Assignment 2

Kartikey Agarwal

Task Insights

Working through Assignment-2 offered hands-on experience with several core concepts in image processing and computer vision:

1 Task 1: Image Processing Fundamentals

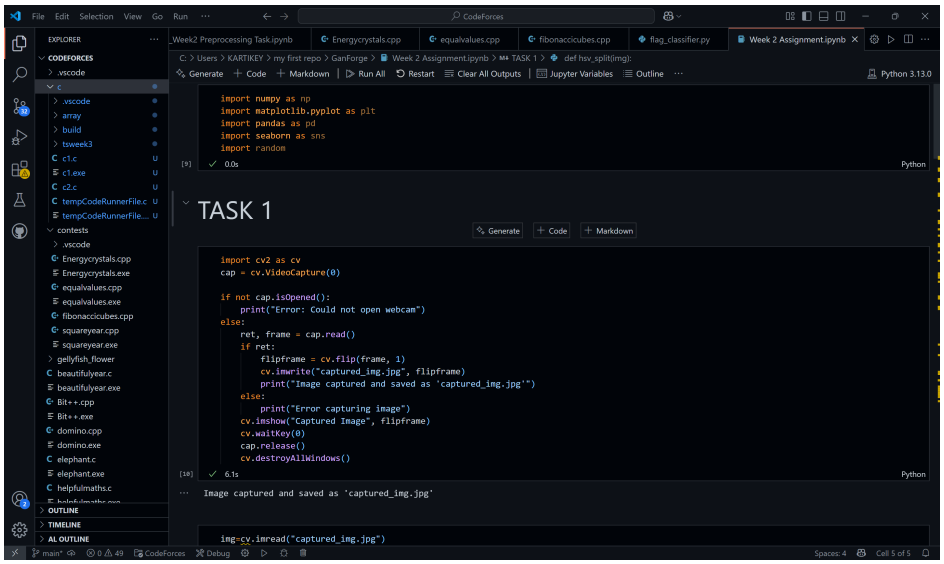
- Captured real-time webcam data using OpenCV and applied various transformations in Python.
- Converted images between color spaces (RGB $\rightarrow$ HSV, LAB), which is useful for tasks like segmentation or analysis.
- Visualized individual channels such as Hue, Saturation, Value, and LAB components.
- Applied histogram equalization to enhance contrast in grayscale images, improving visual clarity.
- Explored edge detection using Laplacian and Scharr filters, observing how different kernels affect output.
- Used posterization and binary thresholding to simplify images for further processing.
- Added and reduced salt-and-pepper noise with a median filter, and performed sharpening using unsharp masking.

Key Python Functions

```
def hsv_split(img):
    hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    h, s, v = cv.split(hsv)
    return h, s, v

def histogram_equalization(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    eq = cv.equalizeHist(gray)
    return gray, eq
```

```
def binary_inversion(img):
    threshold_value = 127
    max_value = 255
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, binary = cv.threshold(gray, threshold_value, max_value,
                  cv.THRESH_BINARY_INV)
    return binary

def posterize(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    posterized = (gray // 64) * 85
    return posterized

def edge_detection(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    blurred_image = cv.GaussianBlur(gray, (3, 3), 0)
    laplacian = cv.Laplacian(blurred_image, cv.CV_64F)
    laplacian_abs = cv.convertScaleAbs(laplacian)
    Gx = cv.Scharr(gray, cv.CV_64F, 1, 0)
    Gy = cv.Scharr(gray, cv.CV_64F, 0, 1)
    gradient_magnitude = cv.magnitude(Gx, Gy)
    return laplacian_abs, gradient_magnitude

def median_filter_denoise(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    noisy = gray.copy()
    h, w = gray.shape
    num_noise = int(h * w * 0.05)
    for _ in range(num_noise):
        i = random.randint(0, h - 1)
        j = random.randint(0, w - 1)
        noisy[i, j] = 255 if random.random() < 0.5 else 0
    denoised = cv.medianBlur(noisy, 3)
    return noisy, denoised

def unsharp_masking(img):
    amount = 1.5
    blur_ksize = (5, 5)
    sigma = 1.0
    blur = cv.GaussianBlur(img, blur_ksize, sigma)
    sharpened = cv.addWeighted(img, 1 + amount, blur, -amount, 0)
    return sharpened

def convert_to_lab(img):
    lab = cv.cvtColor(img, cv.COLOR_BGR2Lab)
    l, a, b = cv.split(lab)
```

```
return l, a, b
```



