Task 1: Learnings

Learned to separate an image into different channels like HSV, LAB, etc. With histogram equalisation, I understood how pixel intensities are redistributed to improve contrast. Learned how to convert a default BGR image in OpenCV to various formats such as grayscale and RGB.

Through binary inversion and posterization, I learnt how images are quantified into intensity levels. Also, I explored how filters like Laplacian and Scharr are used for edge detection, and techniques for removing salt-and-pepper noise and applying unsharp masking.

All of the above tasks help in forming various models related to computer vision, each with varying uses.

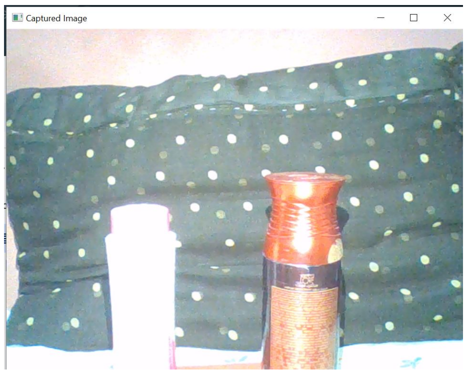


Figure 1: captured image

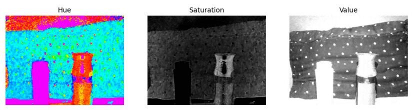


Figure 2: HSV colour channels

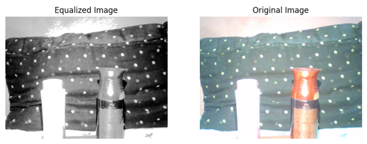


Figure 3: Histogram Equalization

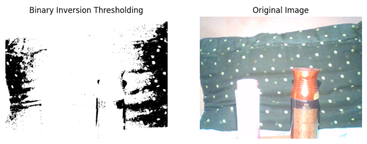


Figure 4: Binary Thresholding

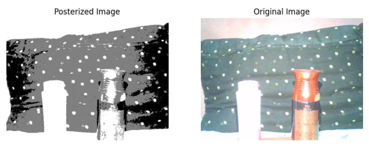


Figure 5: Posterized image

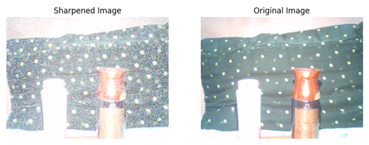


Figure 6: Sharpening of an Image

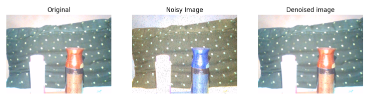


Figure 7: Denoising the Image

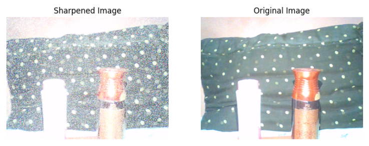


Figure 8: Sharpening of an Image

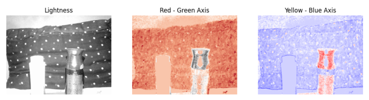
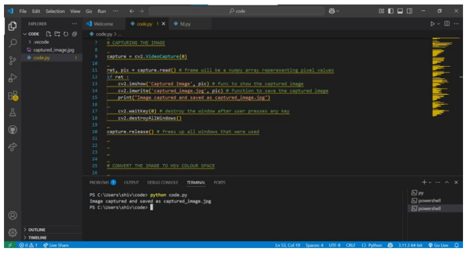


Figure 9: RGB to LAB color space



```python
# CAPTURING THE IMAGE

capture = cv2.VideoCapture(0)

ret, pic = capture.read() # frame will be a numpy array repersenting pixel values
if ret :
    cv2.imshow('Captured Image', pic) # func to show the captured image
    cv2.imwrite('captured_image.jpg', pic) # function to save the captured image
    print("Image captured and saved as captured_image.jpg")

    cv2.waitKey(0) # destroy the window after user presses any key
    cv2.destroyAllWindows()

capture.release() # frees up all windows that were used




# CONVERT THE IMAGE TO HSV COLOUR SPACE
```

```
PS C:\Users\shiv\code> python code.py
Image captured and saved as captured_image.jpg
PS C:\Users\shiv\code>
```

Figure 10: Terminal Screenshot

Task 2

LEARNINGS

- Cloning `yolov5` repository.
- Using `yolov5` for object detection.
- Gained experience on how to operate with Anaconda Prompt.

