Image Processing Internship Report

Author: Sneha Dadhich
Topic: Learning OpenCV for Image Processing
Duration: 5 May 2025 to 28 May 2025
Tools Used: Python, OpenCV, NumPy

1. What is Image Processing?

Definition:
Image processing is a method to perform operations on an image to enhance it or extract useful information. It involves manipulating pixel values of images using mathematical and logical operations.

2. Applications of Image Processing

- **Medical Imaging:** CT scans, MRI enhancements
- **Object Detection:** Surveillance, autonomous vehicles
- **Face Recognition:** Authentication systems
- **Optical Character Recognition (OCR):** Digitizing text from images
- **Industrial Automation:** Detecting defects, sorting objects
- **Augmented Reality & Gaming:** Real-time background replacement, motion capture

3. Introduction to OpenCV

OpenCV (Open Source Computer Vision Library)

- An open-source computer vision and machine learning software library.
- It contains more than 2500 optimized algorithms.
- Supports real-time image and video processing.
- Compatible with multiple languages: Python, C++, Java, etc.

Key Features:

- Image manipulation
- Video capture and analysis
- Machine learning tools (object recognition, face detection)

- Real-time performance

4. OpenCV in Python

a. Installation

- pip install opencv-python

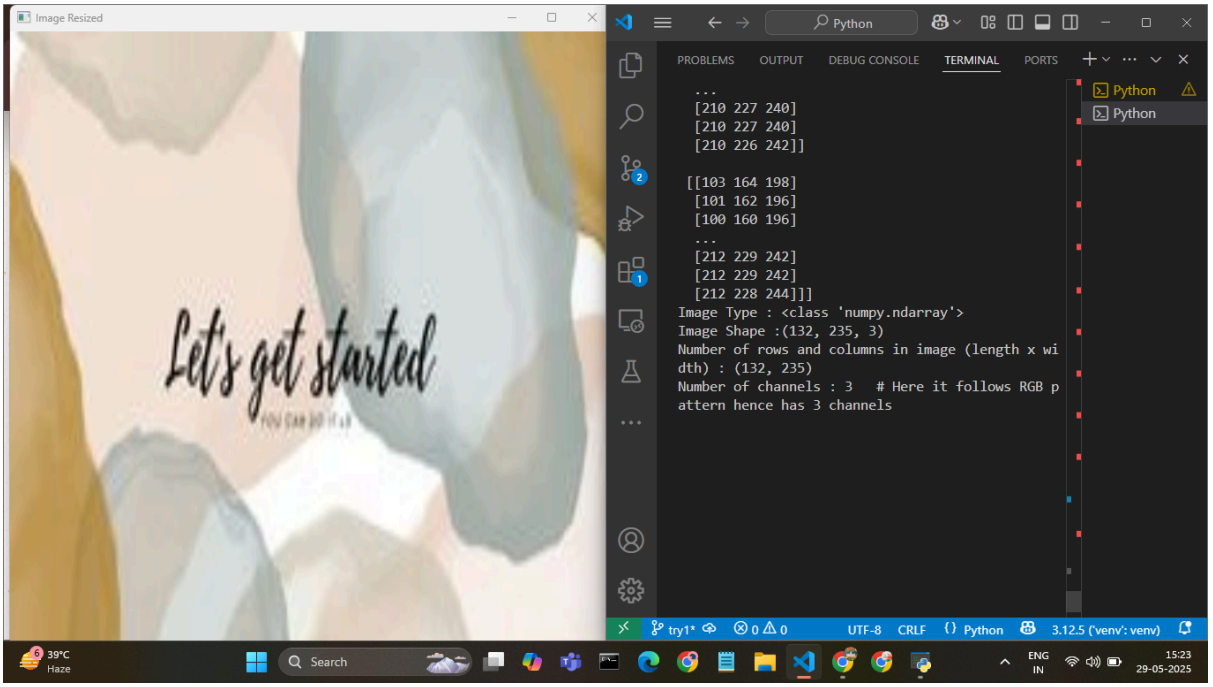
b. Basic Image Operations

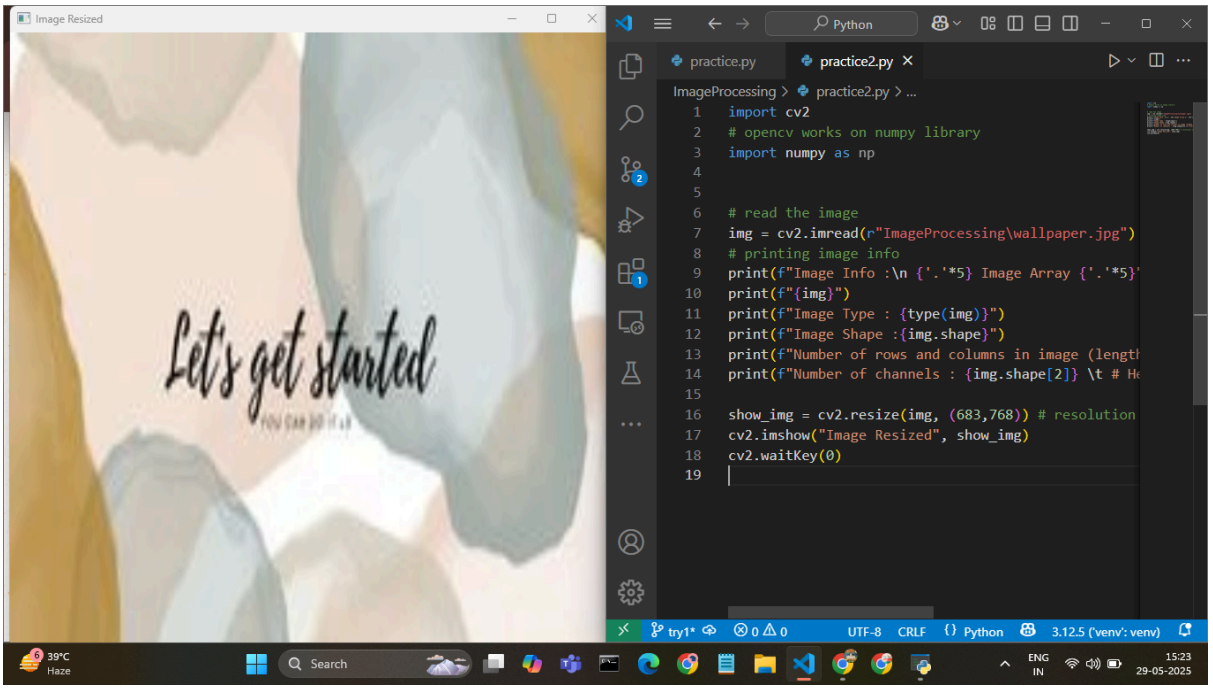
Reading and Displaying Images:

- import cv2
- img = cv2.imread("image.jpg")
- cv2.imshow("Window", img)
- cv2.waitKey(0)
- cv2.destroyAllWindows()

Image Properties:

- print(f"Image Info :\n {"."*5} Image Array {"."*5}")
- print(f"{img}")
- print(f"Image Type : {type(img)}")
- print(f"Image Shape :{img.shape}")
- print(f"Number of rows and columns in image (length x width) : {img.shape[:2]}")
- print(f"Number of channels : {img.shape[2]} \t # Here it follows RGB pattern hence has 3 channels")