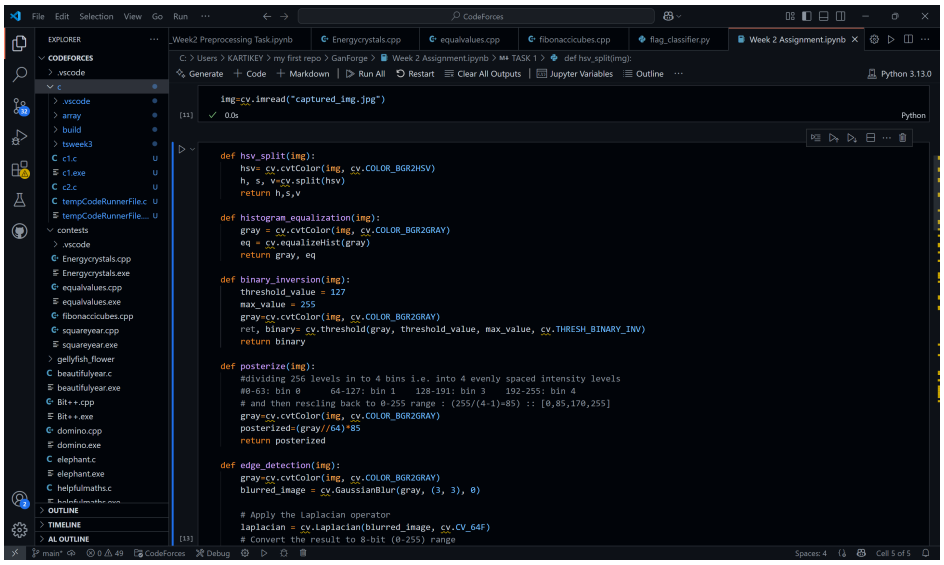
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import random
```

TASK 1

```
import cv2 as cv
cap = cv.VideoCapture(0)

if not cap.isOpened():
    print("Error: Could not open webcam")
else:
    ret, frame = cap.read()
    if ret:
        flipframe = cv.flip(frame, 1)
        cv.imwrite("captured_img.jpg", flipframe)
        print("Image captured and saved as 'captured_img.jpg'")
    else:
        print("Error capturing image")
    cv.imshow("Captured Image", flipframe)
    cv.waitKey(0)
    cap.release()
    cv.destroyAllWindows()
```

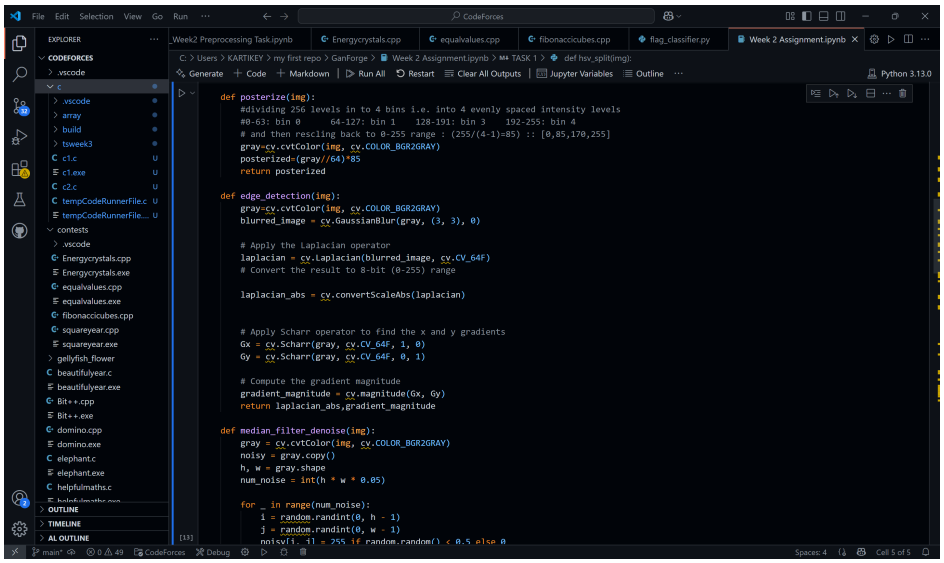
```
Image captured and saved as 'captured_img.jpg'
```

