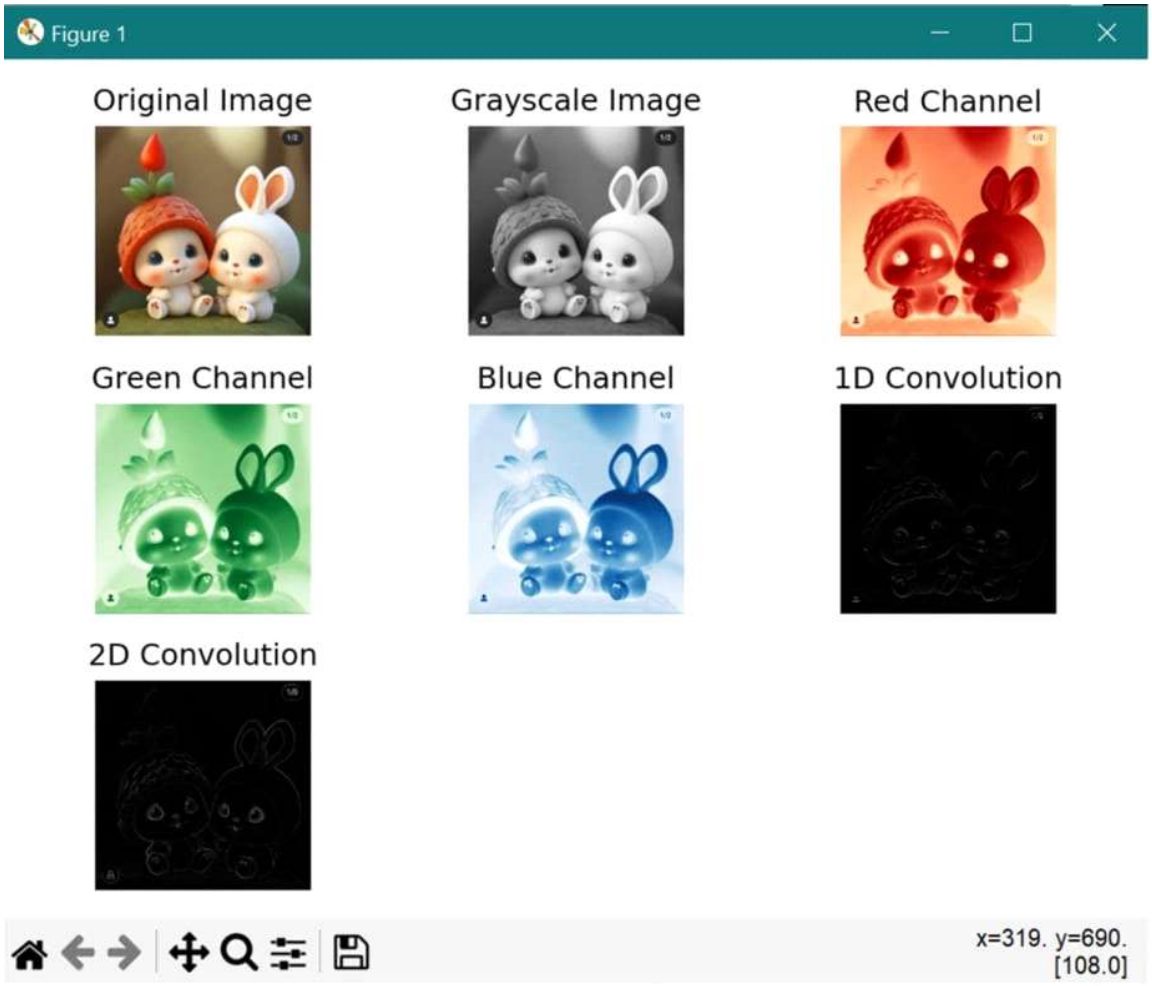
1.Blue , Green , Red , 1D , 2D convolution

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

image = cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf cirtificate.jpg')

plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.subplot(2, 3, 2)
plt.title('Grayscale Image')
plt.imshow(gray_image,cmap='gray')
plt.axis('off')

b, g, r = cv2.split(image)
plt.subplot(2, 3, 3)
plt.title('Red Channel')
plt.imshow(r, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title('Green Channel')
plt.imshow(g, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title('Blue Channel')
plt.imshow(b, cmap='gray')
plt.axis('off')

kernel_1d = np.array([1, 0, -1])
conv_1d_image = cv2.filter2D(gray_image, -1, kernel_1d)
plt.subplot(2, 3, 6)
plt.title('1D Convolution')
plt.imshow(conv_1d_image, cmap='gray')
plt.axis('off')


kernel_2d = np.array([[1, 1, 1],
 [1, -8, 1],
 [1, 1, 1]])
conv_2d_image = cv2.filter2D(gray_image, -1, kernel_2d)
plt.subplot(2, 3, 6)
```

