# The Superior University Lahore

## Faculty of Computer Science & Information Technology


**Name:** Alishba Haroon

**Roll No:** BSAI-116-4C


**Date:** 27 Feb 2025

**Subject:** PAI LAB

## **Libraries** :

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt
import imutils
```

- Numpy is a powerful library for numerical computing. It supports large, multi-dimensional arrays and matrices and mathematical functions to operate on them.
- cv2 (OpenCV) is an open-source library for computer vision and image processing. It allows reading, writing, processing, and analyzing images and videos.
- matplotlib.pyplot is used to display images, graphs, and plots in a structured way.
- imutils is a helper library that simplifies many common OpenCV operations, such as resizing, rotating, and displaying images.

->Read Images in OpenCV Python

```python
image = cv2.imread(r'image path')
```

- cv2.imread() is a function from OpenCV (cv2) used to read an image from a specified file path.

## ->Display an Image in Python Using OpenCV

```
image = cv2.imread(r'image path')
cv2.imshow('Image Window', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- cv2.imread() loads the image.
- cv2.imshow() displays it.
- cv2.waitKey(0) keeps the window open until a key is pressed.
- cv2.destroyAllWindows() closes all OpenCV windows.

## ->Read Image in OpenCV Library

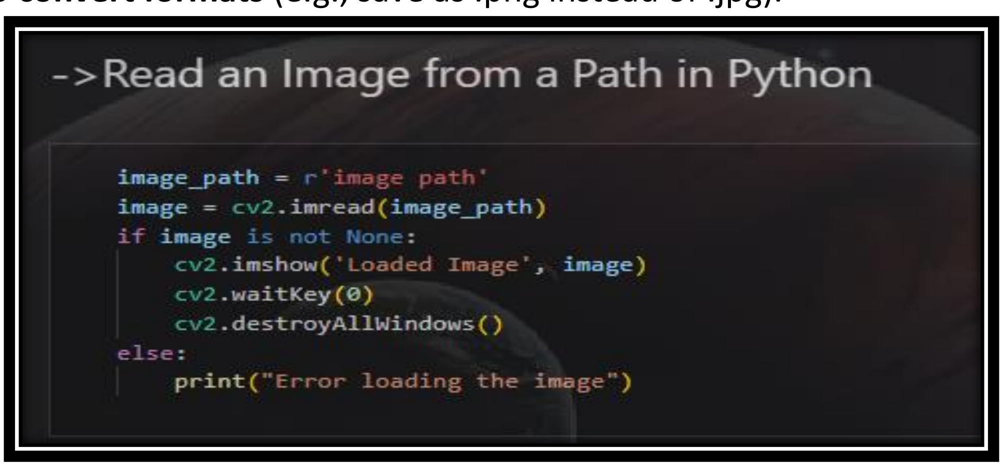
```
gray_image = cv2.imread(r'image path', cv2.IMREAD_GRAYSCALE)
```

- cv2.imread(r'image path', cv2.IMREAD_GRAYSCALE):
- Loads the image in **grayscale mode** (black & white).
- Reduces memory usage and simplifies processing.

```
img = cv2.imread(r"image path")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show(block=True)
```

- **cv2.imread(r"image path")**
- Reads the image using OpenCV (loads it in BGR format).
- **cv2.cvtColor(img, cv2.COLOR_BGR2RGB)**
- Converts the image from **BGR to RGB** (since OpenCV loads images in BGR, while Matplotlib expects RGB).
- **plt.imshow()**
- Displays the RGB image using Matplotlib.
- **plt.show(block=True)**
- Ensures the plot remains open until manually closed.


- **cv2.imread(r"image path")**
- Reads the image from the specified path.
- **cv2.imwrite(r"image path", image)**
- Saves the image to the specified path.
- To **convert formats** (e.g., save as .png instead of .jpg).



- image_path = r'image path'
- Specifies the file path of the image.
- image = cv2.imread(image_path)
- Reads the image from the given path.
- if image is not None:
- Checks if the image was successfully loaded.
- cv2.imshow('Loaded Image', image)
- Displays the image in a window.
- cv2.waitKey(0)
- Waits indefinitely for a key press.
- cv2.destroyAllWindows()
- Closes all OpenCV windows.
- else:
- Executes if the image failed to load.
- print("Error loading the image")
- Prints an error message if the image is not found

