# GANForge Assignment 2

Akshat Shahjade
Roll No: 230100

## Task 1

### What I Learned

- HSV and LAB
- Different masks and filters, especially Scharr (found useful)
- Posterizing

These functions where you input an image and get stylized effects are useful for ideation in image classification tasks (e.g., Task 3). For instance, hue provided clear segregation in a Poland flag image.

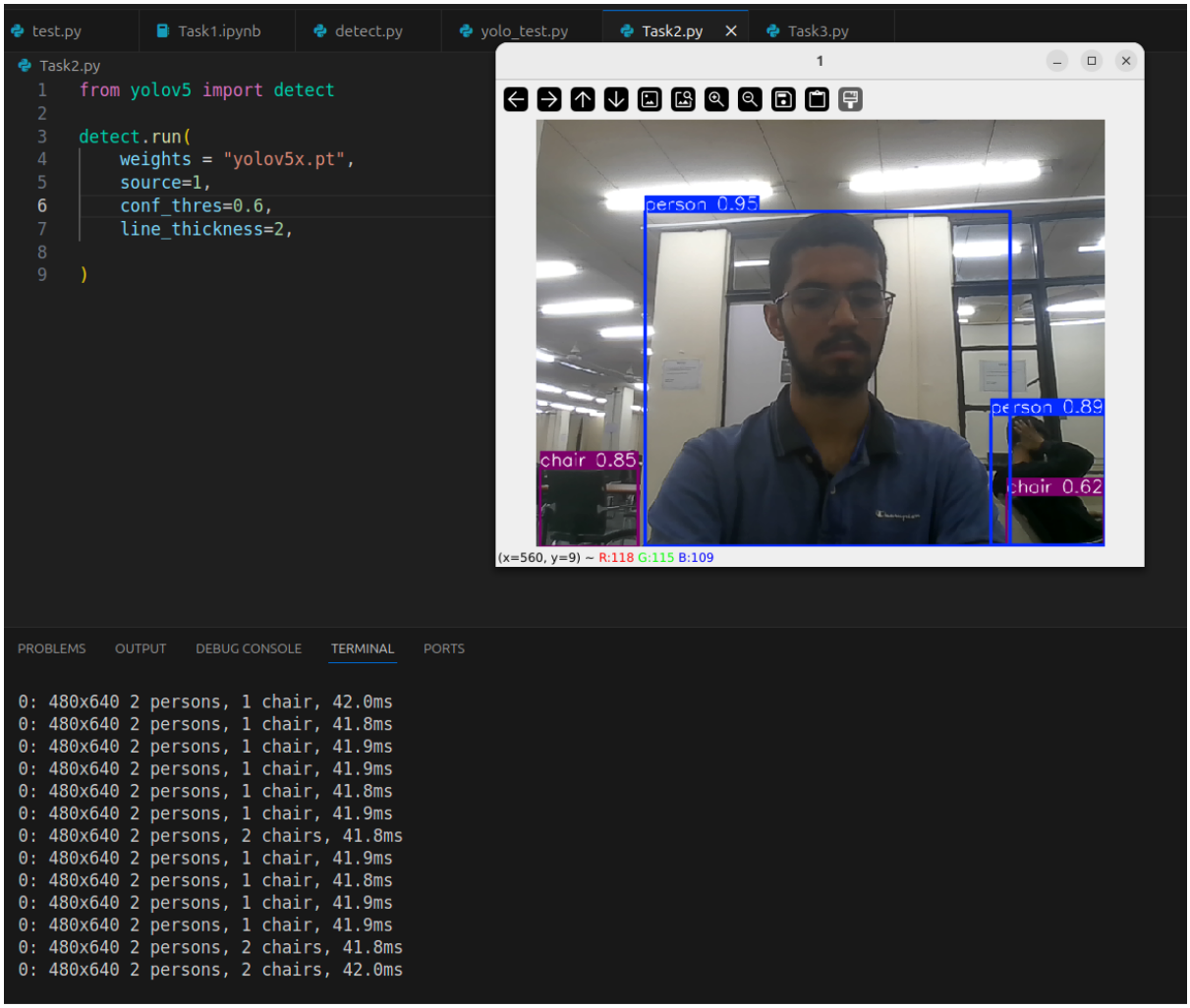

```
    axs[0][0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axs[0][0].set_title("Original")
    hsv,hue,s,val = convert_to_hsv_and_extract_channels(img)
    axs[0][1].imshow(hue, cmap="hsv")
    axs[0][1].set_title("Hue")
    axs[0][2].imshow(s, cmap="gray")
    axs[0][2].set_title("Saturation")
    axs[0][3].imshow(val, cmap="gray")
    axs[0][3].set_title("Value")
    img_gray, img_equalized = grayscale_histogram_equalization(img)
    axs[1][0].imshow(img_gray, cmap="gray")
    axs[1][0].set_title("GrayScale")
    axs[1][1].imshow(img_equalized, cmap="gray")
    axs[1][1].set_title("Equalized")
    img_bin_thresh = binary_inversion_threshold(img, 100)
    axs[1][2].imshow(img_bin_thresh, cmap="gray")
    axs[1][2].set_title("Binary Threshold at 100")
    img_posterized = posterize_to_4_gray_levels(img)
    axs[1][3].imshow(img_posterized, cmap="gray")
    axs[1][3].set_title("4 level Posterized")
    img_laplace, img_scharr = edge_detection_laplacian_scharr(img)
    axs[2][0].imshow(img_laplace, cmap="gray")
    axs[2][0].set_title("Laplacian Filter")
    axs[2][1].imshow(img_scharr, cmap="gray")
    axs[2][1].set_title("Scharr Filter")
    noisy, denoised =  salt_pepper_noise_and_median_filter(img)
    axs[2][2].imshow(cv2.cvtColor(noisy, cv2.COLOR_BGR2RGB))
    axs[2][2].set_title("Salt-And-Pepper Noise Added")
    axs[2][3].imshow(cv2.cvtColor(denoised, cv2.COLOR_BGR2RGB))
    axs[2][3].set_title("Salt-And-Pepper Denoised")
    img_unsharp = unsharp_masking(img)
    axs[3][0].imshow(cv2.cvtColor(img_unsharp, cv2.COLOR_BGR2RGB))
    axs[3][0].set_title("Unsharp Masking")
    lab, l,a,b = convert_to_lab_and_extract_channels(img)
    axs[3][1].imshow(l, cmap="gray")
    axs[3][1].set_title("L")
    axs[3][2].imshow(a, cmap="gray")
    axs[3][2].set_title("A")
```

