poland
flag_rotated_99.jpg: Predicted=Poland, Expected=Poland

Accuracy on 100 images from 'poland_test_set': 52.00%
```

Figure 19: Image 8

YOLO SUMMARY

1. Introduction: What is YOLO and Why YOLOv5?

Object detection is one of the core tasks in computer vision, used in areas such as autonomous vehicles, surveillance, and robotics. Traditional detectors often use a two-stage process (region proposal + classification), but YOLO (You Only Look Once) changed the game by performing both tasks in a single neural network pass.

YOLOv5, released in May 2020 by Ultralytics, is the 5th version in the YOLO family. Although not officially authored by the original YOLO creators, it became widely adopted due to its ease of use, high performance, and being implemented in PyTorch (unlike YOLOv4, which uses Darknet).

Key Architectural Features of YOLOv5

- **Backbone**: Extracts visual features from the input image. YOLOv5 uses CSPDarknet (Cross Stage Partial connections) for efficient and robust feature extraction.

- **Neck**: Enhances feature maps from the backbone using PANet (Path Aggregation Network), which helps detect objects of various sizes.
- **Head**: Predicts bounding boxes, objectness scores, and class probabilities.

Figure 20: YOLOv5 Architecture Overview

- **Efficient Feature Extraction with CSP**

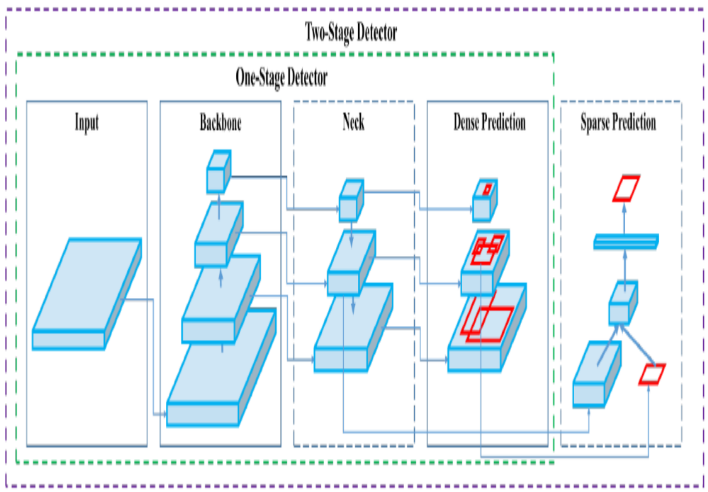


Figure 21: CSP Feature Extraction

- **Enhanced Feature Aggregation with PANet**
  - YOLOv5 uses PA-Net (Path Aggregation Network) in the Neck to combine low- and high-level features.
  - This improves object detection, especially for small objects.

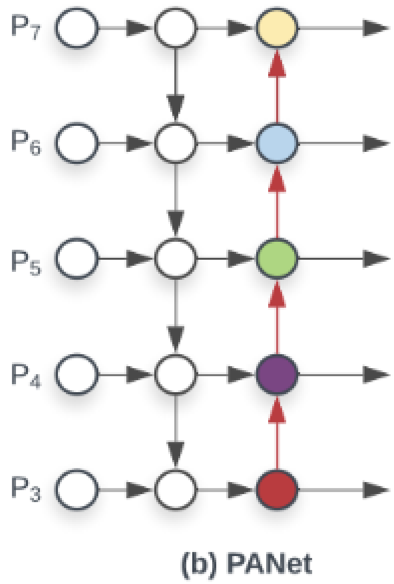


Figure 22: PANet Architecture

YOLOv5 Labelling tools

Roboflow Labeling and exporting custom datasets directly compatible with YOLOv5 training

Conclusion

YOLOv5 has emerged as a significant advancement in object detection, demonstrating a compelling balance of speed, accuracy, and user-friendliness.

GitHub Repository