```
img=cv.imread("captured_img.jpg")
```

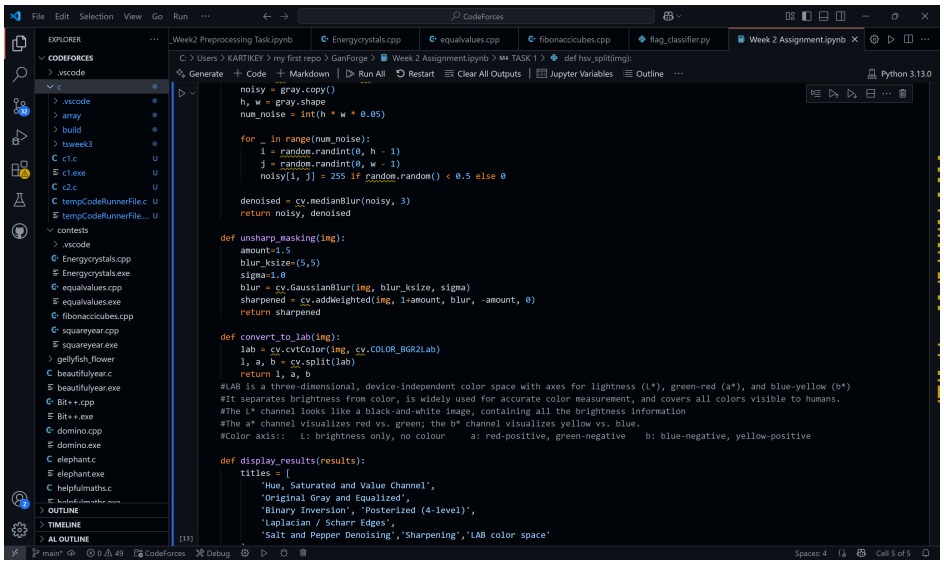


```
img=cv.imread("captured_img.jpg")
```

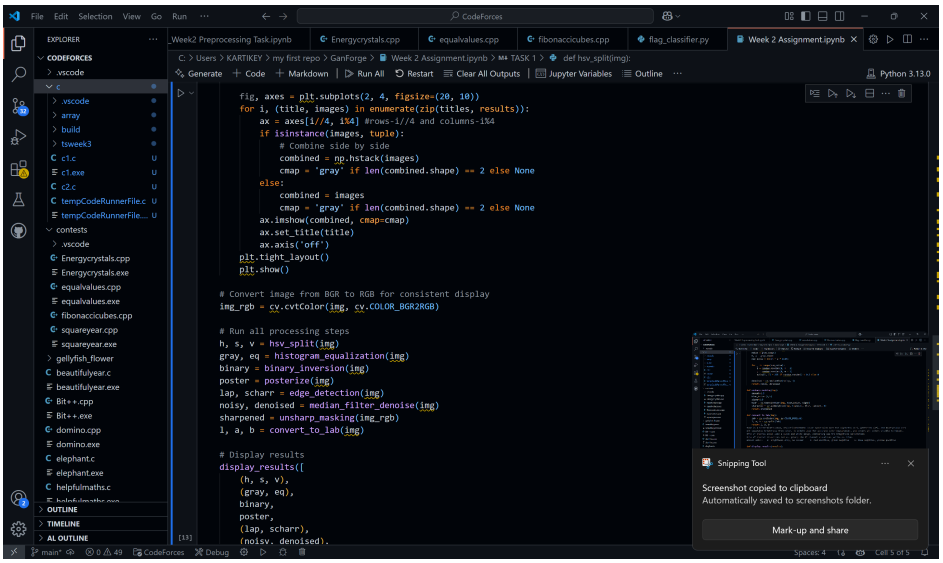
```
def hsv_split(img):
    hsv= cv.cvtColor(img, cv.COLOR_BGR2HSV)
    h, s, v=cv.split(hsv)
    return h,s,v

def histogram_equalization(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    eq = cv.equalizeHist(gray)
    return gray, eq

def binary_inversion(img):
    threshold_value = 127
    max_value = 255
    gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, binary= cv.threshold(gray, threshold_value, max_value, cv.THRESH_BINARY_INV)
    return binary

def posterize(img):
    #dividing 256 levels in to 4 bins i.e. into 4 evenly spaced intensity levels
    #0-63: bin 0      64-127: bin 1    128-191: bin 3    192-255: bin 4
    # and then rescling back to 0-255 range : (255/(4-1)*85) :: [0,85,170,255]
    gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    posterized=(gray//64)*85
    return posterized

def edge_detection(img):
    gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    blurred_image = cv.GaussianBlur(gray, (3, 3), 0)