```
plt.title('2D Convolution')
plt.imshow(conv_2d_image, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()
```



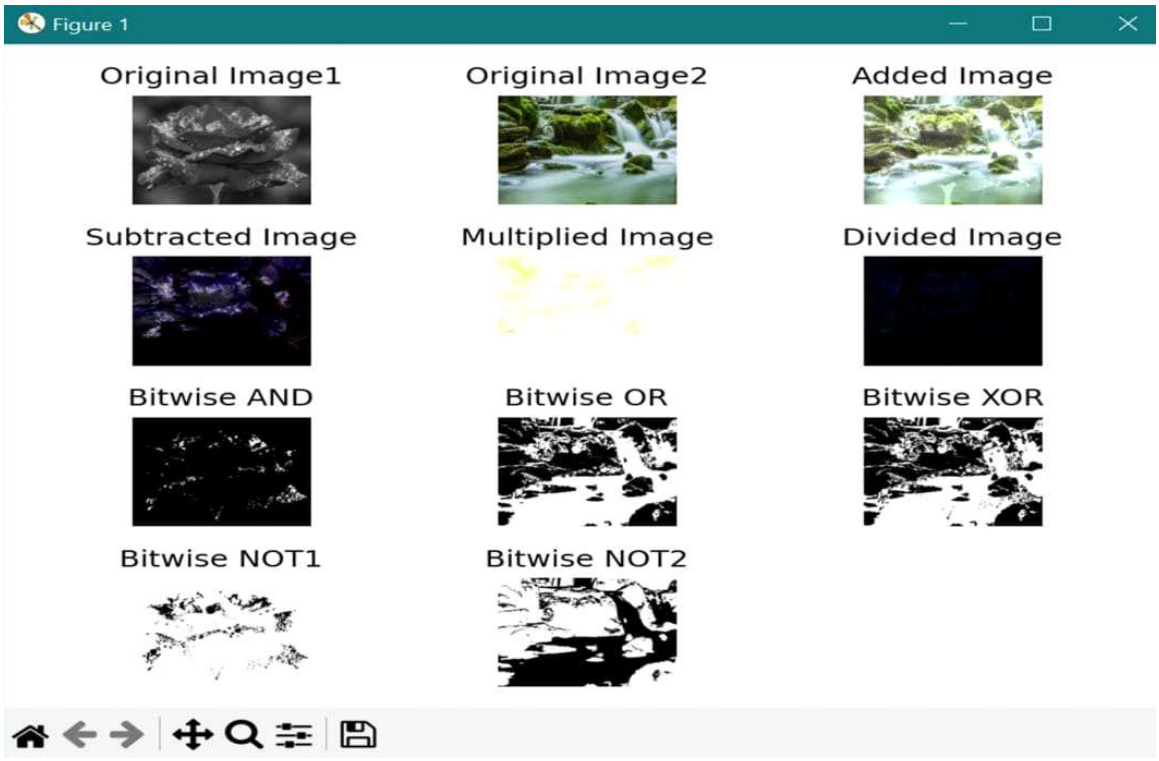
2.Arithmetic and logic operations

```
import cv2

import numpy as np

image1=cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf
```

```
cirtificate.jpg')
image2=cv2.imread('C:/Users/sharana/OneDrive/Pictures/WIN_
20230620_15_39_26_Pro.jpg')
if image1.shape!=image2.shape:
    image2=cv2.resize(image2,(image1.shape[1],image1.shape[0]))