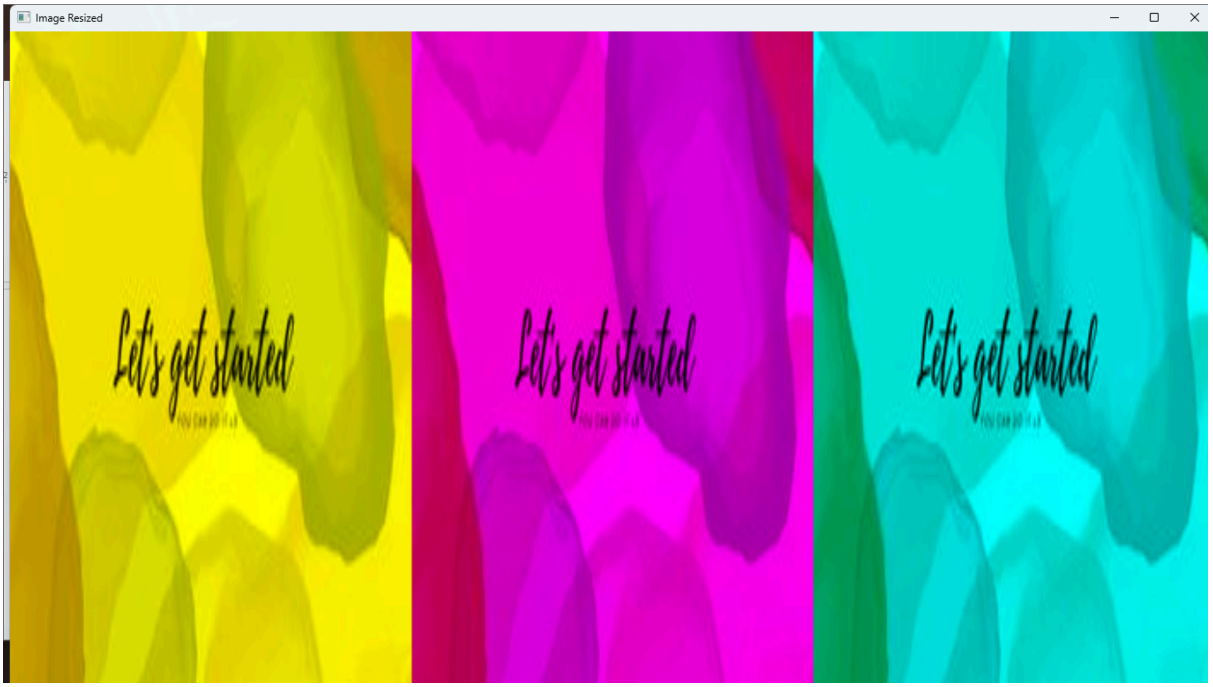


c. Image Manipulation

Color Channels (BGR):

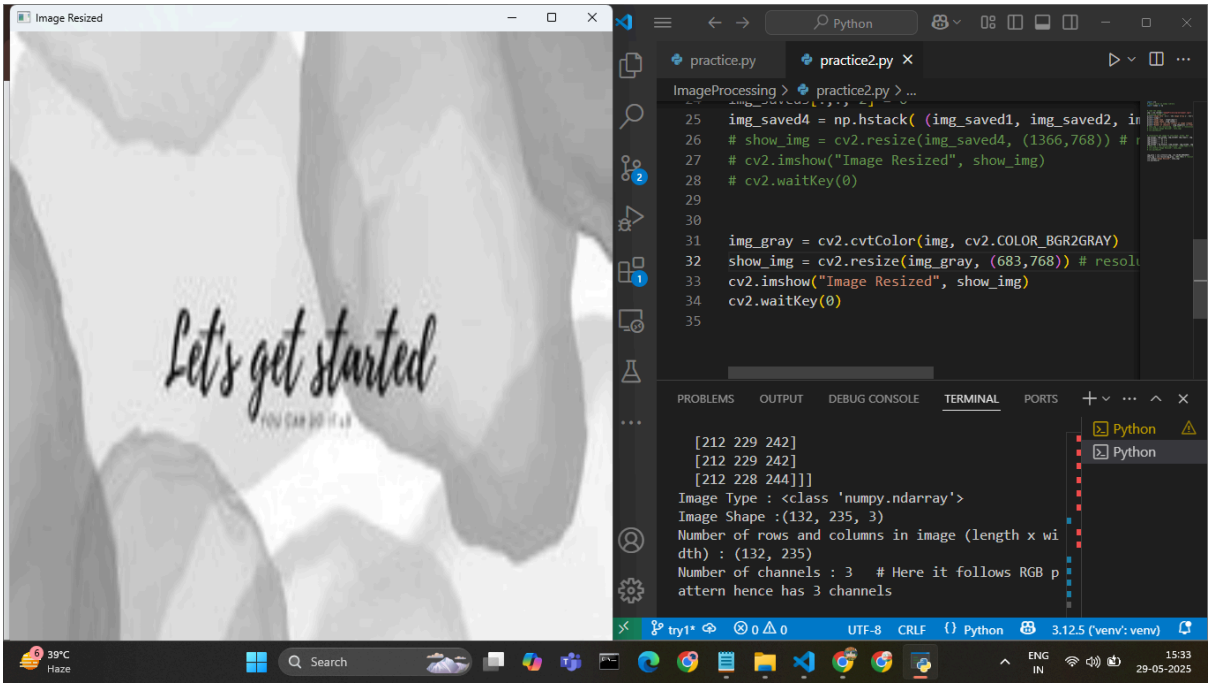
- # converting the image to only Blue, Green, Red
- img_saved1, img_saved2, img_saved3= img.copy(), img.copy(), img.copy()
- img_saved1[:,:, 0] = 0
- img_saved2[:,:, 1] = 0
- img_saved3[:,:, 2] = 0
- img_saved4 = np.hstack((img_saved1, img_saved2, img_saved3))

- show_img = cv2.resize(img_saved4, (683,768)) # resolution of laptop - 1366x768
- cv2.imshow("Image Resized", show_img)
- cv2.waitKey(0)



Grayscale Conversion:

- img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
- show_img = cv2.resize(img_gray, (683,768)) # resolution of laptop - 1366x768
- cv2.imshow("Image Resized", show_img)
- cv2.waitKey(0)

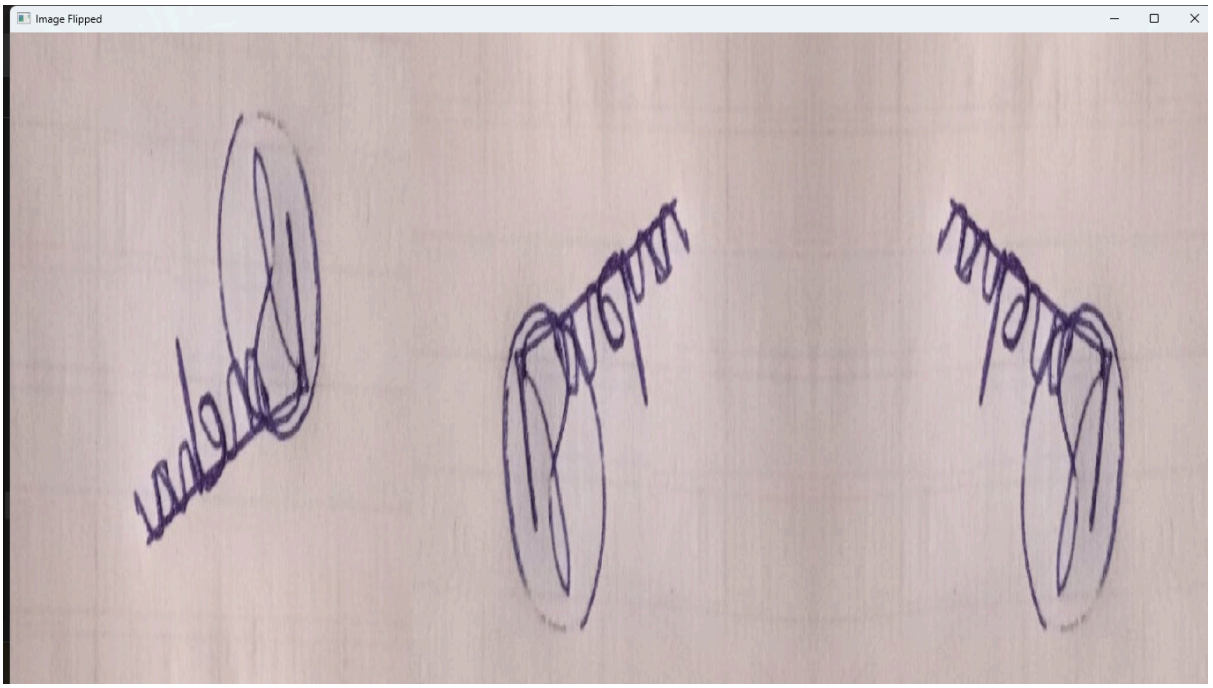


Resize:

- show_img = cv2.resize(img_gray, (683,768)) # resolution of laptop - 1366x768
- cv2.imshow("Image Resized", show_img)
- cv2.waitKey(0)

Flip:

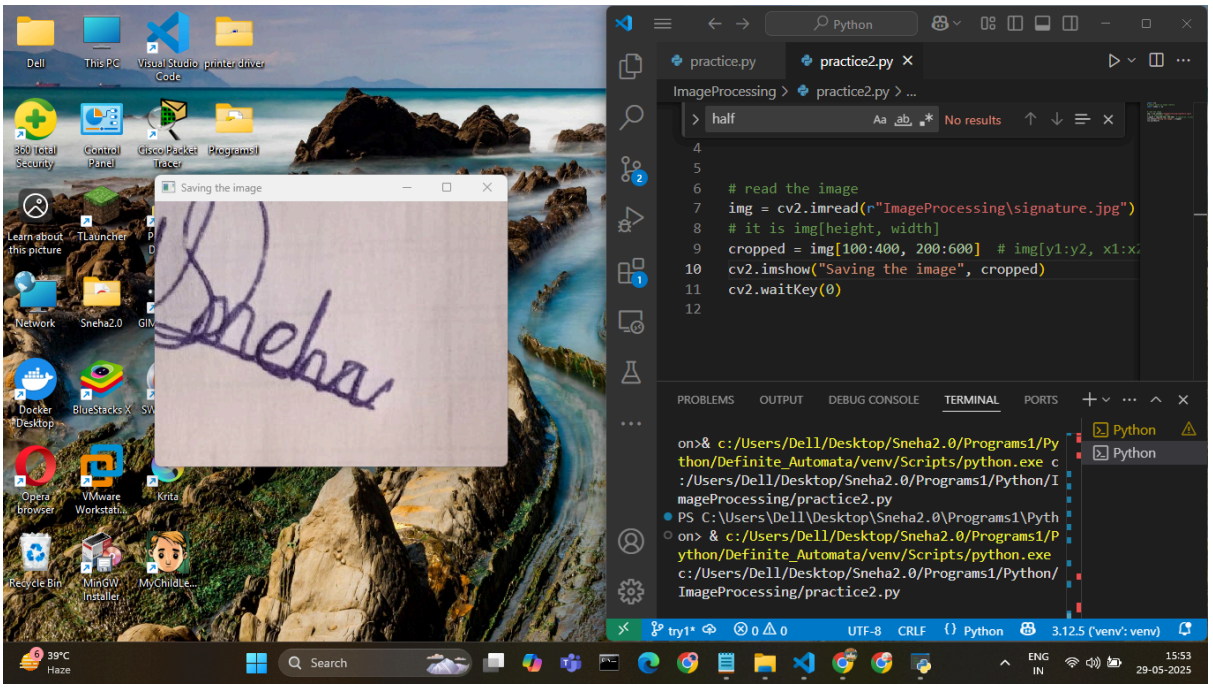
- # flipping the image
- # 0 means 180 degree, 1 means 360 degree
- # 0 means flip by Vertical Axis and 1 means flip by Horizontal Axis
- # -1 means flip by both vertical and horizontal axis
- img_flip1 = cv2.flip(img, 1)
- img_flip2 = cv2.flip(img, 0)
- img_flip3 = cv2.flip(img, -1)
-
- img_flip = np.hstack((img_flip1, img_flip2, img_flip3))
- show_img = cv2.resize(img_flip, (1366,768)) # resolution of laptop - 1366x768
- cv2.imshow("Image Flipped", show_img)
- cv2.waitKey(0)



Crop:

- # read the image
- img = cv2.imread(r"ImageProcessing\signature.jpg")
- # it is img[height, width]
- cropped = img[100:400, 200:600] # img[y1:y2, x1:x2]# cv2.imwrite(r".\ImageProcessing\Cropped_image11.jpg", save_image)

- cv2.imshow("Saving the image", cropped)
- cv2.waitKey(0)



d. Drawing Shapes and Text

- # creating your own image
- # creating an black window
- img_created = img = np.zeros((512, 512, 3))

- # creating a rectangle in the created black window
- cv2.rectangle(img_created, pt1=(100,100), pt2=(300,300), color=(255,0,0), thickness=3)

- # creating a rectangle in the created black window with color filled in it
- cv2.rectangle(img_created, pt1=(150,150), pt2=(300,300), color=(255,0,0), thickness=-100)

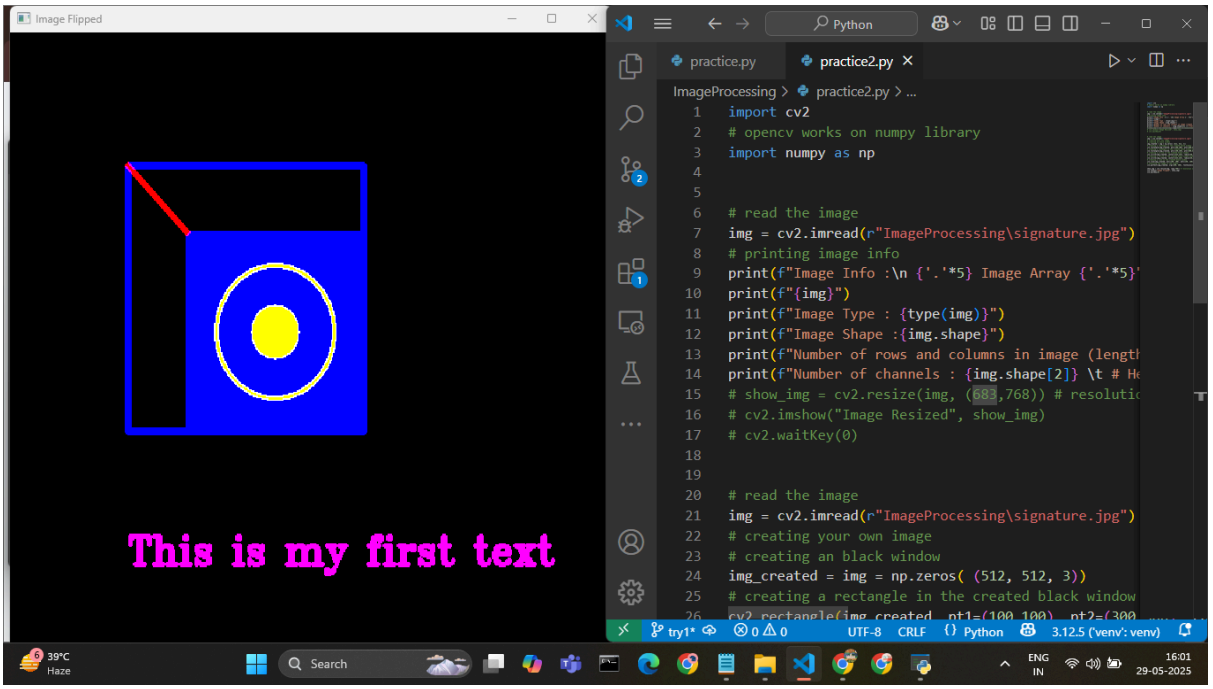
- # creating a circle in the created black window
- cv2.circle(img_created, center=(225,225), radius=50, color=(0,255,255), thickness=2)