    # Apply the Laplacian operator
    laplacian = cv.Laplacian(blurred_image, cv.CV_64F)
    # Convert the result to 8-bit (0-255) range
```

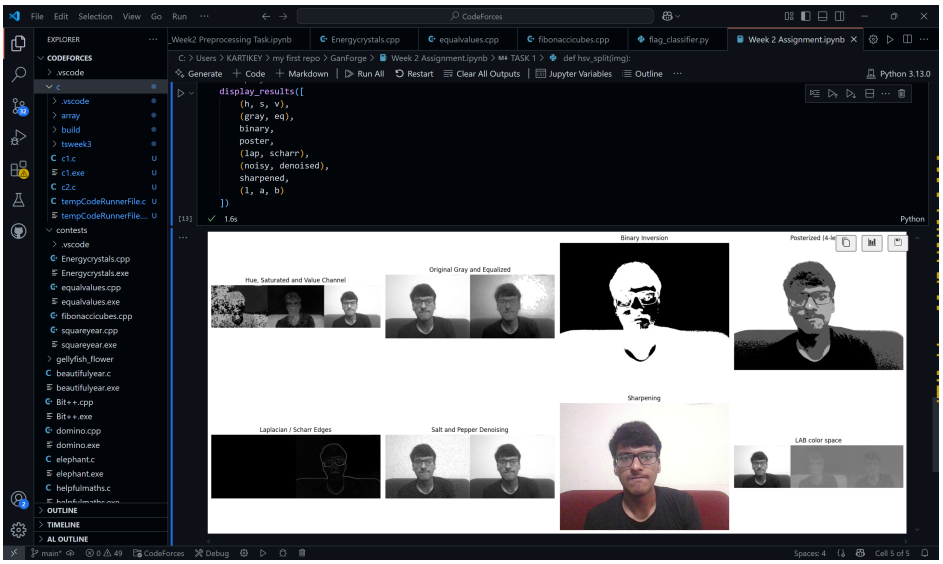


```
def posterize(img):
    #dividing 256 levels in to 4 bins i.e. into 4 evenly spaced intensity levels
    #0-63: bin 0      64-127: bin 1     128-191: bin 3    192-255: bin 4
    # and then rescling back to 0-255 range : (255/(4-1)=85) :: [0,85,170,255]
    gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    posterized=(gray//64)*85
    return posterized

def edge_detection(img):
    gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    blurred_image = cv.GaussianBlur(gray, (3, 3), 0)

    # Apply the Laplacian operator
    laplacian = cv.Laplacian(blurred_image, cv.CV_64F)
    # Convert the result to 8-bit (0-255) range

    laplacian_abs = cv.convertScaleAbs(laplacian)


    # Apply Scharr operator to find the x and y gradients
    Gx = cv.Scharr(gray, cv.CV_64F, 1, 0)
    Gy = cv.Scharr(gray, cv.CV_64F, 0, 1)

    # Compute the gradient magnitude
    gradient_magnitude = cv.magnitude(Gx, Gy)
    return laplacian_abs,gradient_magnitude

def median_filter_denoise(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    noisy = gray.copy()
    h, w = gray.shape
    num_noise = int(h * w * 0.05)