addition=cv2.add(image1,image2)
subtraction=cv2.subtract(image1,image2)
bitwise_and=cv2.bitwise_and(image1,image2)
bitwise_or=cv2.bitwise_or(image1,image2)
bitwise_xor=cv2.bitwise_xor(image1,image2)
bitwise_not_image1=cv2.bitwise_not(image1)
bitwise_not_image2=cv2.bitwise_not(image2)
cv2.imshow('image1',image1)
cv2.imshow('image2',image2)
cv2.imshow('Addtion',addition)
cv2.imshow('Substrution',subtraction)
cv2.imshow('Bitwise And',bitwise_and)
cv2.imshow('Bitwise OR',bitwise_or)
cv2.imshow('Bitwise xor',bitwise_xor)
cv2.imshow('Bitwise NOT image1',bitwise_not_image1)
cv2.imshow('Bitwise NOT image2',bitwise_not_image2)
```

plt.tight_layout()

plt.show()



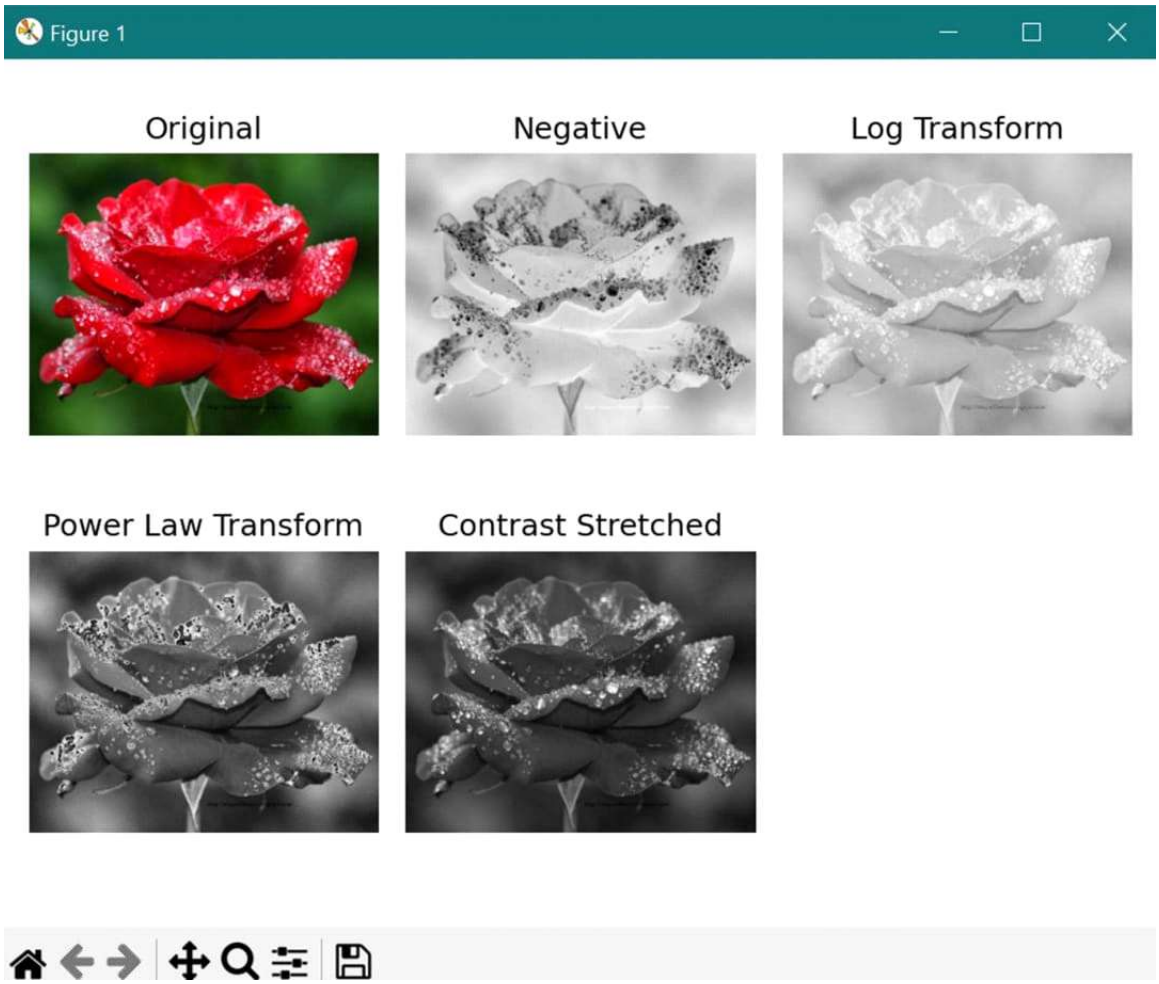
3.Gray Level transformation

import cv2

import numpy as np

image = cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf cirtificate.jpg', cv2.IMREAD_GRAYSCALE)

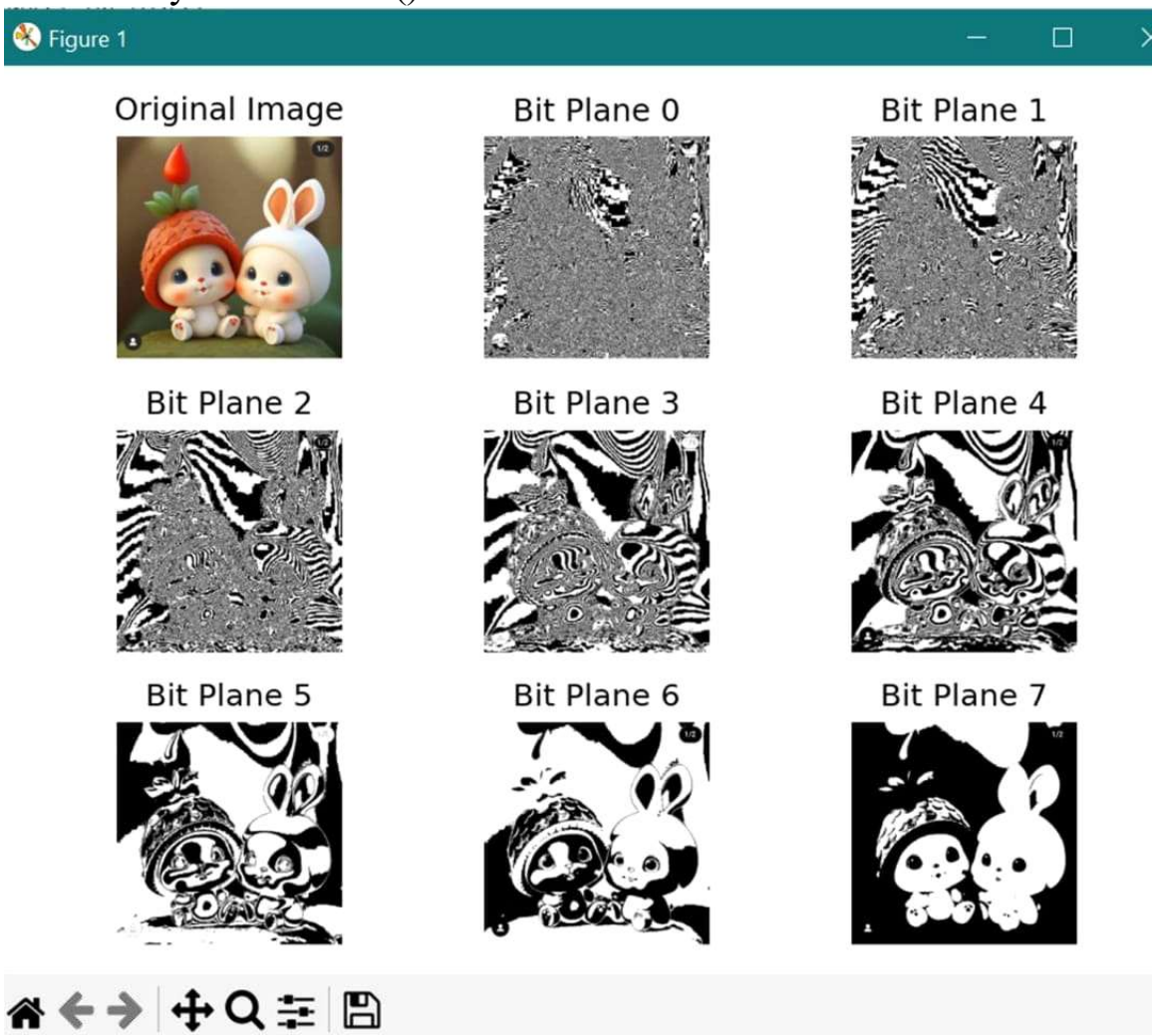
```
negative_image = 255 - image