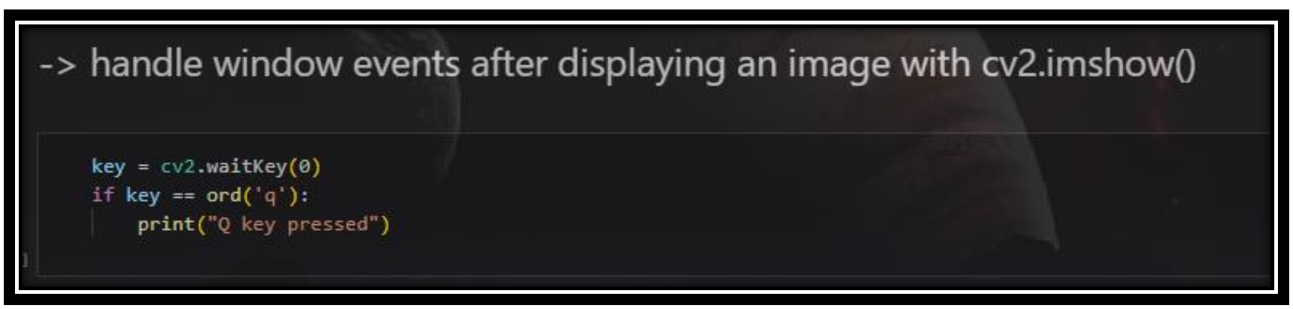
->read and show image in grayscale mode

```
path = r'image path'
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- path = r'image path'
- Specifies the file path of the image.
- img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
- Reads the image in grayscale mode.
- cv2.imshow('image', img)
- Displays the grayscale image in a window.
- cv2.waitKey(0)

- Waits indefinitely for a key press.
- cv2.destroyAllWindows()
- Closes all OpenCV windows.


- The .shape attribute of an image (NumPy array) returns its dimensions.
- The output (3072, 4608) indicates:
- **3072**: Height (number of rows)
- **4608**: Width (number of columns)

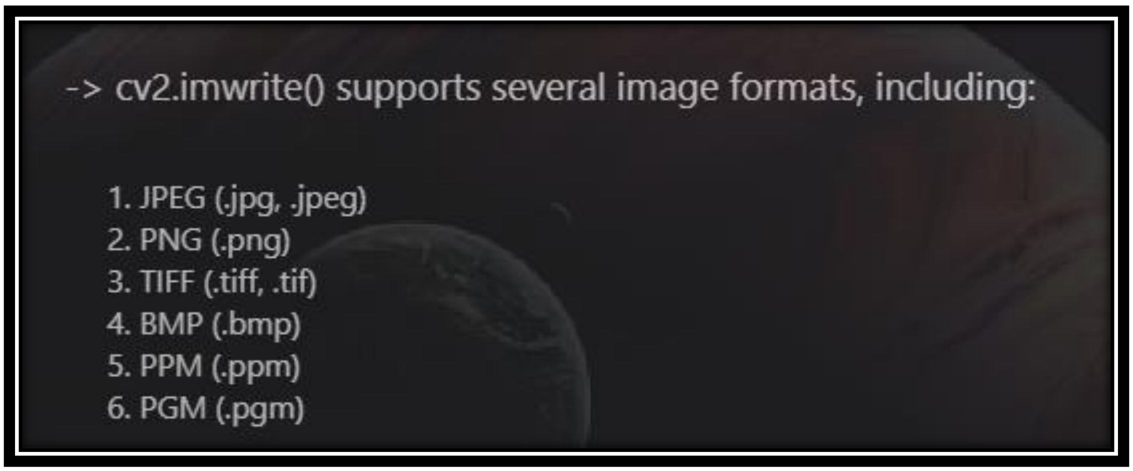


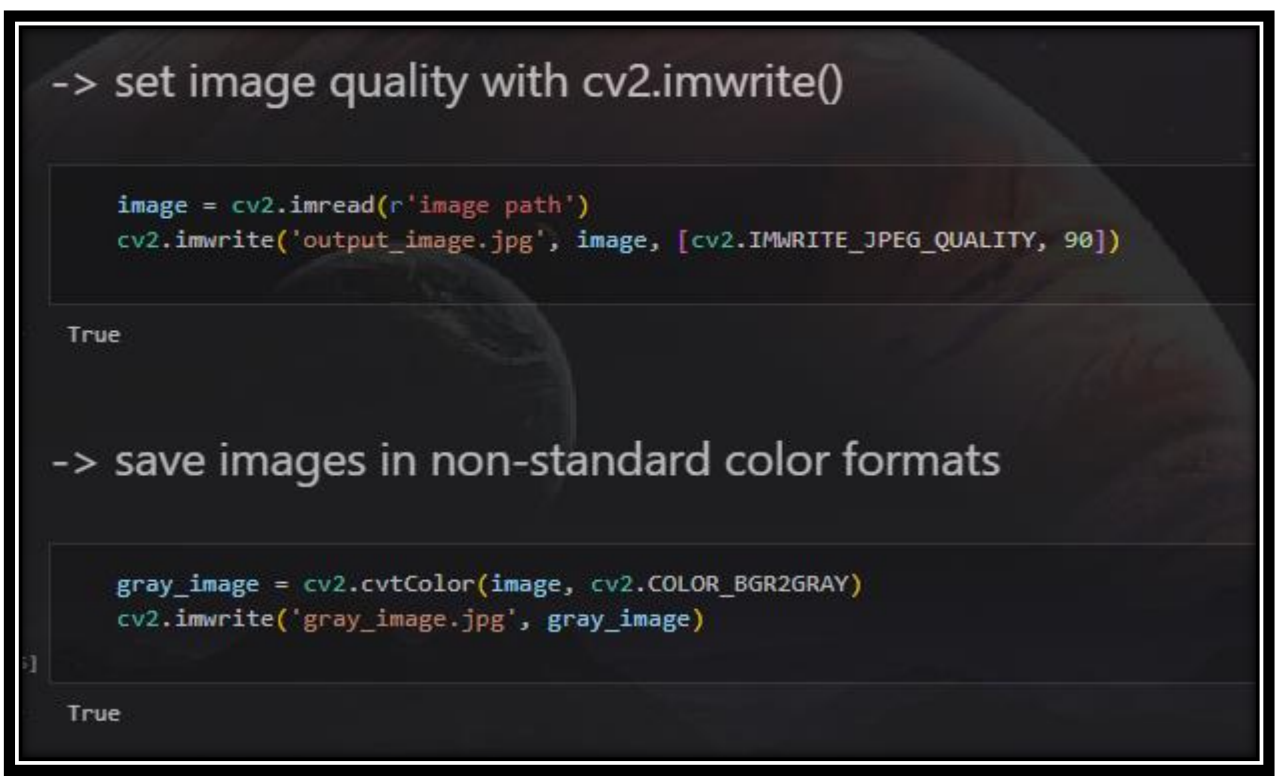

- key = cv2.waitKey(0):
- Waits indefinitely for a key press and stores the ASCII value of the pressed key.
- if key == ord('q'):
- Checks if the pressed key is 'q' (by comparing its ASCII value using ord('q')).
- print("Q key pressed"):
- If 'q' is pressed, it prints "Q key pressed" to the console.


- image1 = cv2.imread(r'image path')
- Reads the first image from the specified path.
- image2 = cv2.imread(r'image path')
- Reads the second image from the specified path.
- cv2.imshow('Image 1', image1)
- Displays image1 in a window titled **"Image 1"**.
- cv2.imshow('Image 2', image2)
- Displays image2 in a separate window titled **"Image 2"**.
- cv2.waitKey(0)
- Waits indefinitely for a key press.
- cv2.destroyAllWindows()
- Closes all OpenCV windows when a key is pressed.


- cv2.destroyAllWindows()
- Closes all previously opened OpenCV windows.
- cv2.destroyWindow('Window Name')

- Closes a specific window with the given name **"Window Name"** if it exists.
- cv2.imshow('Window Name', image)
- Displays the image in a window named **"Window Name"**.
- cv2.waitKey(0)
- Waits indefinitely for a key press.
- cv2.destroyAllWindows()
- Closes all OpenCV windows when a key is pressed.



- **Set Image Quality:** Saves an image as JPEG with quality **90** using cv2.imwrite().
- **Save in Grayscale:** Converts an image to **grayscale** with cv2.cvtColor() and saves it.

-> Visualizing the different color channels of an RGB image.

```python
image = cv2.imread(r'image path')
B, G, R = cv2.split(image)
cv2.imshow("original", image)
cv2.waitKey(0)
cv2.imshow("blue", B)
cv2.waitKey(0)
cv2.imshow("Green", G)
cv2.waitKey(0)
cv2.imshow("red", R)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- This code splits an RGB image into **Blue, Green, and Red** channels using cv2.split() and displays each channel separately using cv2.imshow().

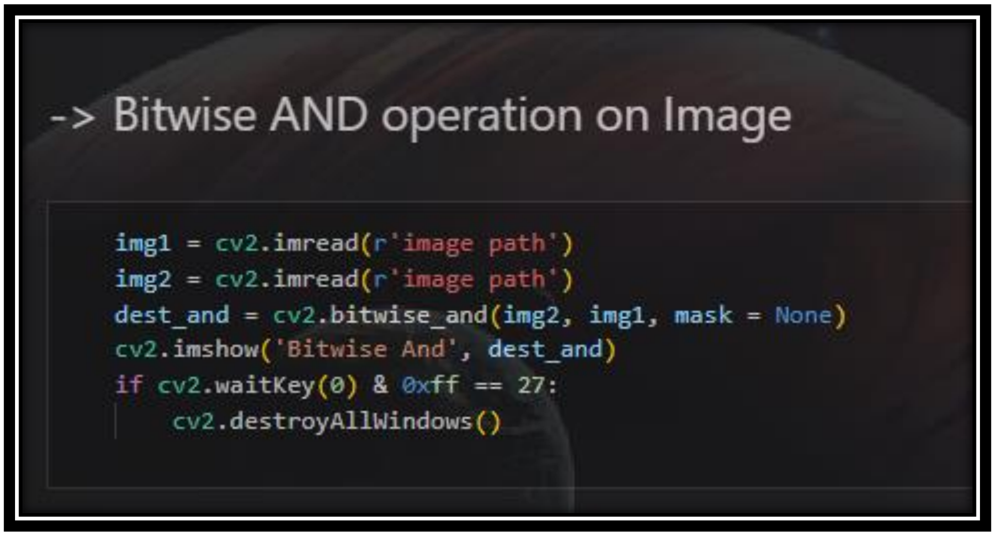
-> Addition of two images

```python
image1 = cv2.imread(r'image path')
image2 = cv2.imread(r'image path')
weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.8, 0)
cv2.imshow('Weighted Image', weightedSum)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

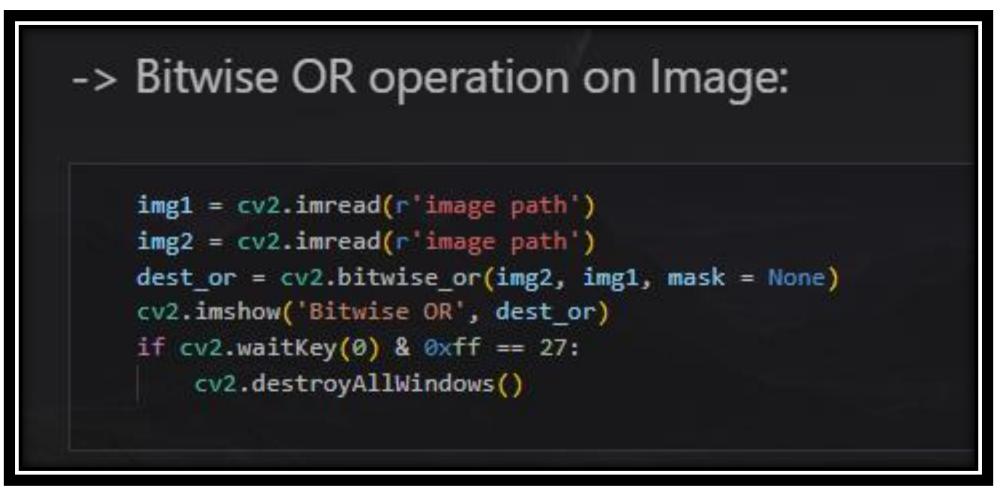
-> Subtraction of pixels of two images