    for _ in range(num_noise):
        i = random.randint(0, h - 1)
        j = random.randint(0, w - 1)
        noisy[i, j] = 255 if random.random() < 0.5 else 0
```



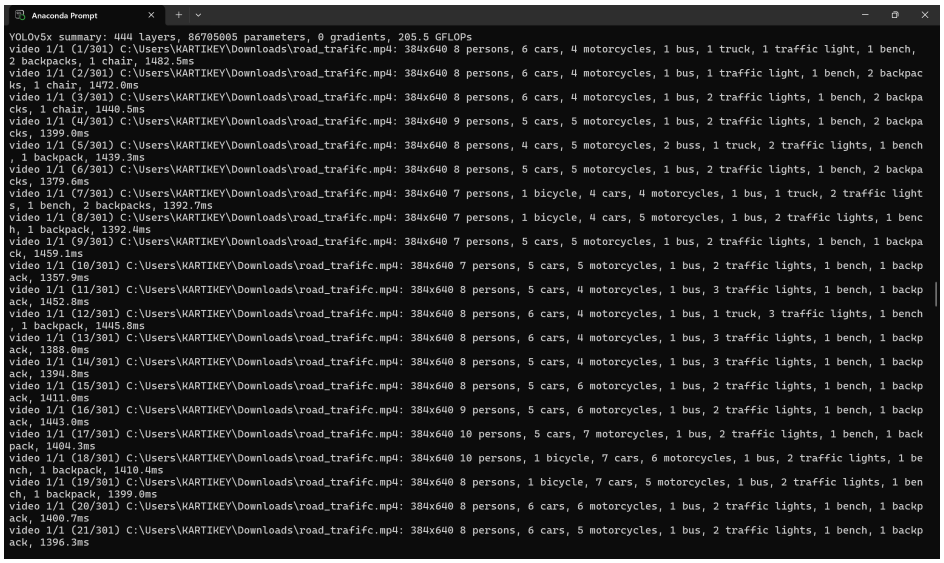
```
    noisy = gray.copy()
    h, w = gray.shape
    num_noise = int(h * w * 0.05)

    for _ in range(num_noise):
        i = random.randint(0, h - 1)
        j = random.randint(0, w - 1)
        noisy[i, j] = 255 if random.random() < 0.5 else 0

    denoised = cv.medianBlur(noisy, 3)
    return noisy, denoised

def unsharp_masking(img):
    amount=1.5
    blur_ksize=(5,5)
    sigma=1.0
    blur = cv.GaussianBlur(img, blur_ksize, sigma)
    sharpened = cv.addWeighted(img, 1+amount, blur, -amount, 0)
    return sharpened

def convert_to_lab(img):
    lab = cv.cvtColor(img, cv.COLOR_BGR2Lab)
    l, a, b = cv.split(lab)
    return l, a, b
#LAB is a three-dimensional, device-independent color space with axes for lightness (L*), green-red (a*), and blue-yellow (b*)
#It separates brightness from color, is widely used for accurate color measurement, and covers all colors visible to humans.
#The L* channel looks like a black-and-white image, containing all the brightness information
#The a* channel visualizes red vs. green; the b* channel visualizes yellow vs. blue.
#Color axis::   L: brightness only, no colour     a: red-positive, green-negative     b: blue-negative, yellow-positive