c = 255 / np.log(1 + np.max(image))

log_transformed = c * (np.log(image + 1))

gamma = 0.5

power_law_transformed = np.power(image / float(np.max(image)),
gamma) * 255

min_intensity = np.min(image)

max_intensity = np.max(image)

a = 0

b = 255

contrast_stretched = (image - min_intensity) * ((b - a) /
(max_intensity - min_intensity)) + a

log_transformed = np.uint8(log_transformed)

power_law_transformed = np.uint8(power_law_transformed)

contrast_stretched = np.uint8(contrast_stretched)

cv2.imshow('Original Image', image)

cv2.imshow('Negative Transformation', negative_image)

cv2.imshow('Log Transformation', log_transformed)

cv2.imshow('Power-law Transformation', power_law_transformed)

cv2.imshow('Contrast Stretched', contrast_stretched)

cv2.waitKey(0)

cv2.destroyAllWindows()
```



4.Bit plane slicing

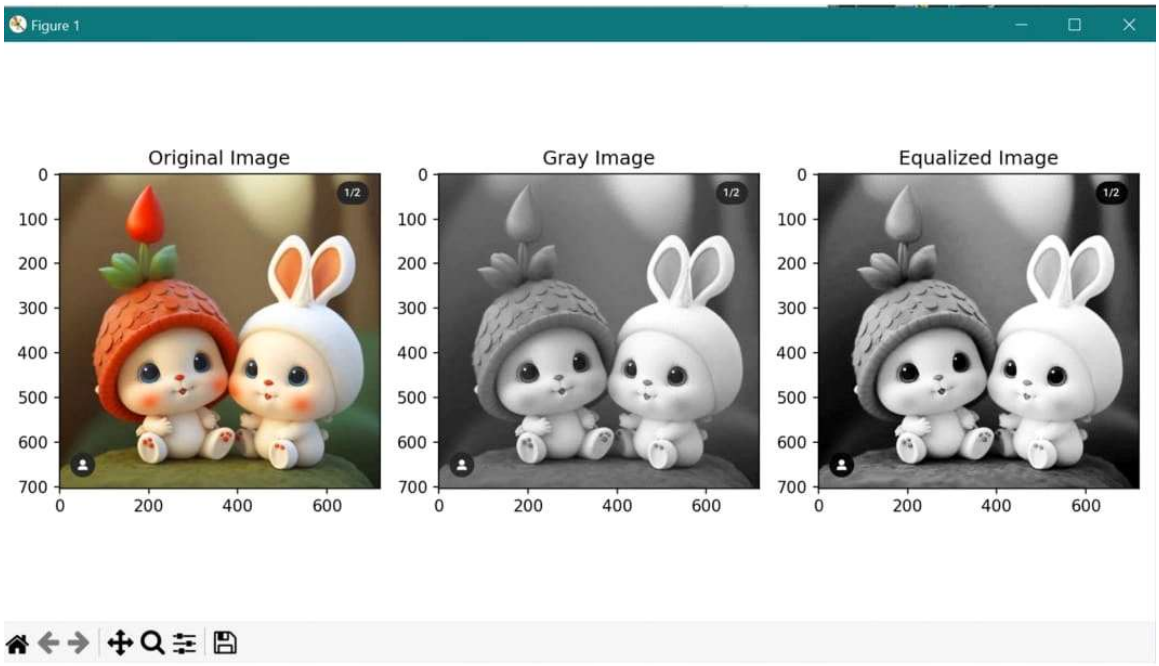
```
import cv2
import numpy as np
def bit_plane_slice(image,bit):
    plane=np.bitwise_and(image,2**bit)
    return plane
image=cv2.imread('C:/Users/sharana/One
Drive/Pictures/WIN_20230620_15_39_
26
```