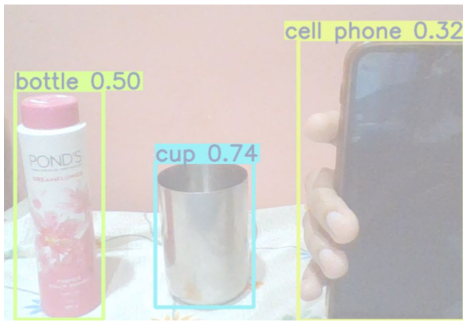
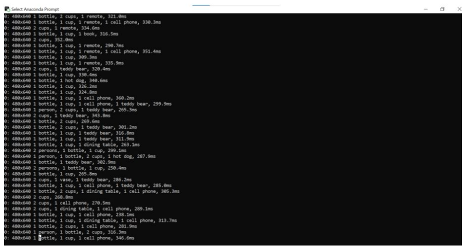


Figure 11: Objects Detected by YOLOv5



```
0: 480x640 1 bottle, 2 cups, 1 remote, 321.0ms
0: 480x640 1 bottle, 1 cup, 1 remote, 1 cell phone, 330.3ms
0: 480x640 2 cups, 1 remote, 334.6ms
0: 480x640 1 bottle, 1 cup, 1 book, 316.5ms
0: 480x640 2 cups, 352.0ms
0: 480x640 1 bottle, 1 cup, 1 remote, 290.7ms
0: 480x640 1 bottle, 1 cup, 1 remote, 1 cell phone, 351.4ms
0: 480x640 1 bottle, 1 cup, 309.3ms
0: 480x640 1 bottle, 1 cup, 1 remote, 335.9ms
0: 480x640 2 cups, 1 teddy bear, 320.4ms
0: 480x640 1 bottle, 1 cup, 330.4ms
0: 480x640 1 bottle, 1 hot dog, 340.6ms
0: 480x640 1 bottle, 1 cup, 326.2ms
0: 480x640 1 bottle, 1 cup, 324.8ms
0: 480x640 1 bottle, 1 cup, 1 cell phone, 360.2ms
0: 480x640 1 bottle, 1 cup, 1 cell phone, 1 teddy bear, 299.9ms
0: 480x640 1 person, 2 cups, 1 teddy bear, 265.3ms
0: 480x640 2 cups, 1 teddy bear, 343.8ms
0: 480x640 1 bottle, 2 cups, 269.6ms
0: 480x640 1 bottle, 2 cups, 1 teddy bear, 301.2ms
0: 480x640 1 bottle, 1 cup, 1 teddy bear, 316.8ms
0: 480x640 1 bottle, 1 cup, 1 teddy bear, 311.9ms
0: 480x640 1 bottle, 1 cup, 1 dining table, 263.1ms
0: 480x640 2 persons, 1 bottle, 1 cup, 299.1ms
0: 480x640 1 person, 1 bottle, 2 cups, 1 hot dog, 287.9ms
0: 480x640 1 bottle, 1 teddy bear, 302.9ms
0: 480x640 2 persons, 1 bottle, 1 cup, 250.4ms
0: 480x640 1 bottle, 1 cup, 265.8ms
0: 480x640 2 cups, 1 vase, 1 teddy bear, 286.2ms
0: 480x640 1 bottle, 1 cup, 1 cell phone, 1 teddy bear, 285.0ms
0: 480x640 1 bottle, 2 cups, 1 dining table, 1 cell phone, 305.3ms
0: 480x640 2 cups, 268.8ms
0: 480x640 2 cups, 1 cell phone, 270.5ms
0: 480x640 2 cups, 1 dining table, 1 cell phone, 289.1ms
0: 480x640 1 bottle, 1 cup, 1 cell phone, 238.1ms
0: 480x640 1 bottle, 1 cup, 1 dining table, 1 cell phone, 313.7ms
0: 480x640 1 bottle, 2 cups, 1 cell phone, 281.9ms
0: 480x640 1 person, 1 bottle, 2 cups, 316.3ms
0: 480x640 1 bottle, 1 cup, 1 cell phone, 346.6ms
```