- # creating a circle in the created black window with color filled in it
- cv2.circle(img_created, center=(225,225), radius=20, color=(0,255,255), thickness=-1)

- # creating a line in the created black window

- cv2.line(img_created, pt1=(100, 100), pt2=(150, 150), color=(0,0,255), thickness=3)

- # inserting the image in the window
- cv2.putText(img_created, org=(100, 400), fontScale=1, color=(255,0,255), thickness = 2, lineType=cv2.LINE_AA, text="This is my first text", fontFace=cv2.FONT_HERSHEY_TRIPLEX)
- cv2.imshow("Created Image1", img_created)
- cv2.waitKey(0)



e. Handling Mouse Events

Callback Example:

- def draw(event, x, y, flags, param):
- if event == cv2.EVENT_LBUTTONDOWN:
- print("Mouse Clicked at:", x, y)

- cv2.namedWindow("Window")
- cv2.setMouseCallback("Window", draw)

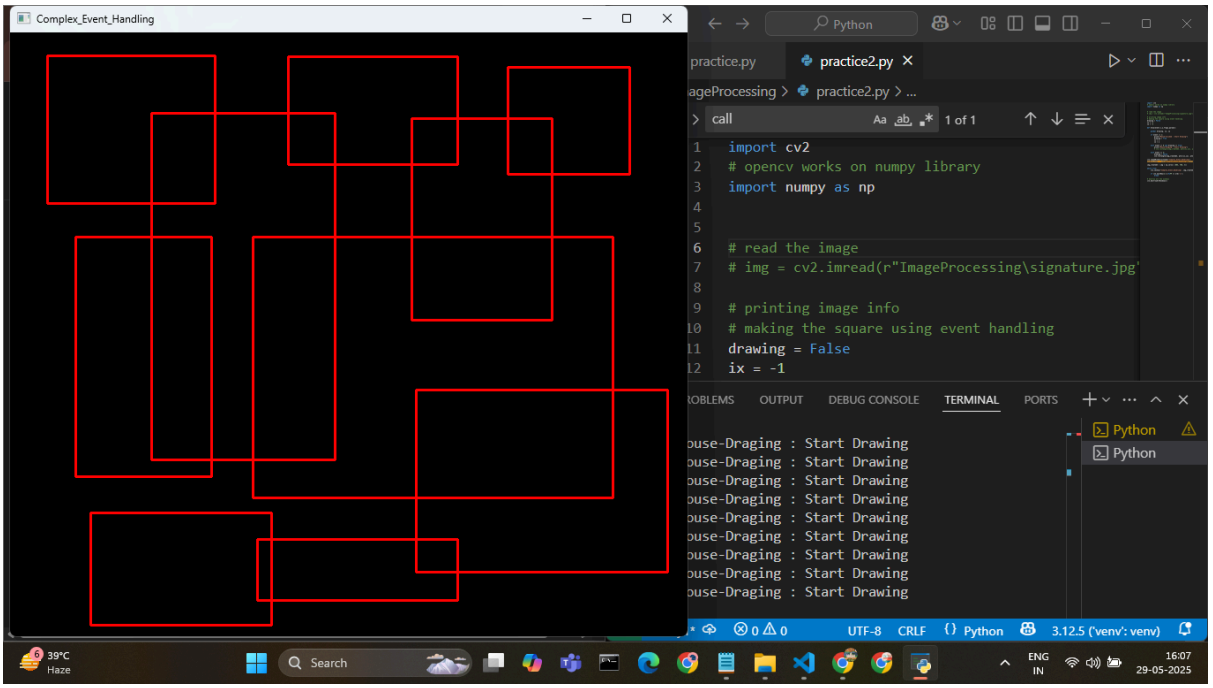
Live Drawing with Mouse:

- Used `EVENT_LBUTTONDOWN`, `EVENT_MOUSEMOVE`, `EVENT_LBUTTONUP`

Draw rectangle interactively

```
import cv2
# opencv works on numpy library
import numpy as np

# read the image
# img = cv2.imread(r"ImageProcessing\signature.jpg")

# printing image info
# making the square using event handling
drawing = False
ix = -1
iy = -1

def draw(event,x,y,flags,params):
    global drawing, ix, iy

    if event == 1:
        print("Mouse-Clicked : Start Drawing")
        drawing = True
        ix = x
        iy = y

    elif event == 0 and drawing == True:
        print("Mouse-Draging : Start Drawing")
        # cv2.rectangle(img_created3, pt1=(ix,iy), pt2=(x,y), color=(0,0,255),
thickness=2)

    elif event == 4:
        drawing = False
        cv2.rectangle(img_created3, pt1=(ix,iy), pt2=(x,y), color=(0,0,255), thickness=2)

cv2.namedWindow(winname="Complex_Event_Handling")
cv2.setMouseCallback("Complex_Event_Handling", draw)

img_created3 = img = np.zeros( (683, 768, 3))

while True:
    cv2.imshow("Complex_Event_Handling", img_created3)

    if cv2.waitKey(1) & 0xFF == ord('x'):
        break
```

# destroy all the windows
cv2.destroyAllWindows()



Crop selected region and save

- import cv2
- # opencv works on numpy library
- import numpy as np

- img = cv2.imread("ImageProcessing\signature.jpg")
- # making the cropping tool using event handling
- drawing = False
- ix = -1
- iy = -1

- def draw(event,x,y,flags,params):
- global drawing, ix, iy
-
- if event == 1:
- print("Mouse-Clicked : Start Drawing")
- drawing = True
- ix = x
- iy = y
-
- elif event == 0 and drawing == True:
- print("Mouse-Draging : Start Drawing")

❖ # cv2.rectangle(img_created3, pt1=(ix,iy), pt2=(x,y), color=(0,0,255), thickness=2)