def display_results(results):
    titles = [
        'Hue, Saturated and Value Channel',
        'Original Gray and Equalized',
        'Binary Inversion', 'Posterized (4-level)',
        'Laplacian / Scharr Edges',
        'Salt and Pepper Denoising','Sharpening','LAB color space'
```





2 Task 2: YOLOv5 Object Detection

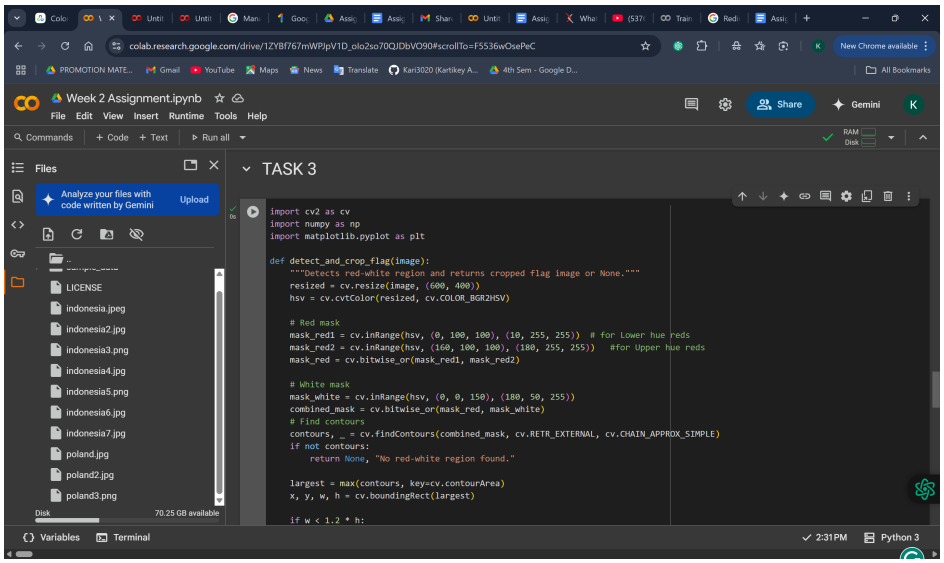
- Implemented object detection using the YOLOv5 model, following a tutorial.
- Tested the model on real-time webcam input and pre-recorded video.
- Detected various objects (persons, bottles, laptops, chairs, etc.).
- Model generally performed well, but sometimes misclassified or falsely detected objects (false positives).
- Some uncommon objects were not detected, likely due to limitations in the pre-trained dataset.


Anaconda Prompt
YOLOv5x summary: 444 layers, 86705005 parameters, 0 gradients, 205.5 GFLOPs
video 1/1 (1/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 4 motorcycles, 1 bus, 1 truck, 1 traffic light, 1 bench, 2 backpacks, 1 chair, 1482.5ms
video 1/1 (2/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 4 motorcycles, 1 bus, 1 traffic light, 1 bench, 2 backpacks, 1 chair, 1472.0ms
video 1/1 (3/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 4 motorcycles, 1 bus, 2 traffic lights, 1 bench, 2 backpacks, 1 chair, 1440.5ms
video 1/1 (4/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 9 persons, 5 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 2 backpacks, 1399.0ms
video 1/1 (5/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 4 cars, 5 motorcycles, 2 buss, 1 truck, 2 traffic lights, 1 bench, 1 backpack, 1439.3ms
video 1/1 (6/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 5 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 2 backpacks, 1379.6ms
video 1/1 (7/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 7 persons, 1 bicycle, 4 cars, 4 motorcycles, 1 bus, 1 truck, 2 traffic lights, 1 bench, 2 backpacks, 1392.7ms
video 1/1 (8/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 7 persons, 1 bicycle, 4 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1392.4ms
video 1/1 (9/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 7 persons, 5 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1459.1ms
video 1/1 (10/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 7 persons, 5 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1357.9ms
video 1/1 (11/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 5 cars, 4 motorcycles, 1 bus, 3 traffic lights, 1 bench, 1 backpack, 1452.8ms