Figure 12: Terminal Screenshot

Task 3

LEARNINGS

Gained hands-on experience in using the functions learned in Task 1. Learned how those functions can be applied to create object detection and processing models, and how they can be very useful for practical purposes. Also learned to consider the intricacies involved in applying various functions and the challenges one might face while working with real-life images.

LOGIC USED

My flag detection algorithm can be summarised in the following steps:

1. Applying binary scale inversion.
2. Dividing the flag into upper half and lower half.
3. Calculating the mean of average intensities of both the upper and lower halves.
4. Comparing the intensities of both the halves.
5. The flag with higher intensity in the upper half is classified as **Poland**, and vice versa.

GENERAL COMMENTS

I realise the limitations this model has. It depends heavily on the flag being distributed symmetrically in the image as well as the background being evenly coloured. Moreover, the image should have even lighting for the algorithm to work correctly.

I tried a different method using a Laplacian filter to perform edge detection, then extract the flag and divide it into two halves. However, I encountered certain difficulties with that approach which I couldn't immediately resolve.

Despite this, the algorithm was successful in identifying the flag mentioned in the assignment document. I will aim to improve it further next time.

Case I: Poland Flag

Case II: Indonesia Flag

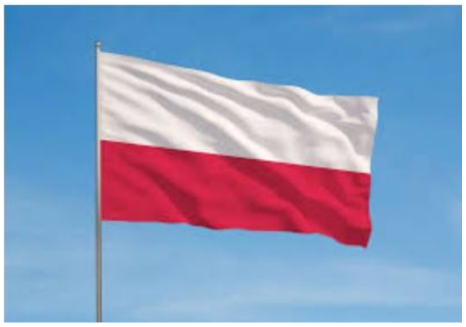
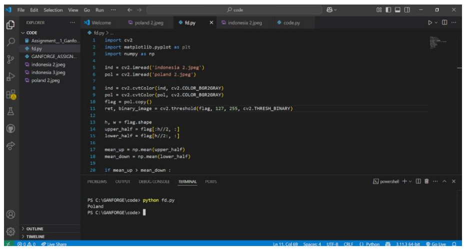


Figure 13: Poland flag that was taken as input[I]



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

ind = cv2.imread('indonesia 2.jpeg')
pol = cv2.imread('poland 2.jpeg')

ind = cv2.cvtColor(ind, cv2.COLOR_BGR2GRAY)
pol = cv2.cvtColor(pol, cv2.COLOR_BGR2GRAY)
flag = pol.copy()
ret, binary_image = cv2.threshold(flag, 127, 255, cv2.THRESH_BINARY)

h, w = flag.shape
upper_half = flag[:h//2, :]
lower_half = flag[h//2:, :]

mean_up = np.mean(upper_half)
mean_down = np.mean(lower_half)