```python
image1 = cv2.imread(r'image path')
image2 = cv2.imread(r'image path')
sub = cv2.subtract(image1, image2)
cv2.imshow('Subtracted Image', sub)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

- **Addition:** Uses cv2.addWeighted() to blend two images with specified weights.
- **Subtraction:** Uses cv2.subtract() to compute pixel-wise differences between two images.




- cv2.bitwise_and(img2, img1, mask=None): Applies a pixel-wise AND operation on img1 and img2. Only overlapping white regions will be retained.
- The result is displayed using cv2.imshow(), and the window closes when the **Esc** key (27) is pressed.



- cv2.bitwise_or(img2, img1, mask=None): Applies a pixel-wise OR operation on img1 and img2. The resulting image retains white regions from both input images.
- The result is displayed using cv2.imshow(), and the window closes when the **Esc** key (27) is pressed.


- **XOR (cv2.bitwise_xor)**: Keeps differing pixels, setting matching ones to black.
- **NOT (cv2.bitwise_not)**: Inverts pixel values (black ↔ white).

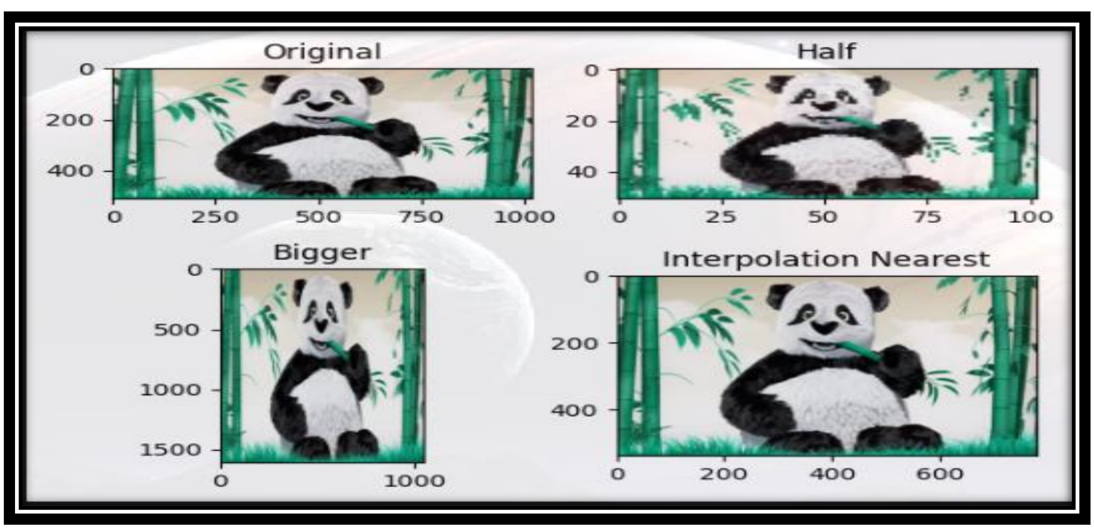
# Image Processing

-> Image Resizing using OpenCV

```
image = cv2.imread(r"image path", 1)
half = cv2.resize(image, (0, 0), fx = 0.1, fy = 0.1)
bigger = cv2.resize(image, (1050, 1610))
stretch_near = cv2.resize(image, (780, 540), interpolation = cv2.INTER_LINEAR)
Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]
images =[image, half, bigger, stretch_near]
count = 4
for i in range(count):
    plt.subplot(2, 2, i + 1)
    plt.title(Titles[i])
    plt.imshow(images[i])
plt.show()
```

- cv2.resize() scales the image:
- fx=0.1, fy=0.1 reduces size.
- (1050, 1610) enlarges it.
- (780, 540) resizes using INTER_LINEAR interpolation.
- plt.subplot() displays the original and resized images in a 2x2 grid.
- plt.title() assigns titles for clarity.

**OUTPUT:**



```
image = cv2.imread(r'image path')
resized_image = cv2.resize(image, (10, 10))
```

-> maintain aspect ratio when resizing images in Python

```
image = cv2.imread(r'image path')
(h, w) = image.shape[:2]
new_width = 800
aspect_ratio = h / w
new_height = int(new_width * aspect_ratio)
resized_image = cv2.resize(image, (new_width, new_height))
```

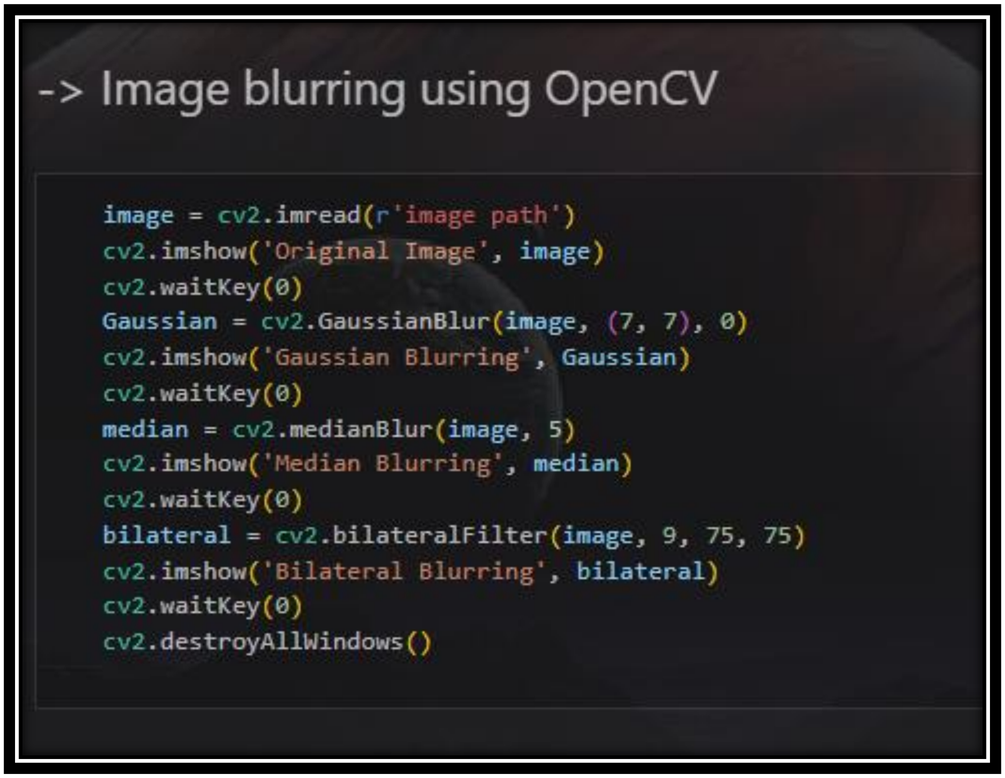
- The first code snippet resizes an image to **(10, 10)** pixels, which may distort it.
- The second snippet maintains the aspect ratio:
- Extracts **height (h) and width (w)**.
- Defines a **new width (800 pixels)**.
- Calculates the **new height** using the aspect ratio (**h/w**).
- Uses cv2.resize() to resize without distortion.