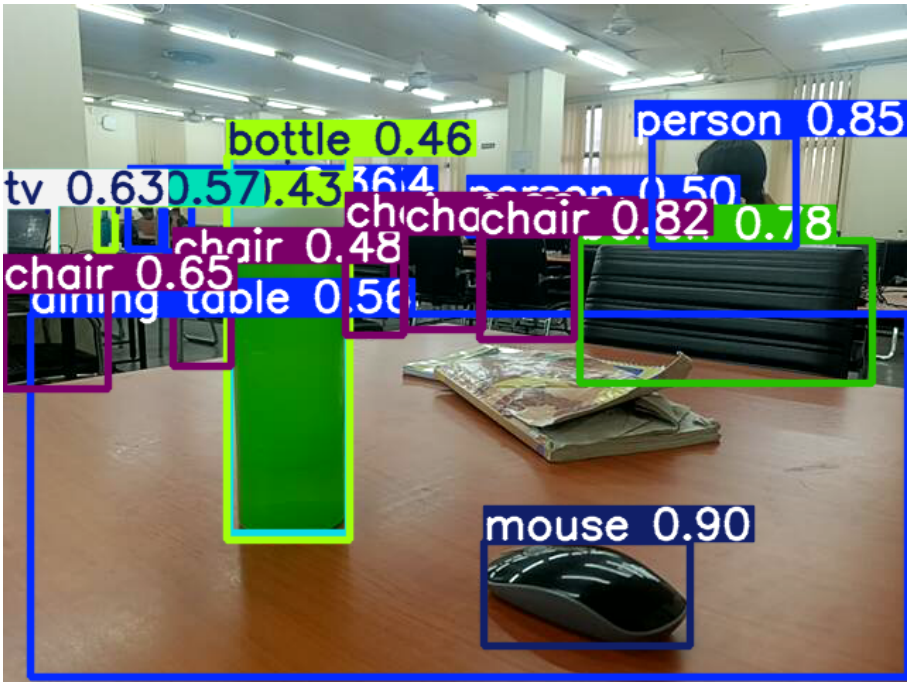
```
img = cv2.imread("task3_input/poland.jpeg")
```

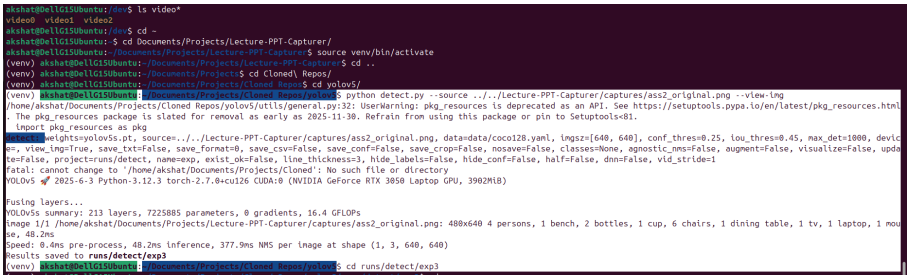
# Task 2

## Learnings

- Learned about `/dev/video*` in Ubuntu and kernel modules.
- Learned how to clone repositories and use them.
- Used DroidCam to connect my mobile to my laptop via shared Wi-Fi, which allowed me to use `/dev/video2` as a video source.