```
_Pro.jpg',cv2.IMREAD_GRAYSCALE)
num_bits=8
bit_planes=[bit_plane_slice(image,bit)for
bit in range(num_bits)]
for bit,plane in enumerate(bit_planes):
    cv2.imshow(f'bit plane{bit}',plane)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



5.Histogram equilisation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



image=cv2.imread('C:/Users/sharana/OneDrive/Pictures/WIN_20230620_15_39_26
_Pro.jpg',cv2.IMREAD_GRAYSCALE)

equalized_image=cv2.equalizeHist(image)

plt.figure(figsize=(10,15))

plt.subplot(1,2,1)

plt.imshow(image,cmap='gray')

plt.title('original image')



plt.subplot(1,2,2)

plt.imshow(equalized_image,cmap='gray')

plt.title('equalized image')



plt.tight_layout()

plt.show()
```

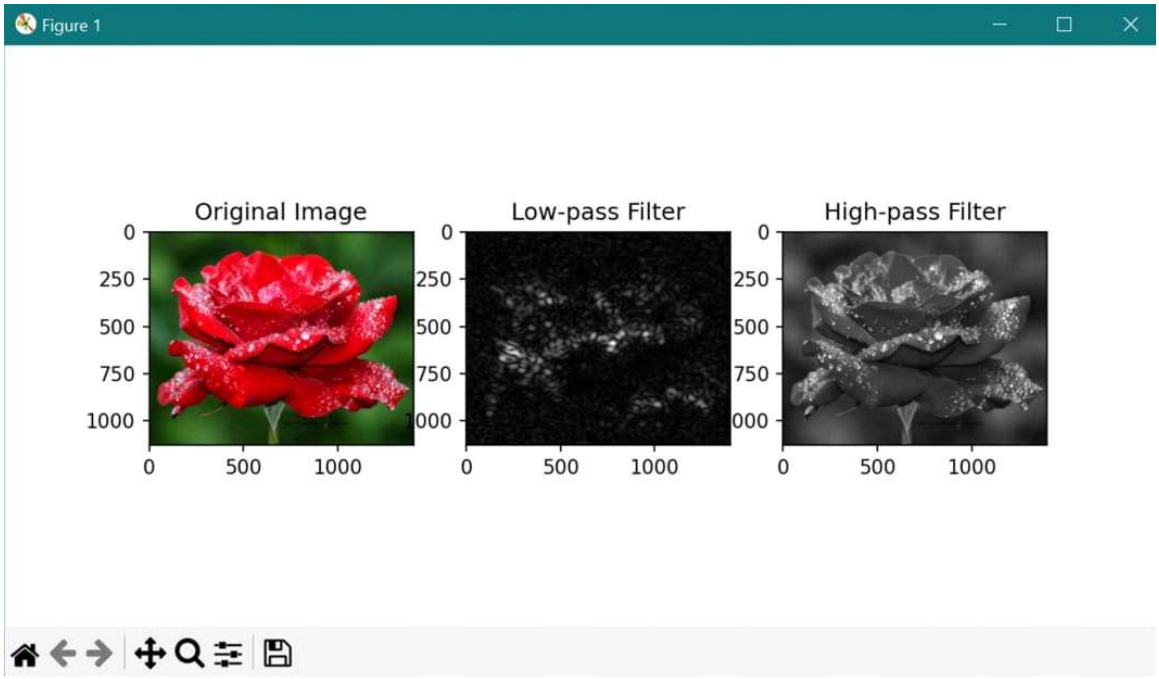


6.Low and High pass in frequency domain

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

def apply_low_pass_filter(image, cutoff_freq):

    rows, cols = image.shape

    crow, ccol = rows // 2, cols // 2

    dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)

    dft_shift = np.fft.fftshift(dft)

    mask = np.zeros((rows, cols, 2), np.uint8)

    mask[crow - cutoff_freq:crow + cutoff_freq, ccol - cutoff_freq:ccol +
```

```
cutoff_freq] =
 dft_shift_low = dft_shift * mask
 f_ishift = np.fft.ifftshift(dft_shift_low)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])
    return img_back
def apply_high_pass_filter(image, cutoff_freq):
    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2
    dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
     mask = np.ones((rows, cols, 2), np.uint8)
    mask[crow - cutoff_freq:crow + cutoff_freq, ccol - cutoff_freq:ccol +
cutoff_freq] = 0
   dft_shift_high = dft_shift * mask
    f_ishift = np.fft.ifftshift(dft_shift_high)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])
    return img_back
image = cv2.imread('C:/Users/sharana/OneDrive/Pictures/WIN_20230620_
15_39_26_Pro.jpg', cv2.IMREAD_GRAYSCALE)
cutoff_freq_low = 50
```