video 1/1 (12/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 4 motorcycles, 1 bus, 1 truck, 3 traffic lights, 1 bench, 1 backpack, 1445.8ms
video 1/1 (13/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 4 motorcycles, 1 bus, 3 traffic lights, 1 bench, 1 backpack, 1388.0ms
video 1/1 (14/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 5 cars, 4 motorcycles, 1 bus, 3 traffic lights, 1 bench, 1 backpack, 1394.8ms
video 1/1 (15/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 5 cars, 6 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1411.0ms
video 1/1 (16/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 9 persons, 5 cars, 6 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1443.0ms
video 1/1 (17/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 10 persons, 5 cars, 7 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1404.3ms
video 1/1 (18/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 10 persons, 1 bicycle, 7 cars, 6 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1410.4ms
video 1/1 (19/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 1 bicycle, 7 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1399.0ms
video 1/1 (20/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 6 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1400.7ms
video 1/1 (21/301) C:\Users\KARTIKEY\Downloads\road_trafifc.mp4: 384x640 8 persons, 6 cars, 5 motorcycles, 1 bus, 2 traffic lights, 1 bench, 1 backpack, 1396.3ms

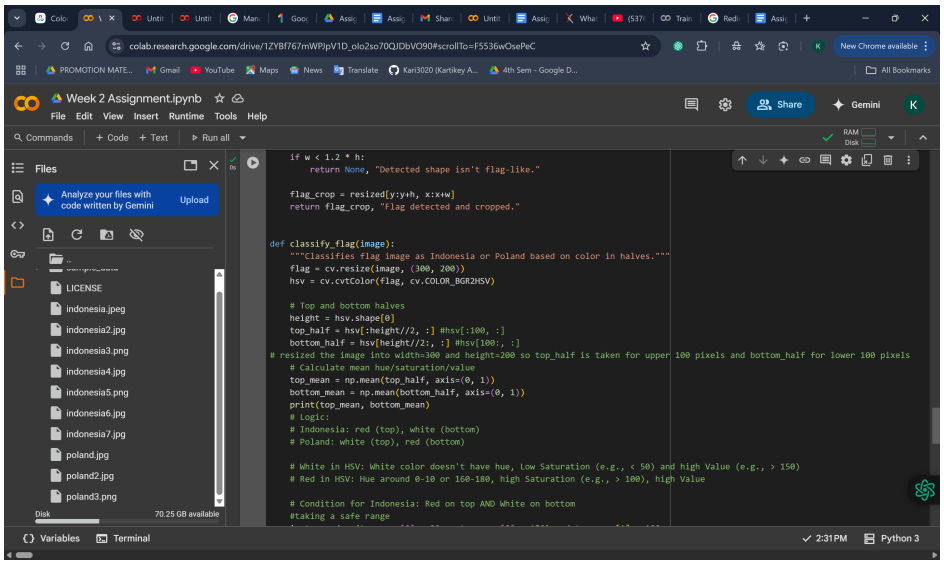

bus 0.78
0.87
car 0.92
bench 0.38
00:00:04
00:00:06
road_trafifc

3 Task 3: Indonesia vs Poland Flag Classification

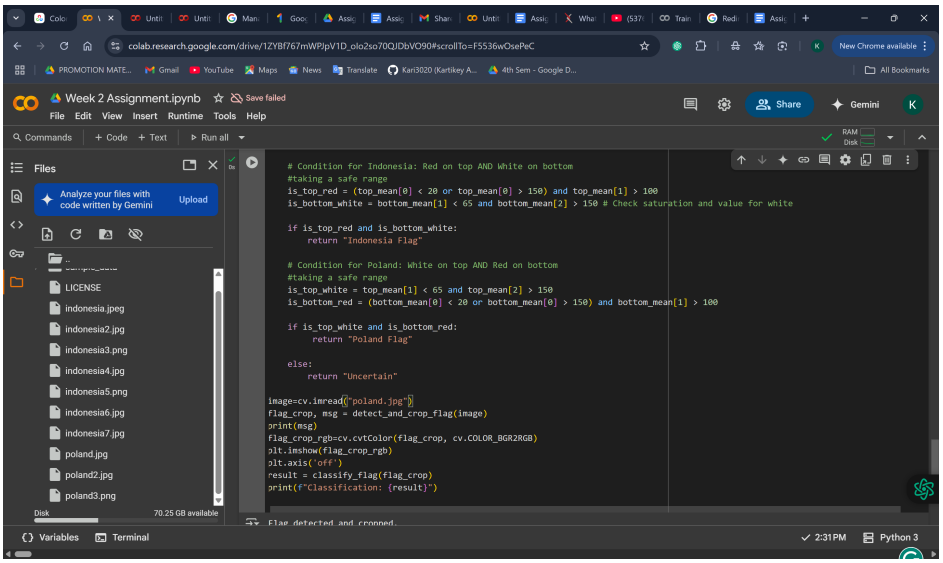
- Developed a Python program with OpenCV to classify an uploaded image as the flag of Indonesia or Poland.
- Both flags have two horizontal stripes (red and white), distinguished by the order:
  - **Indonesia**: Red on top, white on bottom
  - **Poland**: White on top, red on bottom
- Steps:
  1. Image preprocessing: Resize and crop to focus on the central portion.
  2. Color analysis: Convert to HSV, apply color masks to upper and lower halves.
  3. Color ratio comparison: Determine dominant color in each half to classify the flag.
- Classification worked well in clean conditions, but lighting, noise, and orientation could cause errors.
- A YOLO-based approach could improve robustness but would require a dedicated dataset.

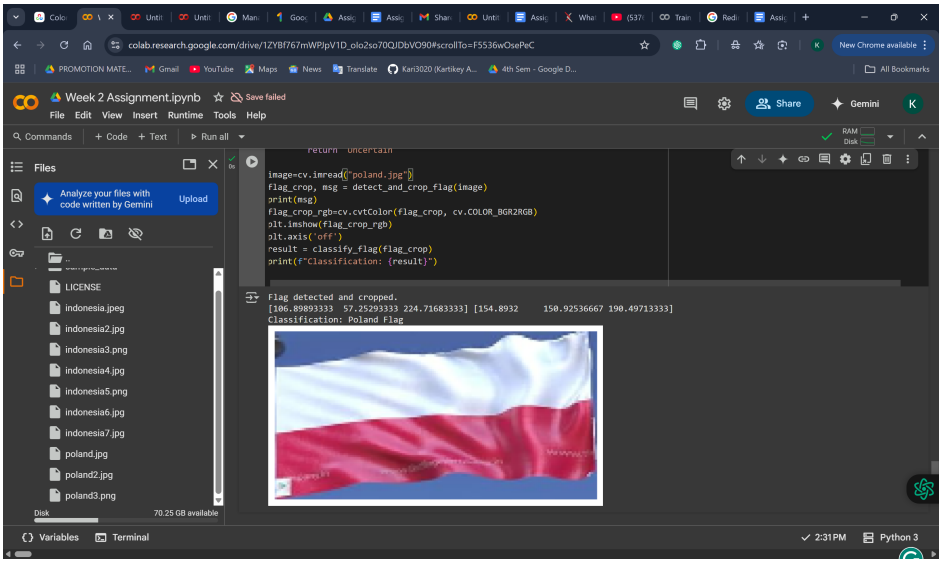