-> cv2.erode() method

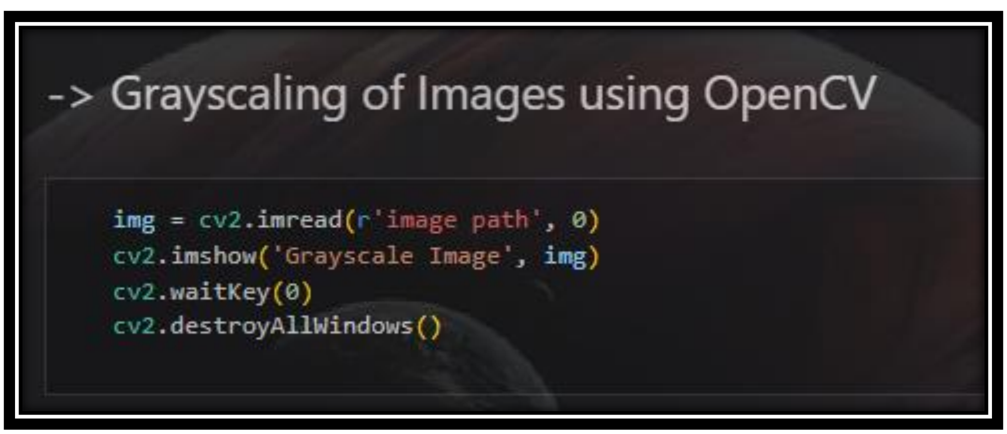
```
path = r'image path'
image = cv2.imread(path)
window_name = 'Image'
kernel = np.ones((5, 5), np.uint8)
image = cv2.erode(image, kernel)
cv2.imshow(window_name, image)
```

- Reads an image from the given path using cv2.imread().
- Creates a **5x5 kernel** of ones (np.ones((5,5), np.uint8)).

- Applies **erosion** using cv2.erode(), which reduces white regions and removes small noise.
- Displays the processed image using cv2.imshow().




- **Gaussian Blur** - Smooths the image using a Gaussian kernel.
- **Median Blur** - Removes noise using the median of pixels.
- **Bilateral Filter** - Reduces noise while preserving edges.




- cv2.imread('image path', 0): Loads the image in grayscale.
- cv2.imshow(): Displays the grayscale image.

- cv2.waitKey(0): Waits for a key press.
- cv2.destroyAllWindows(): Closes the image window.

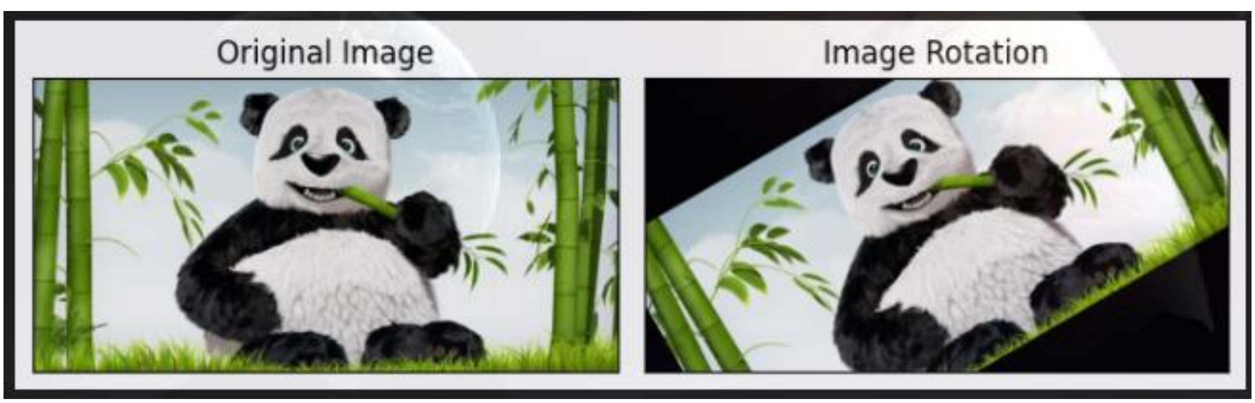
# Scaling, Rotating, Shifting and Edge Detection

-> Rotating

```
img = cv2.imread(r'image path')
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
center = (image_rgb.shape[1] // 2, image_rgb.shape[0] // 2)
angle = 30
scale = 1
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)
rotated_image = cv2.warpAffine(image_rgb, rotation_matrix, (img.shape[1], img.shape[0]))
fig, axs = plt.subplots(1, 2, figsize=(7, 4))
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')
axs[1].imshow(rotated_image)
axs[1].set_title('Image Rotation')
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
```

- The image shows **image rotation using OpenCV**. It loads an image, converts it to RGB, calculates a rotation matrix, applies rotation with cv2.warpAffine(), and displays both the original and rotated images using Matplotlib.

## OUTPUT:



-> shearing

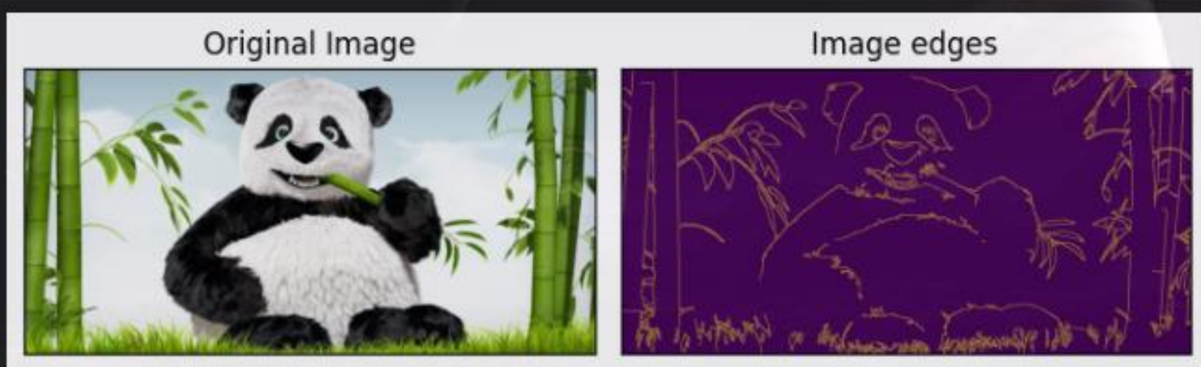
```
image = cv2.imread(r'image path')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
width = image_rgb.shape[1]
height = image_rgb.shape[0]
shearX = -0.15
shearY = 0
transformation_matrix = np.array([[1, shearX, 0], [0, 1, shearY]], dtype=np.float32)
sheared_image = cv2.warpAffine(image_rgb, transformation_matrix, (width, height))
fig, axs = plt.subplots(1, 2, figsize=(7, 4))
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')
axs[1].imshow(sheared_image)
axs[1].set_title('Sheared image')
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
```

- The image shows **image shearing using OpenCV**. It reads an image, applies a shearing transformation using a transformation matrix, and displays both the original and sheared images using Matplotlib.