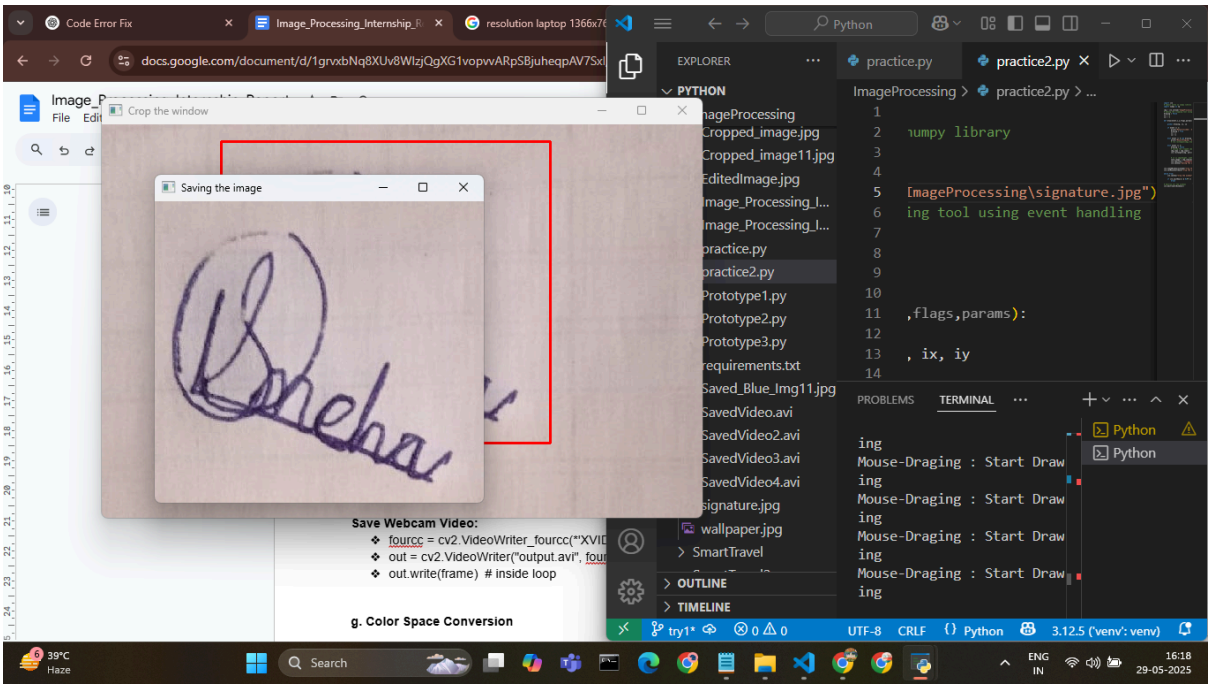
❖ elif event == 4:
❖ drawing = False
❖ # saving the cropped image
❖ img_copy = img.copy()
❖ cv2.rectangle(img, pt1=(ix,iy), pt2=(x,y), color=(0,0,255), thickness=2)

❖ # it is img[height, width]
❖ save_image = img_copy[min(iy-1, y-1):max(iy-1, y-1), min(ix-1, x-1):max(ix-1, x-1)]
❖ cv2.imwrite(r".\ImageProcessing\Cropped_image.jpg", save_image)
❖ cv2.imshow("Saving the image", save_image)

❖ cv2.namedWindow(winname="Crop the window")
❖ cv2.setMouseCallback("Crop the window", draw)

❖ while True:
❖ cv2.imshow("Crop the window", img)
❖
❖ if cv2.waitKey(1) & 0xFF == ord('x'):
❖ break

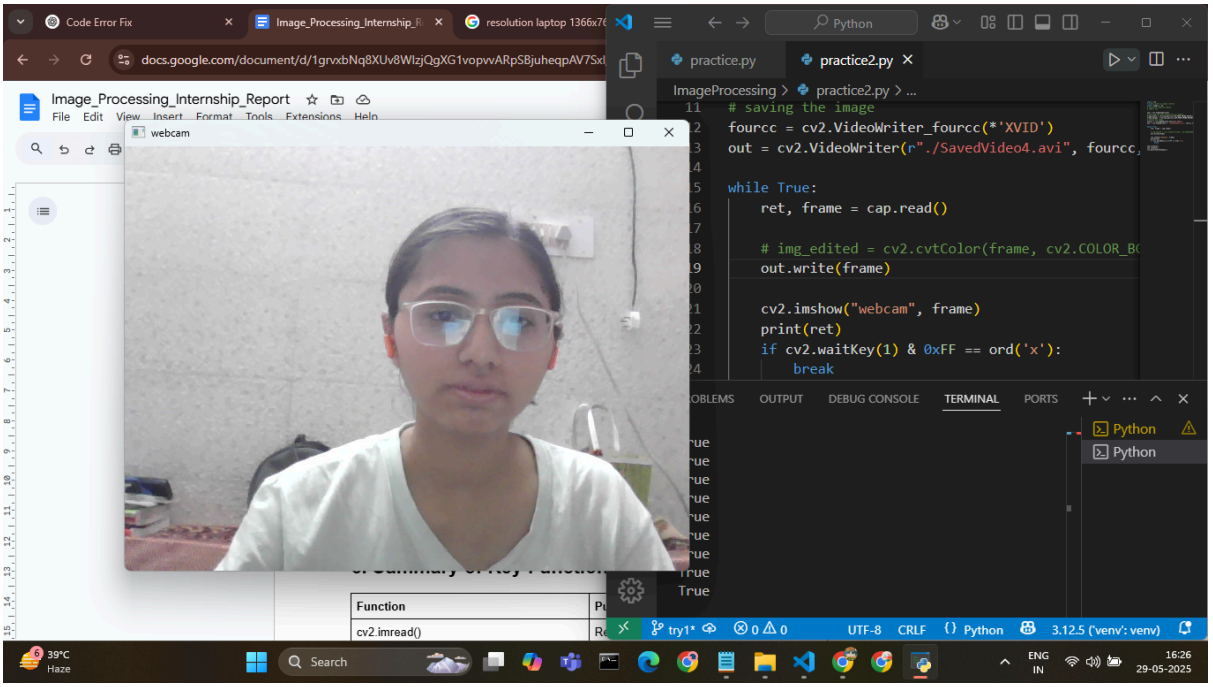
❖ # destroy all the windows
❖ cv2.destroyAllWindows()



f. Working with Video

Capture from Webcam and Saving it:

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

# fetching the exact dimensions of webcam
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# saving the image
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter(r"./SavedVideo4.avi", fourcc,20.00, (frame_width,
frame_height))

while True:
    ret, frame = cap.read()

    # img_edited = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    out.write(frame)

    cv2.imshow("webcam", frame)
    print(ret)
    if cv2.waitKey(1) & 0xFF == ord('x'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```



6. Summary of Key Functions

Function	Purpose
cv2.imread()	Read an image
cv2.imshow()	Display an image
cv2.waitkey()	Wait for keypress
cv2.destroyAllWindows()	Close all windows
cv2.cvtColor()	Convert color spaces
cv2.resize()	Resize image
cv2.flip()	Flip image
cv2.rectangle()	Draw rectangle
cv2.circle()	Draw circle
cv2.line()	Draw line
cv2.putText()	Draw text
cv2.VideoCapture()	Capture video stream
cv2.VideoWriter()	Save video to file
cv2.setMouseCallback()	Set mouse event handler

cv2.imwrite()	Save image to disk

7. Image Enhancement

Image Enhancement in Python refers to the process of improving the visual appearance of an image or making it more suitable for analysis by modifying its features such as contrast, brightness, sharpness, or noise.

Python provides several libraries for image enhancement, especially:

- **OpenCV** (`cv2`)
- **PIL / Pillow**
- **scikit-image**

Common Image Enhancement Techniques

a. Brightness Adjustment

Brightness adjustment is the process of increasing or decreasing the intensity of all pixels in an image. In simple terms, it makes the image look **brighter** or **darker** by modifying its **pixel values**.