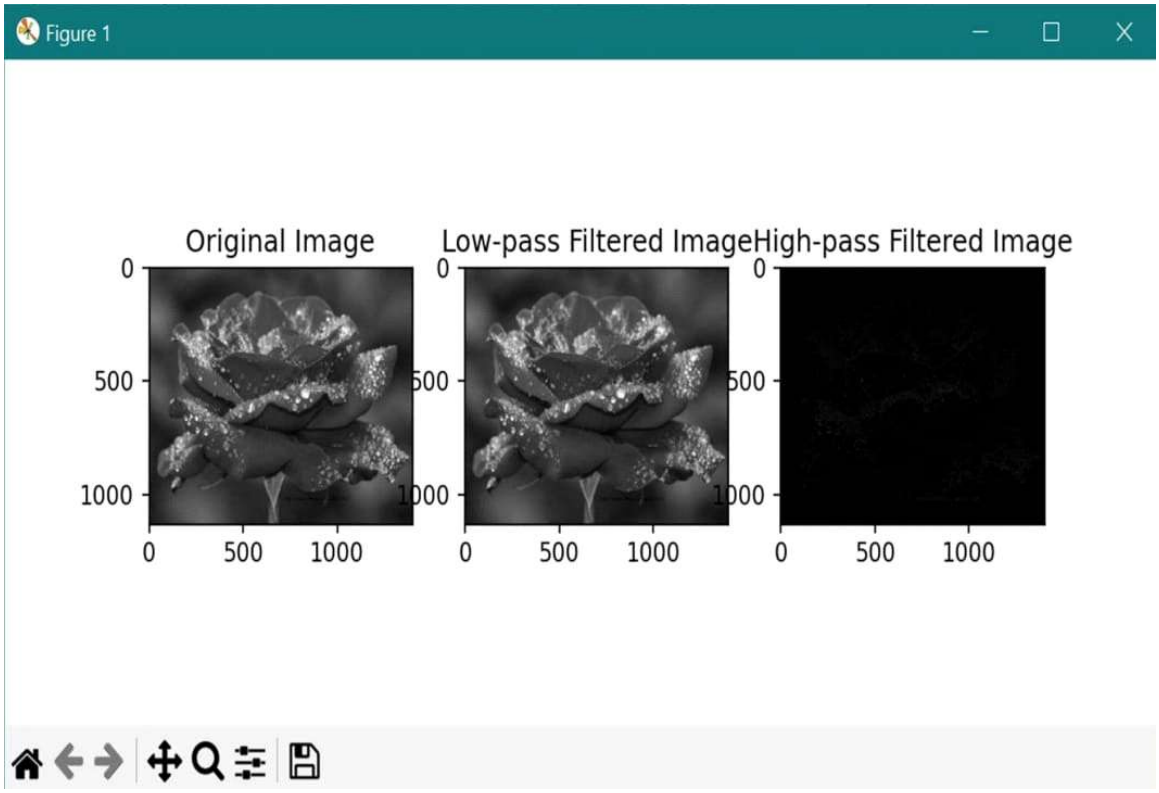
```
cutoff_freq_high = 50

smoothed_image_low = apply_low_pass_filter(image, cutoff_freq_low)

smoothed_image_high = apply_high_pass_filter(image, cutoff_freq_high)

plt.figure(figsize=(10, 5))

plt.subplot(1, 3, 1)

plt.imshow(image, cmap='gray')

plt.title('Original Image')

plt.axis('off')

plt.subplot(1, 3, 2)

plt.imshow(smoothed_image_low, cmap='gray')

plt.title('Low-Pass Filtered Image')

plt.axis('off')

plt.subplot(1, 3, 3)

plt.imshow(smoothed_image_high, cmap='gray')

plt.title('High-Pass Filtered Image')

plt.axis('off')

plt.tight_layout()

plt.show()
```



