2(f)
- PyCharm: Usually 'Darcula' (dark theme)
- Jupyter Notebook: Light theme by default

Insights:

1) Image Capture
The `cv2.VideoCapture()` function captures the image using the system webcam.

2)Image Processing Techniques

GreyScale Conversion:

- Uses `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` to reduce the image to a single channel.
- Removes color information, leaving intensity data only.
- Greyscale images are efficient for edge detection and thresholding.

Black & White Thresholding:

- Uses `cv2.threshold()` to classify pixels as black or white based on a threshold value (127).
- Creates a binary image useful for object segmentation.
- Simplifies object detection or pattern recognition.

16-Level Greyscale Thresholding:

- Reduces greyscale levels to 16 using integer division.
- Coarser grayscale image with visible level steps.
- Reduces data size for simplified image representation.

Sobel Filter:

- Applies gradients in the X and Y directions to detect edges.
- Highlights edges with orientation sensitivity.
- Useful for detecting gradient changes and features.

Canny Edge Detection:

- Detects strong edges using two thresholds (`cv2.Canny`).
- Produces cleaner and sharper edges compared to Sobel.
- Widely used in computer vision for edge-based object detection.

Gaussian Blur:

- Applies a Gaussian filter with kernel size `(5,5)` to reduce noise.
- Smoothens the image.
- Preprocessing step before edge detection to reduce noise sensitivity.

Image Sharpening:

- Uses a kernel to enhance edges and details.
- Makes the image sharper by intensifying edges.
- Combining Gaussian blur and sharpening can enhance details effectively.

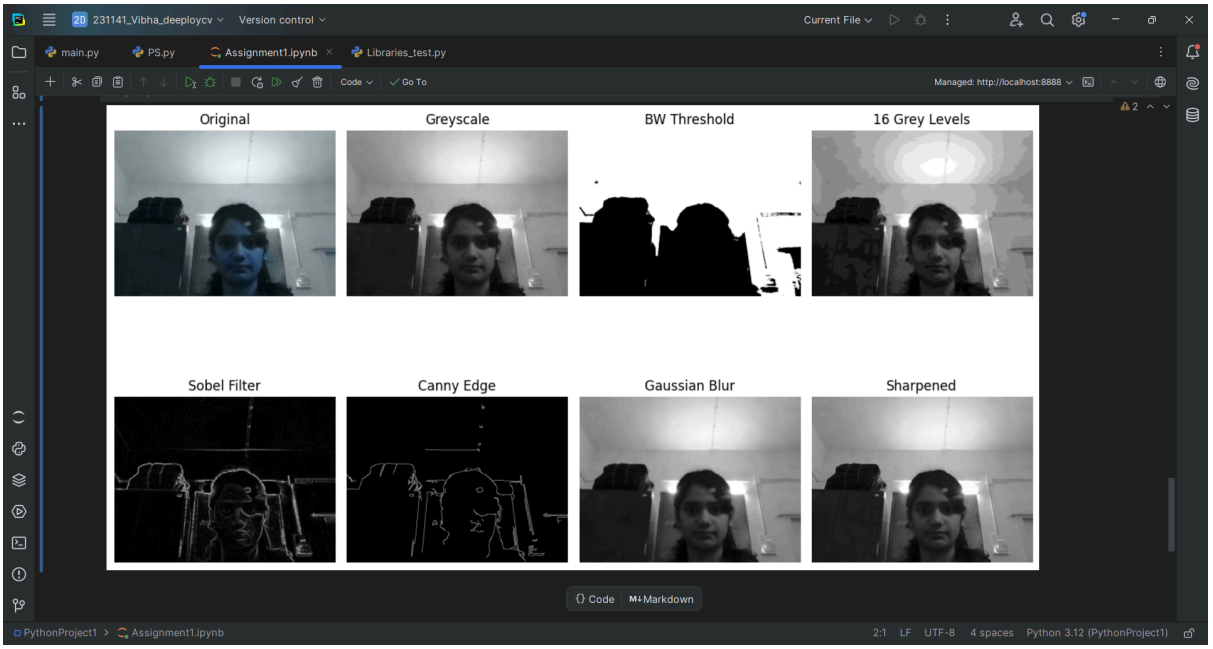
RGB to BGR Conversion:

- Uses `cv2.cvtColor(image, cv2.COLOR_RGB2BGR)` for color channel adjustment.
- OpenCV uses BGR as its default color space, while matplotlib expects RGB.

The Sobel filter highlights gradients.

Canny edges provide clean edge outlines.

Gaussian blur reduces noise but softens sharp edges.