## <u>OUTPUT:</u>

## -> Edge detection of Image

```
img = cv2.imread(r'image path')
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
edges = cv2.Canny(image= image_rgb, threshold1=100, threshold2=700)
fig, axs = plt.subplots(1, 2, figsize=(7, 4))
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')
axs[1].imshow(edges)
axs[1].set_title('Image edges')
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
```



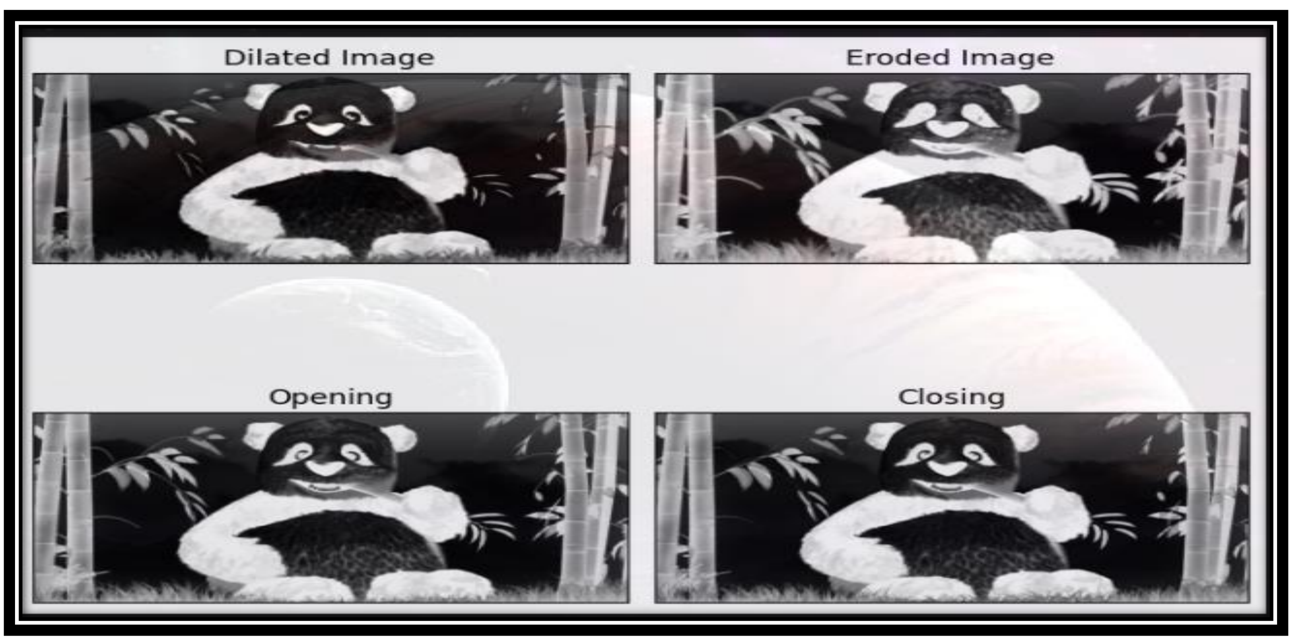

- **Reads** the image and converts it to RGB.
- **Applies Canny Edge Detection** to highlight edges.
- **Displays** both the original and processed images in a subplot.

-> Morphological Image Processing

```python
image = cv2.imread(r'image path')
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
kernel = np.ones((3, 3), np.uint8)
dilated = cv2.dilate(image_gray, kernel, iterations=2)
eroded = cv2.erode(image_gray, kernel, iterations=2)
opening = cv2.morphologyEx(image_gray, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(image_gray, cv2.MORPH_CLOSE, kernel)
fig, axs = plt.subplots(2, 2, figsize=(7, 7))
axs[0,0].imshow(dilated, cmap='Greys')
axs[0,0].set_title('Dilated Image')
axs[0,0].set_xticks([])
axs[0,0].set_yticks([])
axs[0,1].imshow(eroded, cmap='Greys')
axs[0,1].set_title('Eroded Image')
axs[0,1].set_xticks([])
axs[0,1].set_yticks([])
axs[1,0].imshow(opening, cmap='Greys')
axs[1,0].set_title('Opening')
axs[1,0].set_xticks([])
axs[1,0].set_yticks([])
axs[1,1].imshow(closing, cmap='Greys')
axs[1,1].set_title('Closing')
axs[1,1].set_xticks([])
axs[1,1].set_yticks([])
plt.tight_layout()
plt.show()
```

- This code demonstrates **morphological operations** in image processing using OpenCV and Matplotlib. These operations are typically used in **noise removal, shape detection, and image enhancement**, especially in binary or grayscale images.

## **OUTPUT:**



-> Erosion and Dilation

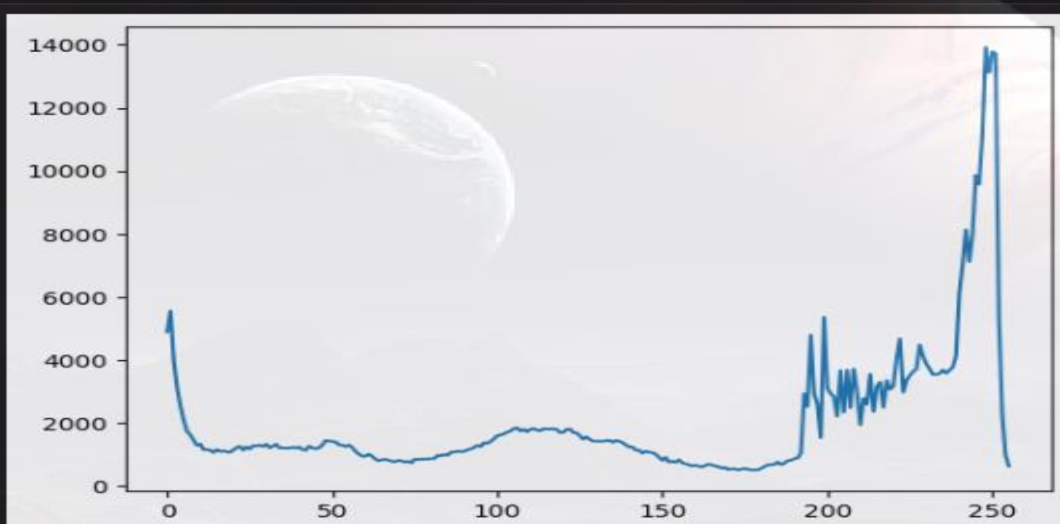
```
img = cv2.imread(r'image path', 0)
kernel = np.ones((5, 5), np.uint8)
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)
cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)
cv2.waitKey(0)
```

```
-1
```

- **Erosion** removes noise, separates touching objects.
- **Dilation** enhances important features, fills gaps.
- Useful in **preprocessing for edge detection, segmentation, and OCR**.

-> image using Histogram

```
img = cv2.imread(r'image path',0)
histr = cv2.calcHist([img],[0],None,[256],[0,256])
plt.plot(histr)
plt.show()
```




- This histogram shows the distribution of pixel intensities in a grayscale image. It helps analyze contrast and brightness. The peaks at the ends indicate a mix of dark and bright regions, suggesting high contrast.