```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

def detect_and_crop_flag(image):
    """Detects red-white region and returns cropped flag image or None."""
    resized = cv.resize(image, (600, 400))
    hsv = cv.cvtColor(resized, cv.COLOR_BGR2HSV)

    # Red mask
    mask_red1 = cv.inRange(hsv, (0, 100, 100), (10, 255, 255))  # for Lower hue reds
    mask_red2 = cv.inRange(hsv, (160, 100, 100), (180, 255, 255))   #for Upper hue reds
    mask_red = cv.bitwise_or(mask_red1, mask_red2)

    # White mask
    mask_white = cv.inRange(hsv, (0, 0, 150), (180, 50, 255))
    combined_mask = cv.bitwise_or(mask_red, mask_white)
    # Find contours
    contours, _ = cv.findContours(combined_mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    if not contours:
        return None, "No red-white region found."

    largest = max(contours, key=cv.contourArea)
    x, y, w, h = cv.boundingRect(largest)

    if w < 1.2 * h:
```




```
    if w < 1.2 * h:
        return None, "Detected shape isn't flag-like."

    flag_crop = resized[y:y+h, x:x+w]
    return flag_crop, "Flag detected and cropped."


def classify_flag(image):
    """Classifies flag image as Indonesia or Poland based on color in halves."""
    flag = cv.resize(image, (300, 200))
    hsv = cv.cvtColor(flag, cv.COLOR_BGR2HSV)

    # Top and bottom halves
    height = hsv.shape[0]
    top_half = hsv[:height//2, :] #hsv[:100, :]
    bottom_half = hsv[height//2:, :] #hsv[100:, :]
# resized the image into width=300 and height=200 so top_half is taken for upper 100 pixels and bottom_half for lower 100 pixels
    # Calculate mean hue/saturation/value
    top_mean = np.mean(top_half, axis=(0, 1))
    bottom_mean = np.mean(bottom_half, axis=(0, 1))
    print(top_mean, bottom_mean)
    # Logic:
    # Indonesia: red (top), white (bottom)
    # Poland: white (top), red (bottom)

    # White in HSV: White color doesn't have hue, Low Saturation (e.g., < 50) and high Value (e.g., > 150)
    # Red in HSV: Hue around 0-10 or 160-180, high Saturation (e.g., > 100), high Value

    # Condition for Indonesia: Red on top AND White on bottom
    #taking a safe range
```





4 Task 4: Understanding YOLOv5

Introduction

YOLOv5 is a popular model for object detection in images and videos. It processes the entire image in one pass, making it faster and suitable for real-time applications.

Key Components

- **Backbone (CSPDarknet):** Extracts features from the input image using Cross-Stage Partial connections.

- **Neck (PANet):** Aggregates features from different layers, enhancing detection at various scales.
- **Head:** Produces the final predictions (object locations and classes).

Training Techniques

- **Data Augmentation:** Mosaic augmentation, combining multiple images for better generalization.
- **Loss Functions:** Advanced loss functions balance localization, classification, and objectness.
- **Anchor Generation:** Uses anchor boxes to predict object sizes and shapes.
- **FP16 Precision:** Reduces memory and speeds up computation.

Transition to PyTorch

YOLOv5 moved from Darknet to PyTorch, making it more accessible and easier to integrate.

Performance and Variants

YOLOv5 comes in several sizes (n, s, m, l, x), offering trade-offs between speed and accuracy for different applications.

Conclusion

YOLOv5 is notable for its speed, accuracy, and ease of use, making it a preferred choice for real-time object detection tasks.

Github link: `https://github.com/Kari3020/GanForge`