Rotation:

- Images are rotated at different angles.
- **Use Case**: Helps the model recognize objects irrespective of orientation.

Shifting (Translation):

- The image has been shifted along the X and Y axes.
- **Effect**: Simulates object placement variations.

Zooming (Scaling):

- Some images appear larger or smaller due to zooming in/out.
- **Use Case**: Simulates objects at varying distances.

Flipping (Horizontal/Vertical):

- Certain images are flipped horizontally or vertically.
- **Insight**: Horizontal flips are often used for symmetry-aware tasks.

Cropping:

- Parts of the image are cropped to simulate partial visibility.
- **Use Case**: Makes the model robust to occlusions.

Color Augmentations:

- Adjustments in hue, saturation, contrast, and brightness.
- Some images appear brighter, darker, or color-shifted.
- **Effect**: Helps the model generalize to varying lighting conditions.

Blurring:

- Some images appear blurred (Gaussian or motion blur).
- **Use Case**: Simulates real-world noise.

Edge Detection:

- A few images exhibit edge-like structures.
- **Insight**: Can be achieved using filters like Sobel or Canny.

Noise Addition:

- Some images have colored or random noise.
- **Use Case**: Increases model resilience to noisy data.

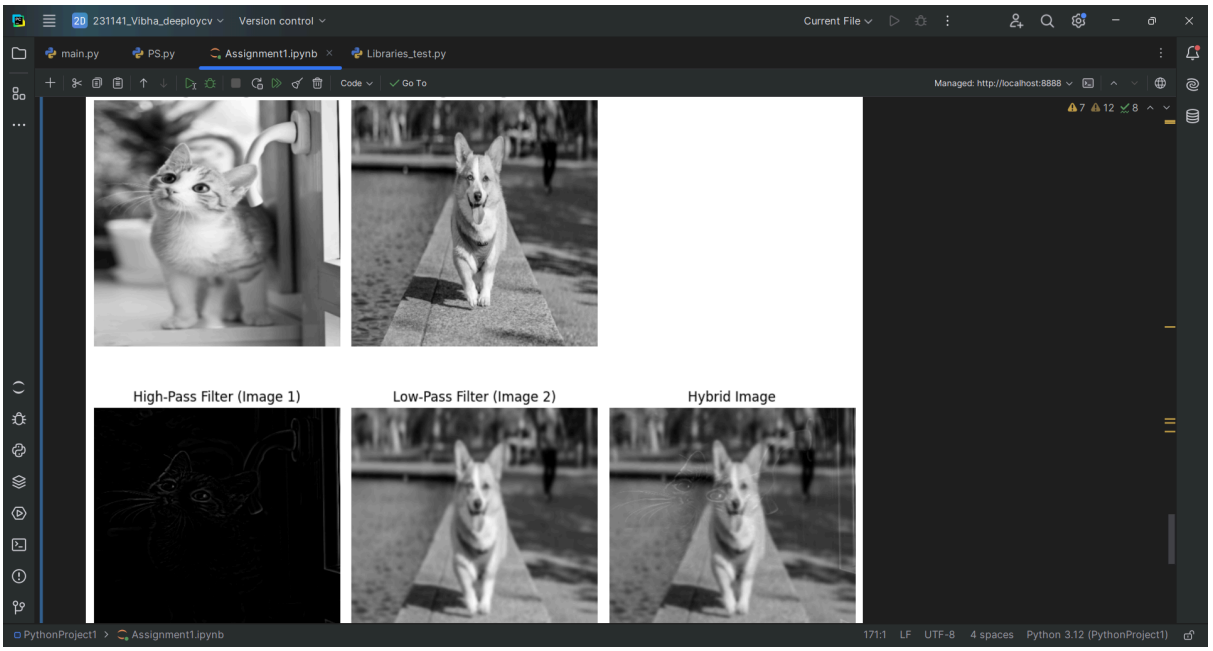
Perspective Transformations:

- Images appear warped or stretched, simulating changes in viewing angles.
- **Insight**: Useful for scene recognition with different perspectives.

Color Mapping:

- False color schemes or colormaps are applied.
- **Use Case**: Common in scientific and thermal imaging.

3)



How High-Pass Filter Works:

1. **Purpose**: The high-pass filter enhances sharp features and retains **high-frequency components** like edges, details, and textures.
2. **Process**:
    - A **Gaussian blur** (low-pass filter) is applied to the original image to smooth it.
    - The blurred (low-frequency) image is subtracted from the original image.
    - The result is the **high-pass filtered image**, which contains only sharp edges and details.
3. **Effect**:
    - In the **top-right image** of the provided example, the high-pass filter enhances fine details like edges in the person's face.
    - The image looks darker and "sharp" because only high-frequency components remain.