Each pixel in an image has an intensity value (ranging from 0 to 255 for 8-bit images). Brightness is adjusted by **adding or subtracting a constant value** to all pixels:

- **Increase Brightness:** Add a positive value to pixel intensities
- **Decrease Brightness:** Subtract a value from pixel intensities

Command: cv2.convertScaleAbs(src, alpha, beta)

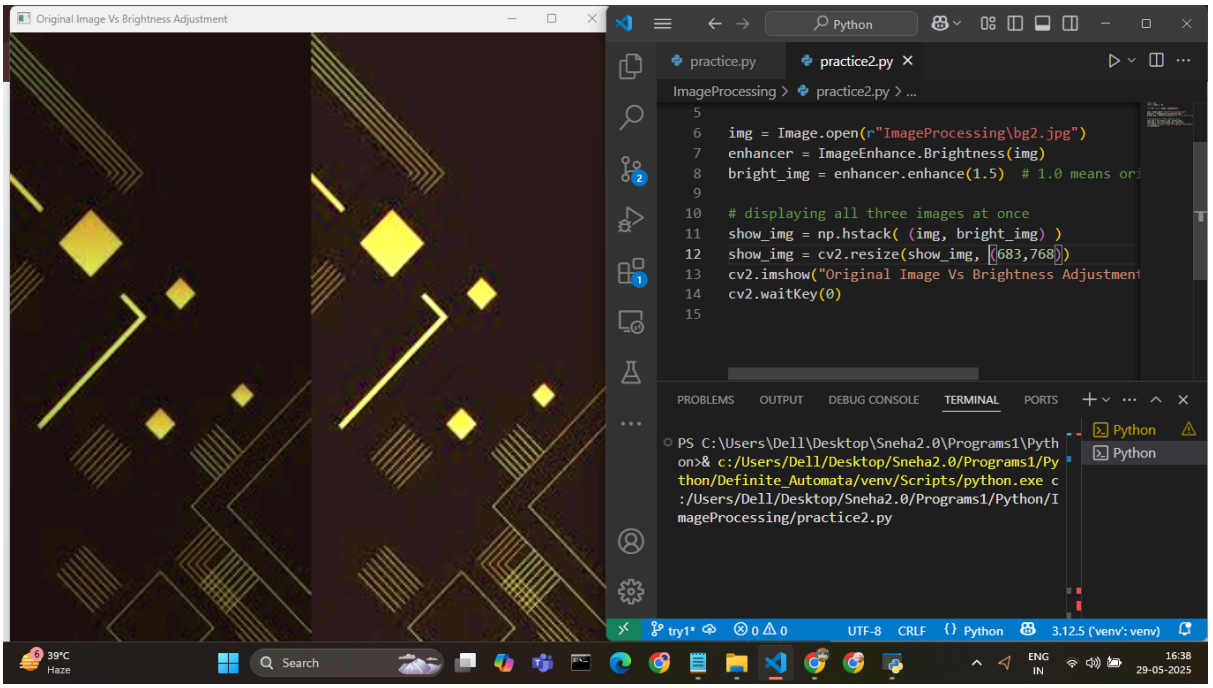
- `alpha`: Contrast control (default is 1.0)
- `beta`: Brightness control (positive to brighten, negative to darken)

Formula used: new_pixel = src_pixel * alpha + beta

Increases or decreases the light intensity of the image.

```
import cv2
import numpy as np
from PIL import Image, ImageEnhance

img = Image.open(r"ImageProcessing\bg2.jpg")
enhancer = ImageEnhance.Brightness(img)
```

- bright_img = enhancer.enhance(1.5) # 1.0 means original, >1 brighter, <1 darker
-
- # displaying all three images at once
- show_img = np.hstack((img, bright_img))
- show_img = cv2.resize(show_img, (683,768))
- cv2.imshow("Original Image Vs Brightness Adjustment", show_img)
- cv2.waitKey(0)



b. Contrast Adjustment

Contrast adjustment enhances the difference between the **lightest** and **darkest** parts of an image. It stretches or compresses the **range of pixel intensity values**, making features more or less distinguishable.

- **Increasing contrast**: Makes darks darker and brights brighter.
- **Decreasing contrast**: Makes the image appear more gray and flat

Command:

- enhancer = ImageEnhance.Contrast(img)
- contrast_img = enhancer.enhance(2.0)

Formula Used: new_pixel = alpha * pixel + beta

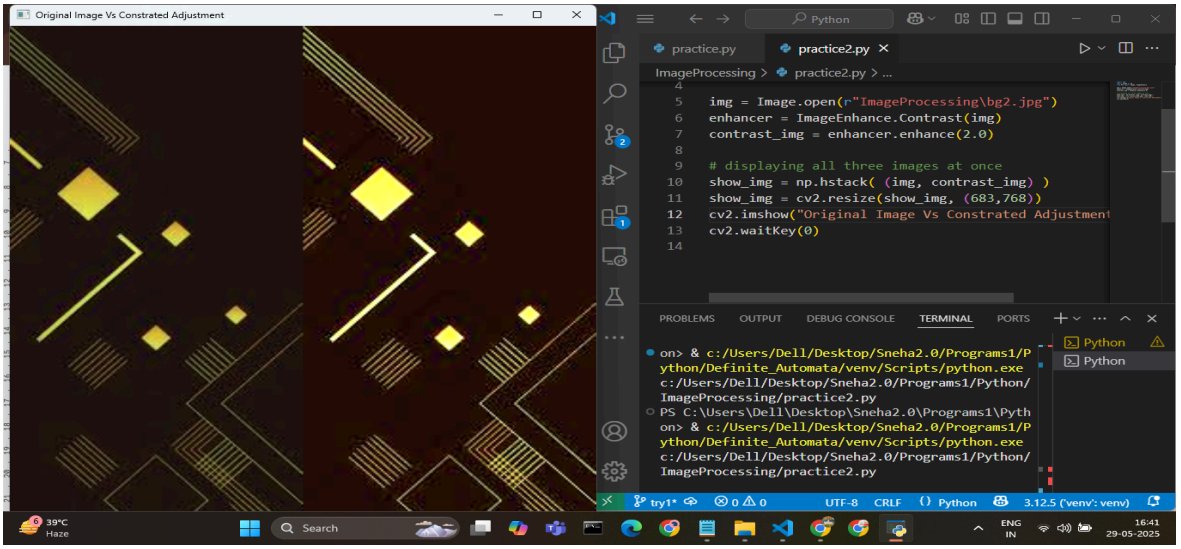
- `alpha` → **Contrast factor** (greater than 1 to increase, between 0–1 to decrease)
- `beta` → **Brightness factor** (optional)

Improves the distinction between light and dark areas.

- import cv2
- import numpy as np
- from PIL import Image, ImageEnhance

- img = Image.open(r"ImageProcessing\bg2.jpg")
- enhancer = ImageEnhance.Contrast(img)
- contrast_img = enhancer.enhance(2.0)

- # displaying all three images at once
- show_img = np.hstack((img, contrast_img))
- show_img = cv2.resize(show_img, (683,768))
- cv2.imshow("Original Image Vs Constrated Adjustment", show_img)
- cv2.waitKey(0)



c. Sharpness Adjustment

Sharpness adjustment enhances the clarity of edges and fine details in an image. Increasing sharpness makes edges more distinct, while decreasing sharpness (blurring) softens the image.