-> Thresholding techniques using OpenCV

```python
image1 = cv2.imread(r'image path')
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)
cv2.imshow('Binary Threshold', thresh1)
cv2.imshow('Binary Threshold Inverted', thresh2)
cv2.imshow('Truncated Threshold', thresh3)
cv2.imshow('Set to 0', thresh4)
cv2.imshow('Set to 0 Inverted', thresh5)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

- This code applies **five thresholding techniques** in OpenCV to segment a grayscale image: **Binary, Inverted Binary, Truncated, To Zero, and Inverted To Zero.** It helps in **image segmentation and preprocessing** for computer vision tasks.

-> Thresholding techniques using OpenCV

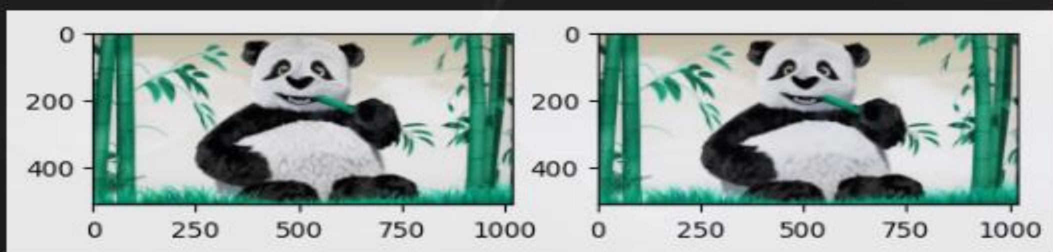
```python
image1 = cv2.imread(r'image path')
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
thresh1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 199, 5)
thresh2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 199, 5)
cv2.imshow('Adaptive Mean', thresh1)
cv2.imshow('Adaptive Gaussian', thresh2)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

- This code applies **adaptive thresholding** in OpenCV using **mean** and **Gaussian** methods. It dynamically adjusts the threshold for different image regions, making it useful for images with varying lighting conditions.

-> Filter Color with OpenCV

-> Denoising of colored images using opencv

```
img = cv2.imread(r'image path')
dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 15)
plt.subplot(121), plt.imshow(img)
plt.subplot(122), plt.imshow(dst)
plt.show()
```




- This code performs **color image denoising** using OpenCV's fastNlMeansDenoisingColored() function, reducing noise while preserving image details. It compares the original and denoised images using Matplotlib.

-> Find Co-ordinates of Contours using OpenCV

```
font = cv2.FONT_HERSHEY_COMPLEX
img2 = cv2.imread(r'image path', cv2.IMREAD_COLOR)
img = cv2.imread(r'image path', cv2.IMREAD_GRAYSCALE)
_, threshold = cv2.threshold(img, 110, 255, cv2.THRESH_BINARY)
contours, _= cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours :
    approx = cv2.approxPolyDP(cnt, 0.009 * cv2.arcLength(cnt, True), True)
    cv2.drawContours(img2, [approx], 0, (0, 0, 255), 5)
    n = approx.ravel()
    i = 0
    for j in n :
        if(i % 2 == 0):
            x = n[i]
            y = n[i + 1]
            string = str(x) + " " + str(y)
            if(i == 0):
                cv2.putText(img2, "Arrow tip", (x, y),
                                font, 0.5, (255, 0, 0))
            else:
                cv2.putText(img2, string, (x, y),
                        font, 0.5, (0, 255, 0))
        i = i + 1
cv2.imshow('image2', img2)
if cv2.waitKey(0) & 0xFF == ord('q'):
    cv2.destroyAllWindows()
```

- This code detects contours in an image using OpenCV, finds their coordinates, and labels them. It first converts the image to grayscale, applies thresholding, and then detects contours.

## -> Image inpainting

```
damaged_img = cv2.imread(filename=r"image path")
height, width = damaged_img.shape[0], damaged_img.shape[1]
for i in range(height):
    for j in range(width):
        if damaged_img[i, j].sum() > 0:
            damaged_img[i, j] = 0
        else:
            damaged_img[i, j] = [255, 255, 255]
mask = damaged_img
cv2.imwrite('mask.jpg', mask)
cv2.imshow("damaged image mask", mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## -> Intensity Transformation Operations on Images

```
img = cv2.imread(r'image path')
for gamma in [0.1, 0.5, 1.2, 2.2]:
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')
    cv2.imwrite('gamma_transformed'+str(gamma)+'.jpg', gamma_corrected)
```

## -> Background subtraction using OpenCV

```
cap = cv2.VideoCapture(r'image path')
fgbg = cv2.createBackgroundSubtractorMOG2()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('fgmask', fgmask)
    cv2.imshow('frame',frame )
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

- This code snippet performs image inpainting using OpenCV. It reads a damaged image, processes it by converting non-zero pixels to black and others to white, and saves the result as a mask. The mask is displayed using OpenCV's imshow().

-> Background Subtraction in an Image using Concept of Running Average

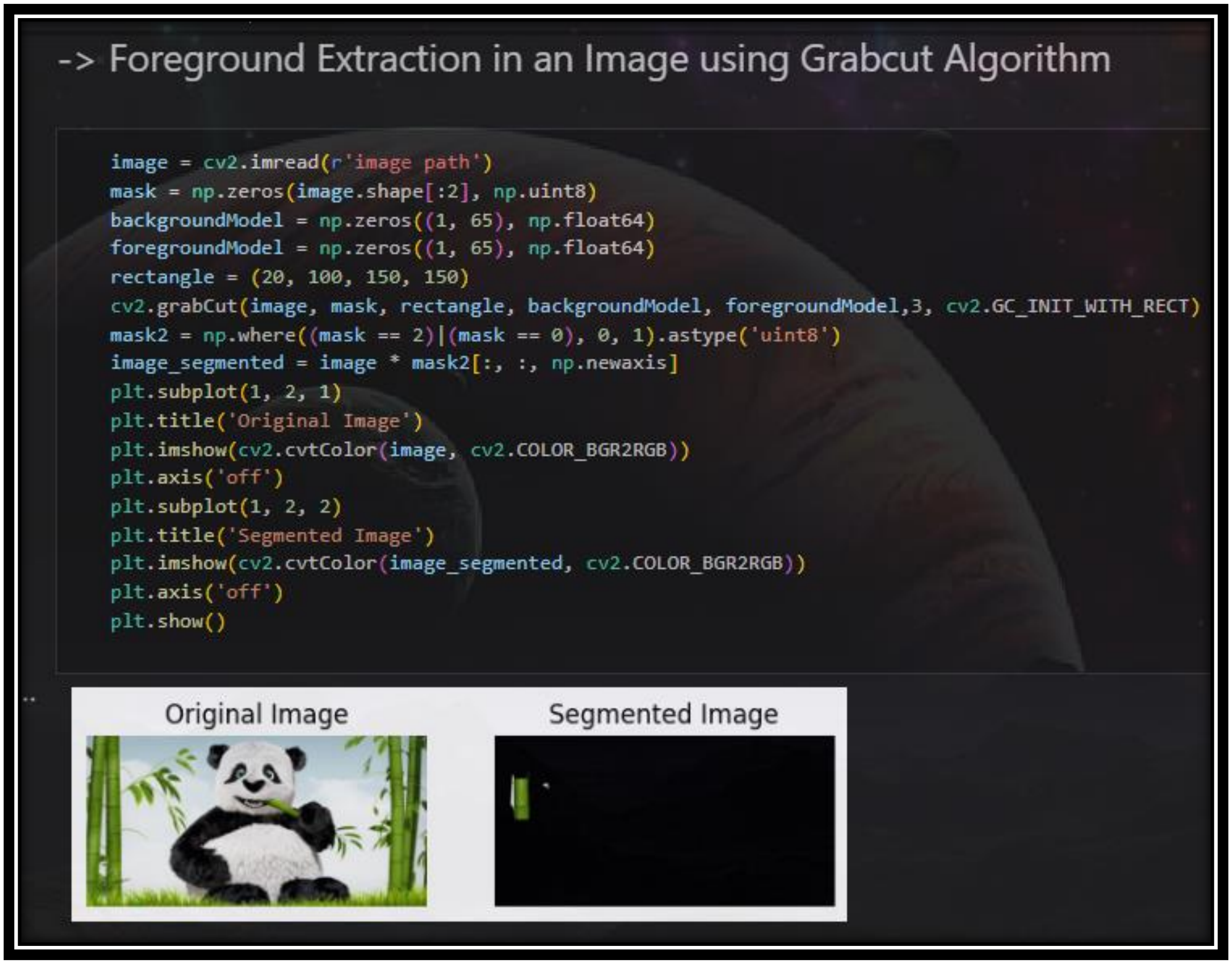
```
cap = cv2.VideoCapture(0)
_, img = cap.read()
averageValue1 = np.float32(img)
while(1):
    _, img = cap.read()
    cv2.accumulateWeighted(img, averageValue1, 0.02)
    resultingFrames1 = cv2.convertScaleAbs(averageValue1)
    cv2.imshow(r'image1 path', img)
    cv2.imshow(r'image2 path', resultingFrames1)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

- The image contains OpenCV code for **Background Subtraction using Running Average**. It continuously updates a background model using cv2.accumulateWeighted() and displays both the original frame and the extracted foreground.

-> Image Translation