if mean_up > mean_down :
```

```
PS C:\GANFORGE\code> python fd.py
Poland
PS C:\GANFORGE\code>
```

Figure 14: Successfully identified that it was a poland flag[I]

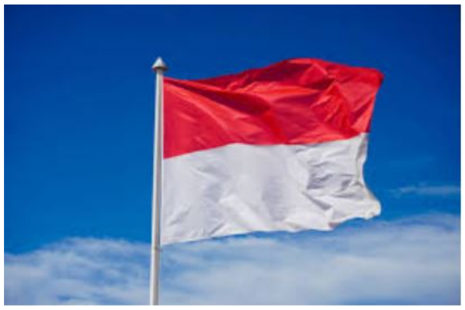
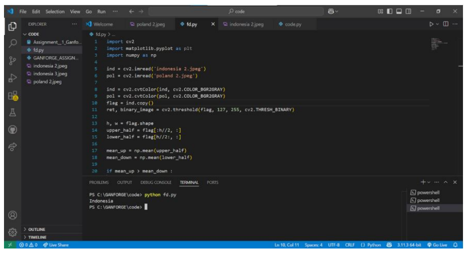


Figure 15: Indonesia FLag that was taken as input[II]



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

ind = cv2.imread('indonesia 2.jpeg')
pol = cv2.imread('poland 2.jpeg')

ind = cv2.cvtColor(ind, cv2.COLOR_BGR2GRAY)
pol = cv2.cvtColor(pol, cv2.COLOR_BGR2GRAY)
flag = ind.copy()
ret, binary_image = cv2.threshold(flag, 127, 255, cv2.THRESH_BINARY)

h, w = flag.shape
upper_half = flag[:h//2, :]
lower_half = flag[h//2:, :]

mean_up = np.mean(upper_half)
mean_down = np.mean(lower_half)

if mean_up > mean_down :
```

```
PS C:\GANFORGE\code> python fd.py
Indonesia
PS C:\GANFORGE\code>
```

Figure 16: Successfully identified that it was an Indonesian flag[II]

Task 4: Summary

Object detection consists of identifying and analyzing images. `YOLOv5` is a distinct algorithm that contrasts with traditional two-stage detection processes. This study evaluates the evolution of the YOLOv5 model, focusing on specific features such as its transition from `Darknet` to `PyTorch`, which contributed to its popularity and ease of use for developers.

How YOLOv5 Works

- **Backbone:** Scans the image, extracts and stores key features.
- **Neck:** Combines and processes features from the backbone.
- **Head:** Generates final predictions from the features.

Techniques Used in Training

- **Data Augmentation:** Introduces variations such as resizing, color modifications, and mosaic augmentation to improve detection performance.
- **Loss Function:** Provides feedback on prediction accuracy. The loss consists of three components:

  $$\text{Loss} = \lambda_1 \cdot L_{\text{cls}} + \lambda_2 \cdot L_{\text{obj}} + \lambda_3 \cdot L_{\text{loc}}$$

  where the λ values balance the contribution of classification, objectness, and localization loss.

From Darknet to PyTorch

Earlier YOLO models were implemented using the `Darknet` framework in C, offering low-level control but making modifications difficult. YOLOv5's transition to PyTorch simplified development and optimization.

Data Augmentation Details

Each image may undergo:

- Size changes
- Color channel alterations
- **Mosaic augmentation:** Combines 4 random images into a collage-like layout. This technique enhances small object detection and exposes the model to diverse spatial patterns.

Bounding Box Anchors

- **YOLOv3:** Uses k-means clustering (and optionally genetic algorithms) to generate anchors that match dataset object sizes.
- **YOLOv5:** Uses predefined anchors and learns offsets during prediction. Proper anchor selection is crucial for accuracy.

Loss Calculation

YOLOv5 uses:

- Binary Cross-Entropy (BCE) for classification and objectness
- Complete Intersection over Union (CIoU) for bounding box regression

16-Bit Floating Point Precision

YOLOv5 can reduce computation from 32-bit to 16-bit precision, significantly improving speed—given compatible hardware. This optimization is still evolving.

CSP Backbone

YOLOv4 and YOLOv5 both use the Cross Stage Partial (CSP) architecture, which splits the feature map and recombines it to:

- Reduce redundant computation
- Improve efficiency
- Mitigate vanishing gradients (inspired by DenseNet)

PA-Net Neck

Both YOLOv4 and YOLOv5 use the **PA-Net** neck to aggregate features across layers, improving multi-scale detection. It's influenced by BiFPN from EfficientDet and builds upon extensive evaluations of neck architectures.

YOLOv5 Model Variants

`YOLOv5n`, `s`, `m`, `l`, and `x` differ in speed and accuracy:

- `n (nano):` Fastest, least accurate
- `x (extra-large):` Most accurate, slowest
- Others represent trade-offs between size, speed, and precision

Choosing the right model depends on the task's real-time vs. accuracy requirements.