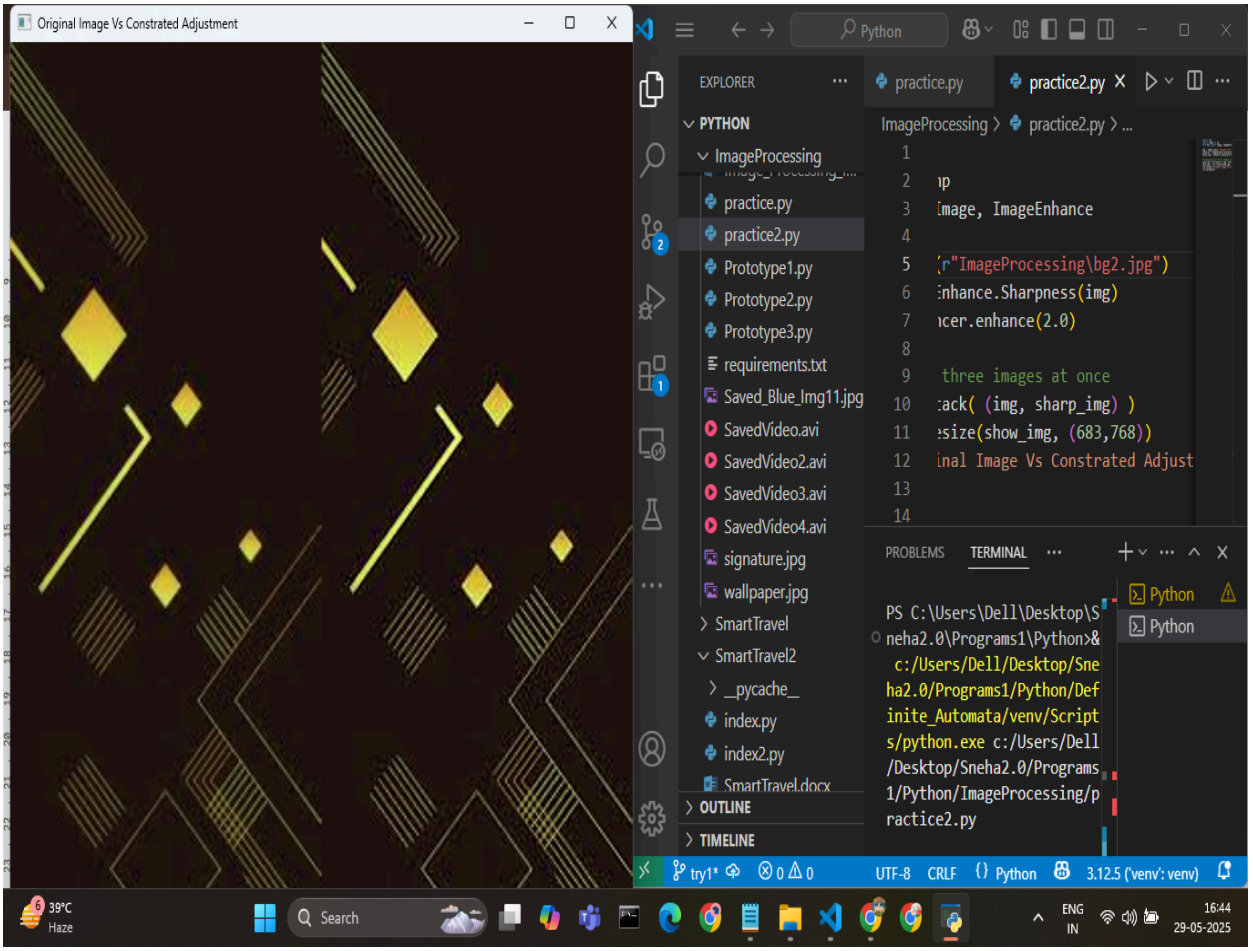
It is commonly achieved by applying **convolution filters** (kernels) that enhance edges and transitions in pixel intensity.

Highlights edges and fine details.

- img = Image.open(r"ImageProcessing\bg2.jpg")
- enhancer = ImageEnhance.Sharpness(img)

- sharp_img = enhancer.enhance(2.0)

- # displaying all three images at once
- show_img = np.hstack((img, sharp_img))
- show_img = cv2.resize(show_img, (683,768))
- cv2.imshow("Original Image Vs Constrated Adjustment", show_img)
- cv2.waitKey(0)



d. Color Enhancement

Color enhancement is the process of improving the **vividness**, **saturation**, or **balance** of colors in an image to make it more visually appealing or informative. This is often used in photography, medical imaging, or satellite imagery.

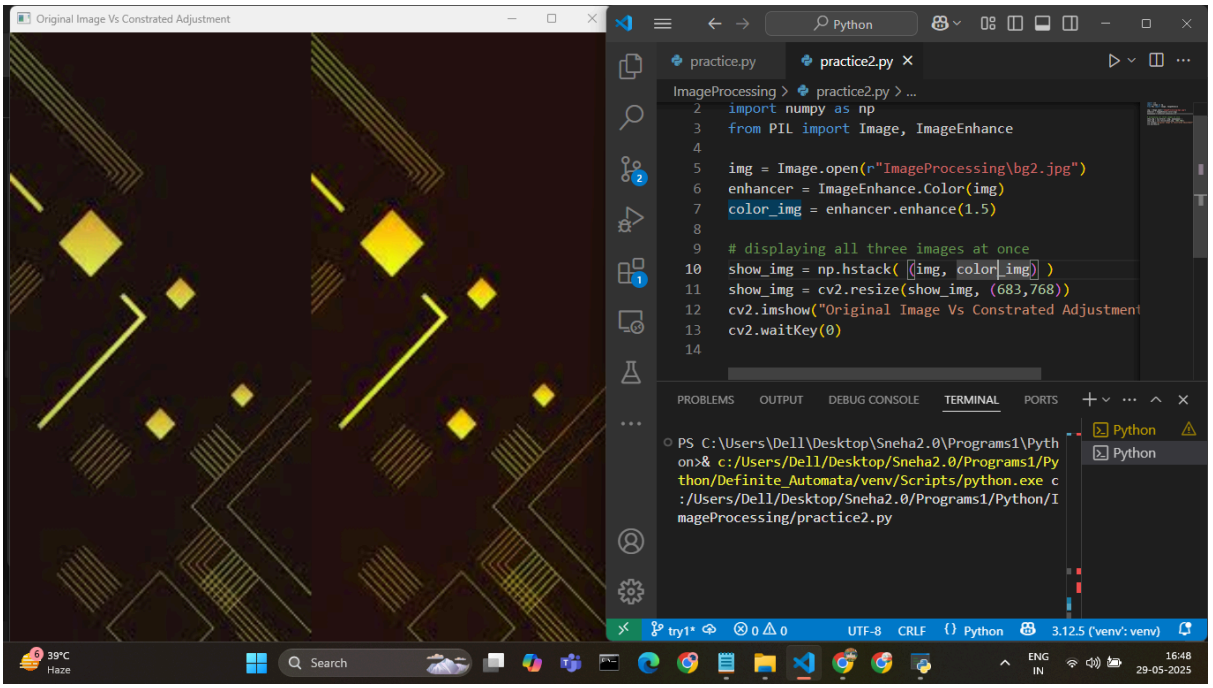
It can involve:

- Increasing **color saturation**
- Adjusting individual **RGB channels**
- Enhancing **warmth** or **coolness**
- Balancing **white levels**

Changes the intensity of colors.

- img = Image.open(r"ImageProcessing\bg2.jpg")
- enhancer = ImageEnhance.Color(img)
- color_img = enhancer.enhance(1.5)

- # displaying all three images at once
- show_img = np.hstack((img, color_img))
- show_img = cv2.resize(show_img, (683,768))
- cv2.imshow("Original Image Vs Constrated Adjustment", show_img)
- cv2.waitKey(0)



e. Histogram Equalization (with OpenCV)

Histogram Equalization is a technique to **improve the contrast** of an image by **spreading out** the most frequent intensity values. This makes **dark areas lighter** and **bright areas more detailed**, which is especially useful in low-contrast images.

Used to enhance contrast, especially in grayscale images.