7.High and low pass in spatial domain

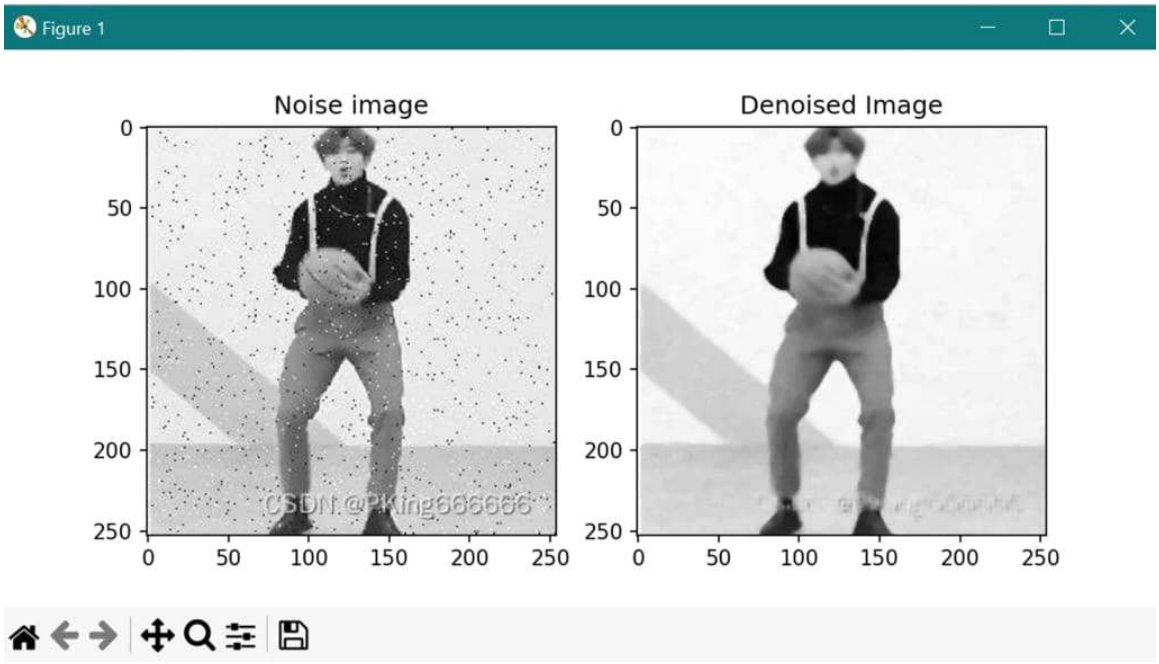
```
import cv2

import numpy as np

import matplotlib.pyplot as plt

image=cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf
cirtificate.jpg', cv2.IMREAD_GRAYSCALE)

kernel_size=3

kernel_lp=np.ones((kernel_size, kernel_size), np.float32)/(kernel_size*
kernel_size)

smoothed_image_lp=cv2.filter2D(image, -1, kernel_lp)

kernel_hp=np.array([[0, -1, 0],

[-1,4,-1],
```

```
[0,-1,0]])

smoothed_image_hp=cv2.filter2D(image, -1, kernel_hp)

plt.figure(figsize=(12,6))

plt.subplot(131),plt.imshow(image,cmap='gray'),plt.title('Original
Image')

plt.subplot(132),plt.imshow(smoothed_image_lp,cmap='gray'),plt.title('
Low pass Filtered Image')

plt.subplot(133),plt.imshow(smoothed_image_hp,cmap='gray'),plt.title(
'High pass Fitered Image')

plt.show()
```



8.salt and pepper using median filter

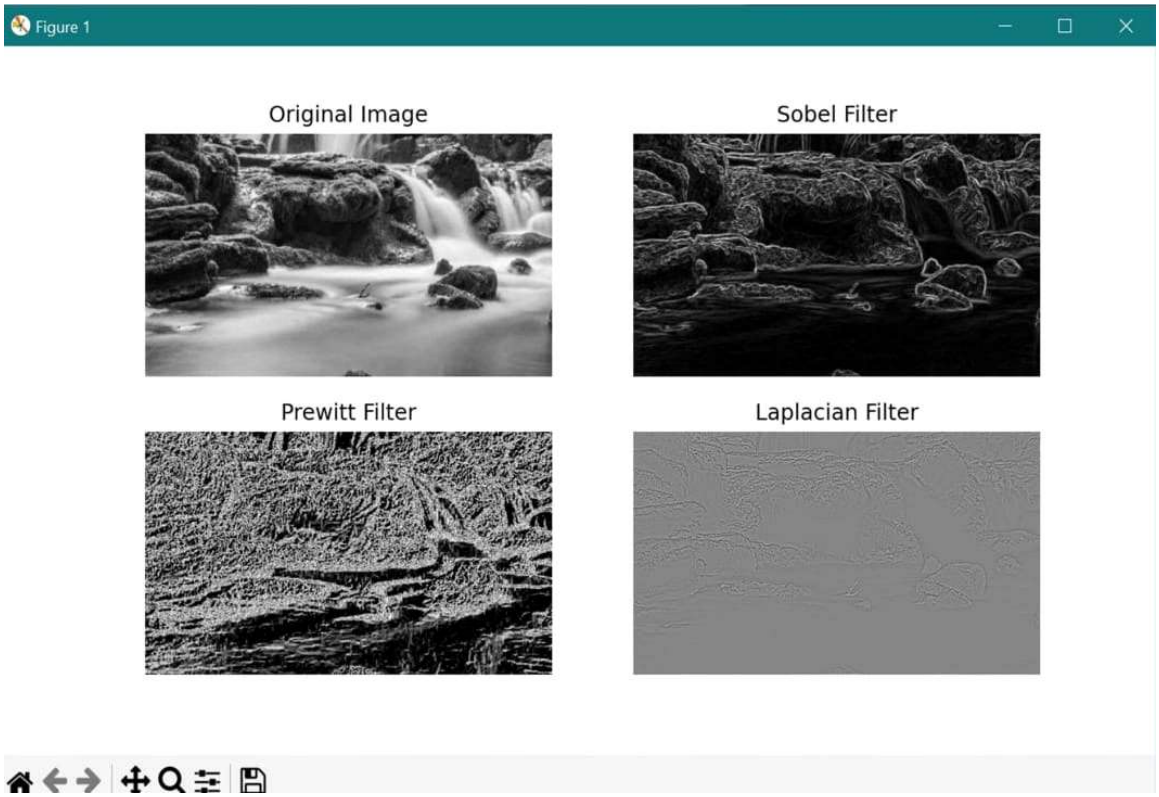
```
import cv2

def remove_salt_and_pepper_noise(image):

    return cv2.medianBlur(image, 5)

noisy_img = cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf cirtificate.jpg', 0)

denoised_img = remove_salt_and_pepper_noise(noisy_img)

cv2.imshow('Noisy Image', noisy_img)

cv2.imshow('Denoised Image', denoised_img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```



9.Sobel , Laplacian , prewitt filter

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('C:/Users/sharana/OneDrive/Pictures/WIN_
20230620_15_39_26_Pro.jpg', 0)

sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)

sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)

sobel = np.sqrt(sobel_x**2 + sobel_y**2)

prewitt_kernel_x = np.array([[-1, 0, 1],

                             [-1, 0, 1],

                             [-1, 0, 1]])

prewitt_kernel_y = np.array([[-1, -1, -1],

                             [0, 0, 0],

                             [1, 1, 1]])

prewitt_x = cv2.filter2D(image, -1, prewitt_kernel_x)

prewitt_y = cv2.filter2D(image, -1, prewitt_kernel_y)

prewitt = np.sqrt(prewitt_x**2 + prewitt_y**2)

laplacian = cv2.Laplacian(image, cv2.CV_64F)

plt.figure(figsize=(12, 8))

plt.subplot(231), plt.imshow(image, cmap='gray'), plt.title('Original
Image')
```