How Low-Pass Filter Works:

1. **Purpose**: The low-pass filter smooths the image by **removing high-frequency components** like edges and details, retaining only general shapes and low-frequency regions.
2. **Process**:
    - A **Gaussian blur** is directly applied to the image.
    - The result is a blurred image with reduced sharpness.
3. **Effect**:
    - In the **bottom-right image** of the provided example, the low-pass filter smooths the details of the person's face.
    - The image appears blurry, with the overall structure visible but edges and details removed.

High-pass filter retains details and edges.

Low-pass filter smooths the image and retains general structures.

The combination creates a **hybrid image** where you perceive different images at different viewing distances.

5)

```
from PIL import Image

import numpy as np

def identify_flag(image_path):

"""

Determines if the input image is the flag of Indonesia or Poland

based on color distribution.

"""

# Open image and convert to RGB

img = Image.open(image_path).convert("RGB")

width, height = img.size

# Crop the image into top and bottom halves

top_half = img.crop((0, 0, width, height // 2))

bottom_half = img.crop((0, height // 2, width, height))

# Convert top and bottom halves to numpy arrays

top_colors = np.array(top_half)

bottom_colors = np.array(bottom_half)

# Calculate average color for top and bottom halves

top_avg_color = np.mean(top_colors, axis=(0, 1))

bottom_avg_color = np.mean(bottom_colors, axis=(0, 1))

# Define basic color thresholds for red and white

red_threshold = [150, 0, 0] # High red, low green and blue

white_threshold = [200, 200, 200] # High R, G, B values (white)
```

```
# Determine whether top/bottom is closer to red or white

def is_red(color):

return color[0] > red_threshold[0] and color[1] < 100 and color[2] < 100

def is_white(color):

return all(color[i] > white_threshold[i] for i in range(3))

# Identify flag based on color positions

if is_red(top_avg_color) and is_white(bottom_avg_color):

return "Flag of Indonesia"

elif is_white(top_avg_color) and is_red(bottom_avg_color):

return "Flag of Poland"

else:

return "Unknown Flag"

# Example usage

if __name__ == "__main__":

image_path = r"C:\Users\Vibha Narayan\OneDrive\Pictures\Screenshots\Screenshot 2024-12-18 215738.png"

result = identify_flag(image_path)

print("Result:", result)
```

Explanation:

1. Importing Libraries

- **Pillow (PIL)**: Used for image processing, such as opening, cropping, and converting images.
- **NumPy**: Used for efficient numerical computations on pixel data, like calculating average colors.

2. `identify_flag` Function

This function determines whether the input image is the flag of Indonesia or Poland.

3. Image Loading and Preprocessing

The image is opened and converted to RGB format to ensure color values are in the Red-Green-Blue format.

`img.size` retrieves the image dimensions (width and height).

4. Splitting the Image

- The image is split into two **equal horizontal halves**:
    - **Top half**: Starts from the top (y=0) and ends at the middle (y=height//2).
    - **Bottom half**: Starts from the middle (y=height//2) and ends at the bottom (y=height).

5. Converting Image Data to Arrays

- The cropped top and bottom halves are converted to NumPy arrays.
- This allows for efficient pixel-wise operations (e.g., calculating the average color).

6. Calculating Average Colors

- **`np.mean`** calculates the average color (R, G, B) for each half.
- `axis=(0, 1)` ensures averaging over both dimensions (width and height).

7. Defining Color Thresholds

Red threshold: A color is considered red if:

- Red component (R) is high (> 150).
- Green and Blue components are low (< 100).

White threshold: A color is considered white if all RGB values are high (> 200).

8. Helper Functions to Check Colors:

`is_red`: Checks if a color matches the red threshold.

`is_white`: Checks if a color matches the white threshold.

9. Identifying the Flag

The program checks the average colors of the **top** and **bottom** halves:

- **Indonesia**: Top half is red, and bottom half is white.
- **Poland**: Top half is white, and bottom half is red.

If neither condition is met, the flag is classified as "Unknown."

The image is opened and split into two horizontal halves.

The average color of each half is calculated.

Helper functions (`is_red` and `is_white`) determine if the average color matches the thresholds for red or white.

Based on the color positions:

- If the top is red and the bottom is white → **Indonesia**.
- If the top is white and the bottom is red → **Poland**.

If no match, it outputs "Unknown Flag."