```
image = cv2.imread(r'image path')
height, width = image.shape[:2]
quarter_height, quarter_width = height / 4, width / 4
T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])
img_translation = cv2.warpAffine(image, T, (width, height))
cv2.imshow("Originalimage", image)
cv2.imshow('Translation', img_translation)
cv2.waitKey()
cv2.destroyAllWindows()
```

- The image using OpenCV for **Image Translation**. It loads an image, calculates a translation matrix to shift the image by a quarter of its width and height, and applies **cv2.warpAffine()** to translate the image. The original and translated images are displayed.




- The image contains using OpenCV for **Foreground Extraction using the GrabCut Algorithm**. It applies GrabCut with an initial bounding box to segment the foreground from the background and displays both the original and segmented images.

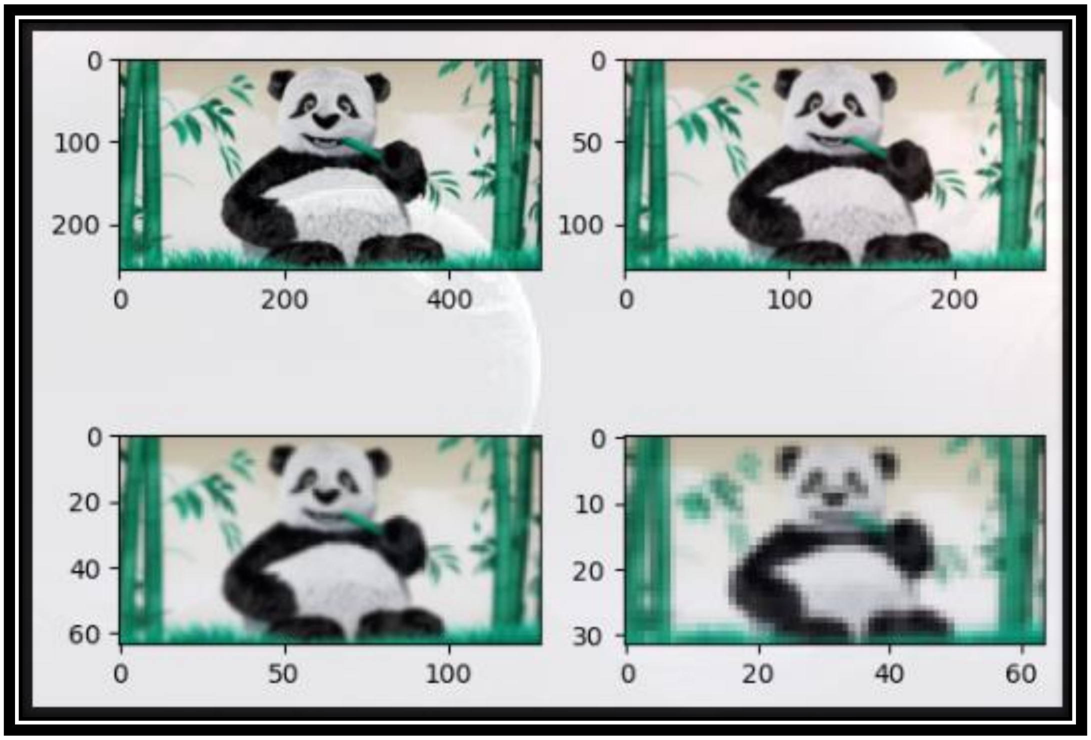
-> Morphological Operations in Image Processing (Gradient)

```
screenRead = cv2.VideoCapture(0)
while(1):
    _, image = screenRead.read()
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    blue1 = np.array([110, 50, 50])
    blue2 = np.array([130, 255, 255])
    mask = cv2.inRange(hsv, blue1, blue2)
    res = cv2.bitwise_and(image, image, mask = mask)
    kernel = np.ones((5, 5), np.uint8)
    gradient = cv2.morphologyEx(mask, cv2.MORPH_GRADIENT, kernel)
    cv2.imshow('Gradient', gradient)
    if cv2.waitKey(1) & 0xFF == ord('a'):
        break
cv2.destroyAllWindows()
screenRead.release()
```

- The image contains using OpenCV for **Morphological Operations in Image Processing (Gradient)**. It captures a video stream, converts frames to HSV, applies a color mask for blue regions, and uses **morphological gradient** to highlight edges. The processed result is displayed in real-time.

-> Image Pyramid

```
img = cv2.imread(r"image path")
layer = img.copy()
for i in range(4):
    plt.subplot(2, 2, i + 1)
    layer = cv2.pyrDown(layer)
    plt.imshow(layer)
    cv2.imshow("str(i)", layer)
    cv2.waitKey(0)
cv2.destroyAllWindows()
```




## 2.3 Feature Detection and Description

-> Line detection

```
img = cv2.imread(r'image path')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
for r_theta in lines:
    arr = np.array(r_theta[0], dtype=np.float64)
    r, theta = arr
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*r
    y0 = b*r
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)
cv2.imwrite('linesDetected.jpg', img)
```

```
True
```

- The image contains demonstrating **Image Pyramid** using OpenCV. It loads an image and applies **cv2.pyrDown()** in a loop to create progressively smaller versions of the image. The images at different pyramid levels are displayed using **cv2.imshow()**.


- The image contains demonstrating **Line Detection** using OpenCV. The process involves:
- Reading an image using cv2.imread().
- Converting it to grayscale with cv2.cvtColor().
- Detecting edges using the **Canny edge detector** (cv2.Canny()).
- Using the **Hough Line Transform** (cv2.HoughLines()) to detect straight lines.
- Iterating over detected lines to compute endpoints and drawing them using cv2.line().
- Saving the output image with detected lines using cv2.imwrite().

-> Corner detection with Harris Corner Detection method using OpenCV

```
image = cv2.imread(r'image path')
operatedImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
operatedImage = np.float32(operatedImage)
dest = cv2.cornerHarris(operatedImage, 2, 5, 0.07)
dest = cv2.dilate(dest, None)
image[dest > 0.01 * dest.max()]=[0, 0, 255]
cv2.imshow('Image with Borders', image)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

- **Read an image** using cv2.imread() & **Convert it to grayscale** cv2.cvtColor().
- **Converting the grayscale image to float32** for processing.
- **Applying Harris Corner Detection** using cv2.cornerHarris().
- **Dilating the detected corners** to enhance them.
- **Marking strong corners** on the original image.
- **Displaying the result** using cv2.imshow().
- **Waiting for a key press** and closing windows when 'Esc' is pressed.