# Task 3

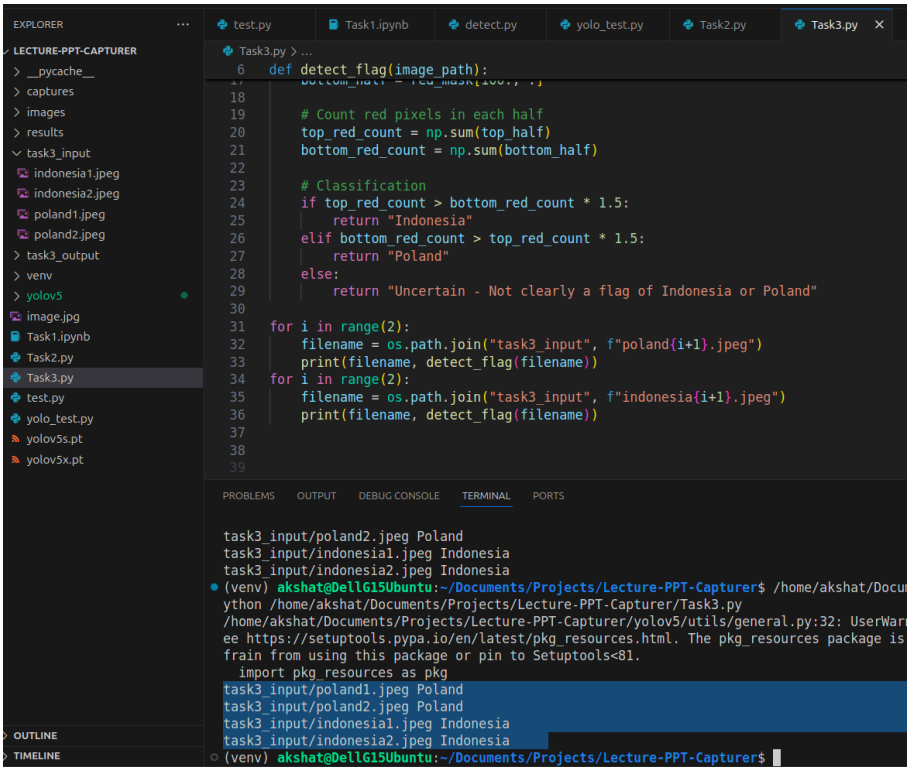
## Code

```
from yolov5 import detect
import os
import cv2
import numpy as np

def detect_flag(image_path):
    # Load the image
    image = cv2.imread(image_path)
    image = cv2.resize(image, (300, 200))  # Normalize size
    hue, _, val = cv2.split(cv2.cvtColor(image, cv2.COLOR_BGR2HSV
        ↪ ))

    # Masking to get only red part (red -> 160-179 in OpenCV)
    # I am ignoring 0-10 as I get better results that way
    red_mask = ((hue > 160)) & (val > 50)

    # Split into top and bottom halves
```

```
    top_half = red_mask[:100, :]
    bottom_half = red_mask[100:, :]

    # Count red pixels in each half
    top_red_count = np.sum(top_half)
    bottom_red_count = np.sum(bottom_half)

    # Classification
    if top_red_count > bottom_red_count * 1.5:
        return "Indonesia"
    elif bottom_red_count > top_red_count * 1.5:
        return "Poland"
    else:
        return "Uncertain - Not clearly a flag of Indonesia or
            ↪ Poland"

for i in range(2):
    filename = os.path.join("task3_input", f"poland{i+1}.jpeg")
    print(filename, detect_flag(filename))

for i in range(2):
    filename = os.path.join("task3_input", f"indonesia{i+1}.jpeg"
        ↪ )
    print(filename, detect_flag(filename))
```

## Logic Explanation

- Convert the image to HSV and extract the hue.
- Create a red mask that highlights red pixels with non-trivial value.
- Split the image into top and bottom halves.
- Count red pixels in each half.
- Classify based on which half has significantly more red pixels.
- Include an “Uncertain” case if results are too close.




# Task 4

## Summary

This paper is more of an overview of the YOLOv5 architecture, covering what led to its popularity and what makes it efficient.

Unlike 2-stage object detectors, YOLO (You Only Look Once) predicts bounding boxes and class probabilities in a single CNN pass.

YOLOv5 was ported from DarkNet (a flexible but complex environment) to PyTorch, which facilitates rapid experimentation and integration.

While architecture attracts attention, training techniques also matter:

- Data Augmentation (e.g., mosaic augmentation)
- Careful loss function selection


YOLOv5 is well-suited for custom datasets due to how it generates anchor boxes using K-means and dataset-specific statistics.

For speed, it implements:

- Cross Stage Partial connections (CSP)
- Mixed precision (float16 instead of float32)

The model is divided into:

- Backbone
- Neck
- Head

YOLOv5 variants:

- `n`, `s`, `m`, `l`, `x` – ordered from smallest (fastest) to largest (most accurate)
- Accuracy saturates with larger models; best trade-off is often with `s` or `m`

**Conclusion:** YOLOv5's strengths lie in efficiency, accuracy, and accessibility, making it suitable for a broad user base.