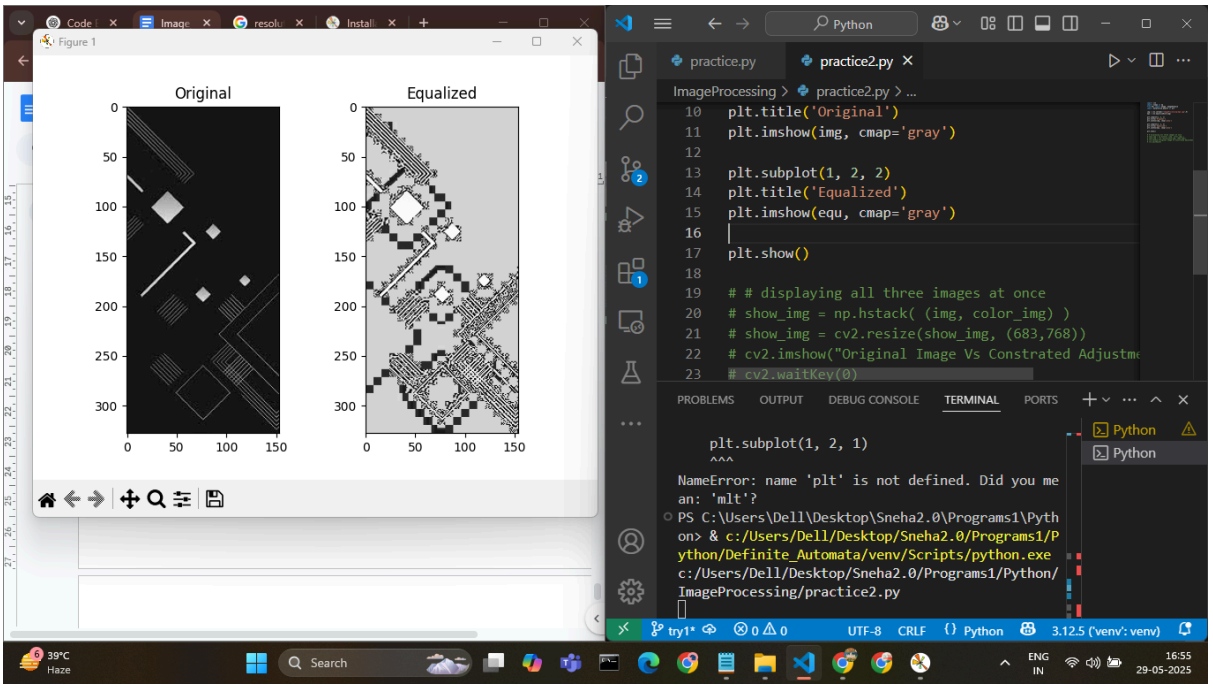
- import cv2
- import numpy as np
- from PIL import Image, ImageEnhance
- import matplotlib.pyplot as plt

- img = cv2.imread(r"ImageProcessing\bg2.jpg",0)
- equ = cv2.equalizeHist(img)

- plt.subplot(1, 2, 1)
- plt.title('Original')
- plt.imshow(img, cmap='gray')

- plt.subplot(1, 2, 2)
- plt.title('Equalized')
- plt.imshow(equ, cmap='gray')

- plt.show()



f. Denoising

Denoising is the process of **removing noise** from an image while preserving important details like edges, textures, and structures.

Noise is unwanted random variation in pixel intensity, often caused by:

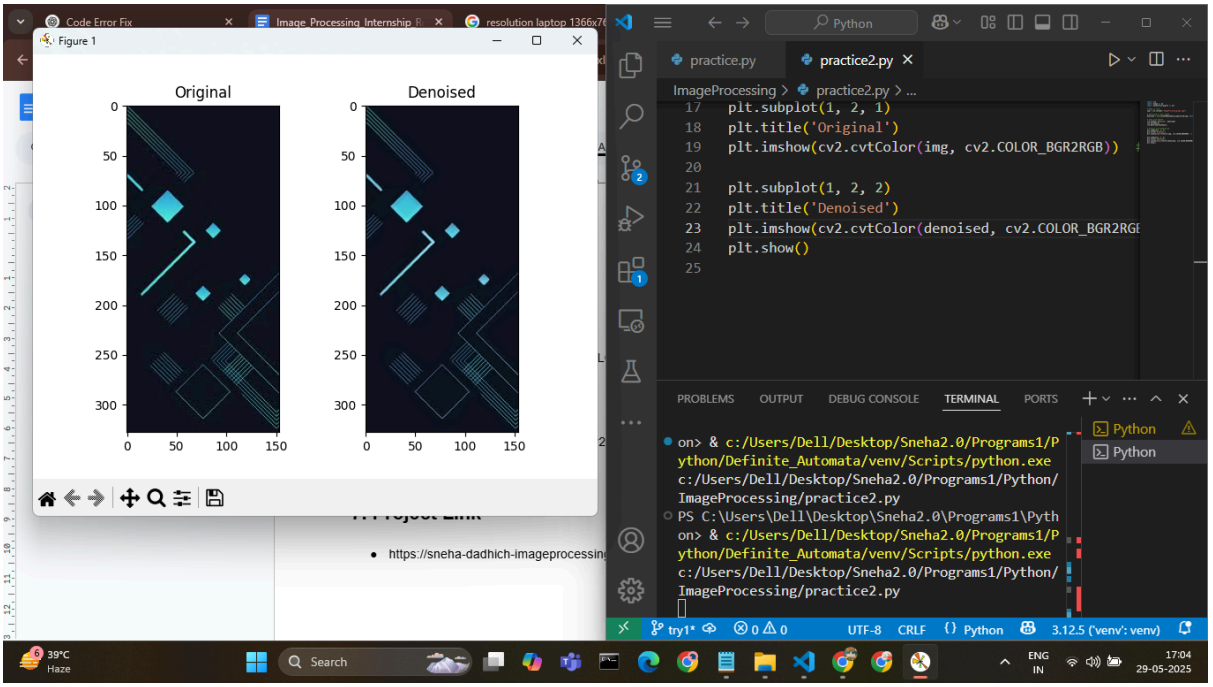
- Poor lighting
- Low-quality sensors
- Compression artifacts

Applications:

- Smooth out noise (random speckles, grain)
- Preserve edges and sharp details
- Improve visual quality or preprocessing for further tasks

Removes noise to smooth the image.

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

# Read in color
img = cv2.imread(r"ImageProcessing\bg2.jpg")

# Denoising a color image
denoised = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)

# Show with OpenCV
cv2.imshow('Denoised', denoised)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Display with matplotlib
plt.subplot(1, 2, 1)
plt.title('Original')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # convert BGR to RGB for matplotlib

plt.subplot(1, 2, 2)
plt.title('Denoised')
plt.imshow(cv2.cvtColor(denoised, cv2.COLOR_BGR2RGB))
plt.show()
```



7. Independent Project

This Streamlit app is an interactive image processing tool that enables users to upload and edit images with ease. It supports common transformations and provides instant previews and downloads.

Key Features:

- Image Upload: Supports PNG, JPG, SVG formats.
- Live Preview: Displays the edited image instantly after any operation.
- Rotate: Rotate images 90° left or right using OpenCV.
- Color Effects:
  - Isolate Red, Green, or Blue channels
  - Convert to Grayscale
  - Convert to HSV color space
- Resize Options:
  - A4 (3508×2480)
  - Website Banner (3508×1280)
  - Portrait (1350×1080)
  - Landscape (1920×1080)
- Download Button: Save the edited image as a PNG.
- Session Handling: Uses `st.session_state` to retain edits across actions.

- Clean UI: Split view layout with operation buttons in the sidebar and live image on the main screen.

8. Conclusion -

During this initial phase of my internship, I explored the fundamentals of image processing using Python and OpenCV. I gained hands-on experience with basic image operations, enhancement techniques, and video handling. I also built an interactive Streamlit-based tool for image manipulation, which helped reinforce my understanding of real-time applications. This assignment has laid a strong foundation for more advanced topics such as segmentation, edge detection, and deep learning-based image analysis, which I look forward to exploring in the upcoming phases of the internship.