-> Find Circles and Ellipses

```
image = cv2.imread(r'image path', 0)
params = cv2.SimpleBlobDetector_Params()
params.filterByArea = True
params.minArea = 100
params.filterByCircularity = True
params.minCircularity = 0.9
params.filterByConvexity = True
params.minConvexity = 0.2
params.filterByInertia = True
params.minInertiaRatio = 0.01
detector = cv2.SimpleBlobDetector_create(params)
keypoints = detector.detect(image)
blank = np.zeros((1, 1))
blobs = cv2.drawKeypoints(image, keypoints, blank, (0, 0, 255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
number_of_blobs = len(keypoints)
text = "Number of Circular Blobs: " + str(len(keypoints))
cv2.putText(blobs, text, (20, 550), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 100, 255), 2)
cv2.imshow("Filtering Circular Blobs Only", blobs)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Reading the image** using cv2.imread().
- **Setting parameters for blob detection**:

- **Creating a blob detector** using cv2.SimpleBlobDetector_create().
- **Detecting keypoints (circular blobs)** in the image.
- **Drawing detected keypoints** on the image.
- **Displaying the number of detected circular blobs** using cv2.putText().
- **Showing the output image** with detected blobs.
- **Waiting for a key press and closing the window**.
- This method is useful for **detecting circular objects** such as coins, bubbles, or cells in an image.

-> Document field detection using Template Matching

```
field_threshold = { "prev_policy_no" : 0.7, "address"    : 0.6, }
def getBoxed(img, img_gray, template, field_name = "policy_no"):
    w, h = template.shape[::-1]
    res = cv2.matchTemplate(img_gray, template, cv2.TM_CCOEFF_NORMED)
    hits = np.where(res >= field_threshold[field_name])
    for pt in zip(*hits[::-1]):
        cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2)
        y = pt[1] - 10 if pt[1] - 10 > 10 else pt[1] + h + 20
        cv2.putText(img, field_name, (pt[0], y), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 1)
    return img
if __name__ == '__main__':
    img = cv2.imread('doc.png')
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    template_add = cv2.imread(r'image1 path', 0)
    template_prev = cv2.imread(r'image2 path', 0)
    img = getBoxed(img.copy(), img_gray.copy(), template_add, 'address')
    img = getBoxed(img.copy(), img_gray.copy(),template_prev, 'prev_policy_no')
    cv2.imshow('Detected', img)
```

- **Implementing the getBoxed function**, which:
- Uses cv2.matchTemplate() to locate template positions.
- Draws bounding boxes around detected fields using cv2.rectangle().
- Adds text labels using cv2.putText().
- **Processing the main document image (doc.png)**:
- Loads template images (image1 path, image2 path).
- Calls getBoxed() for both templates.
- **Displaying the detected fields** in the image using cv2.imshow().
- This method is useful for **automating document field extraction** in applications like OCR, form processing.

-> Drawing a line on black screen using numpy library

```
Img = np.zeros((512, 512, 3), dtype='uint8')
start_point = (100, 100)
end_point = (450, 450)
color = (255, 250, 255)
thickness = 9
image = cv2.line(Img, start_point, end_point, color, thickness)
cv2.imshow('Drawing_Line', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- The image shows Python code for drawing a **white diagonal line** on a black screen using **NumPy and OpenCV**. It creates a **512x512 black image**, defines the **start and end points**, sets the **color (white)** and **thickness (9px)**, draws the line with cv2.line(), and displays it using cv2.imshow().

-> cv2.arrowedLine() method

```
path = r'image path'
image = cv2.imread(path)
window_name = 'Image'
start_point = (0, 0)
end_point = (200, 200)
color = (0, 255, 0)
thickness = 9
image = cv2.arrowedLine(image, start_point, end_point, color, thickness)
cv2.imshow(window_name, image)
```

- The image contains Python code demonstrating the **cv2.arrowedLine()** method in OpenCV. It loads an image, defines **start and end points**, sets the **color (green)** and **thickness (9px)**, then draws an **arrowed line** from (0,0) to (200,200). The image is displayed using cv2.imshow().

-> cv2.ellipse() method

```python
path = r'image path'
image = cv2.imread(path)
window_name = 'Image'
center_coordinates = (120, 100)
axesLength = (100, 50)
angle = 0
startAngle = 0
endAngle = 360
color = (0, 0, 255)
thickness = 5
image = cv2.ellipse(image, center_coordinates, axesLength, angle, startAngle, endAngle, color, thickness)
cv2.imshow(window_name, image)
```

- The image contains Python code demonstrating the **cv2.ellipse()** method in OpenCV. It loads an image and draws an **ellipse** centered at (120,100) with axes lengths (100,50). The ellipse is **red (0,0,255)** with a **thickness of 5px** and covers a full **360-degree** arc. The image is then displayed using cv2.imshow().

-> cv2.rectangle() method

```python
path = r'image path'
image = cv2.imread(path)
window_name = 'Image'
start_point = (5, 5)
end_point = (220, 220)
color = (255, 0, 0)
thickness = 2
image = cv2.rectangle(image, start_point, end_point, color, thickness)
cv2.imshow(window_name, image)
```

- The image contains Python code demonstrating the **cv2.rectangle()** method in OpenCV. It loads an image and draws a **red rectangle (255, 0, 0)** from the **start point (5,5)** to the **end point (220,220)** with a **thickness of 2 pixels**. The image is then displayed using cv2.imshow().

-> cv2.putText() method

```
path = r'image path'
image = cv2.imread(path)
window_name = 'Image'
text = 'GeeksforGeeks'
font = cv2.FONT_HERSHEY_SIMPLEX
org = (00, 185)
fontScale = 1
color = (0, 0, 255)
thickness = 2
image = cv2.putText(image, text, org, font, fontScale,color, thickness, cv2.LINE_AA, False)
image = cv2.putText(image, text, org, font, fontScale,color, thickness, cv2.LINE_AA, True)
cv2.imshow(window_name, image)
```

- The image contains Python code demonstrating the **cv2.putText()** method in OpenCV. It loads an image and adds the text **"GeeksforGeeks"** at the **position (0,185)** using the **FONT_HERSHEY_SIMPLEX** font with a **font scale of 1**, **red color (0, 0, 255)**, and **thickness of 2 pixels**.

-> Find and Draw Contours using OpenCV

```
image = cv2.imread(r'image path')
cv2.waitKey(0)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edged = cv2.Canny(gray, 30, 200)
cv2.waitKey(0)
contours, hierarchy = cv2.findContours(edged,
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
cv2.imshow('Canny Edges After Contouring', edged)
cv2.waitKey(0)
print("Number of Contours found = " + str(len(contours)))
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Number of Contours found = 881
```

- The image contains Python code demonstrating how to **find and draw contours using OpenCV**. It converts an image to grayscale, applies Canny edge detection, finds contours using cv2.findContours(), and then draws them on the original image using cv2.drawContours(). The total number of contours found is printed, and the results are displayed using cv2.imshow().