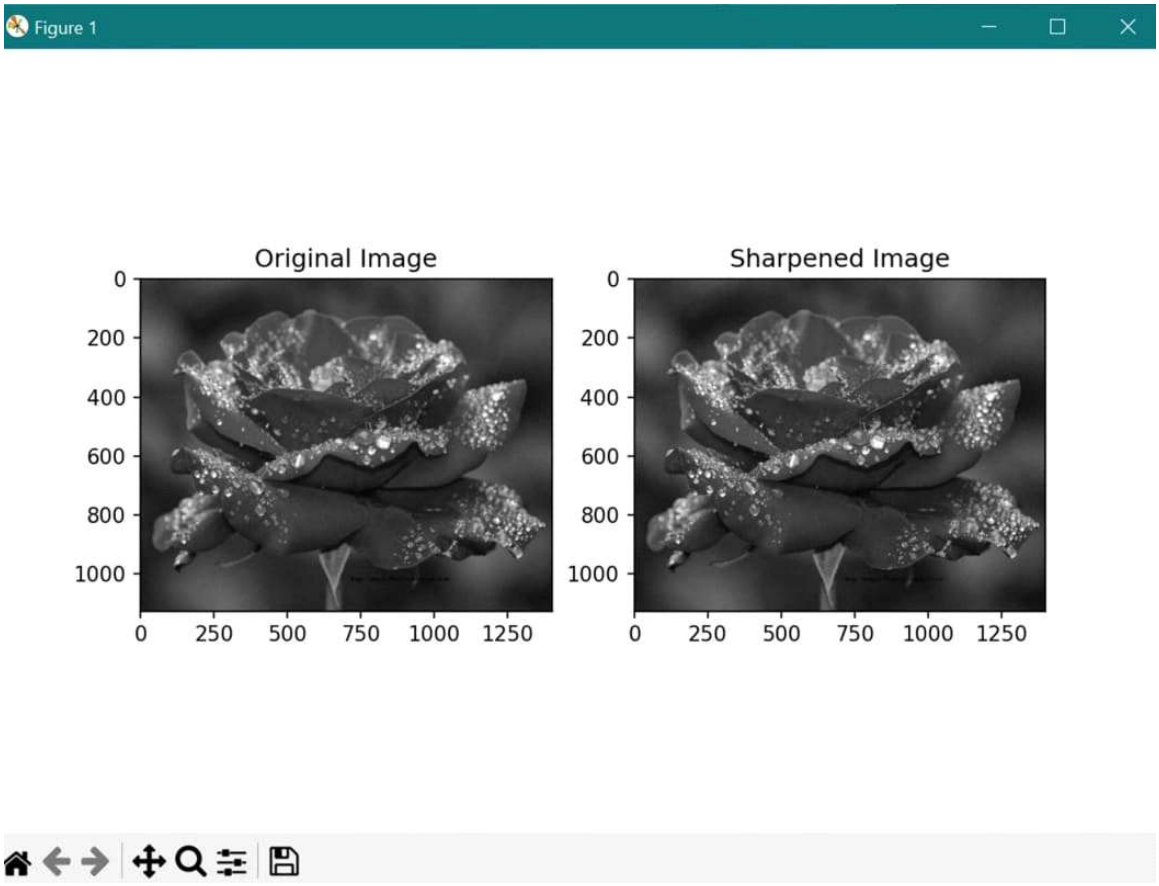
```
plt.subplot(232), plt.imshow(sobel, cmap='gray'), plt.title('Sobel Filter')

plt.subplot(233), plt.imshow(prewitt, cmap='gray'), plt.title('Prewitt
Filter')

plt.subplot(234), plt.imshow(laplacian, cmap='gray'),
plt.title('Laplacian Filter')

plt.show()
```



10.Sharpening image

```
import cv2

import numpy as np
```

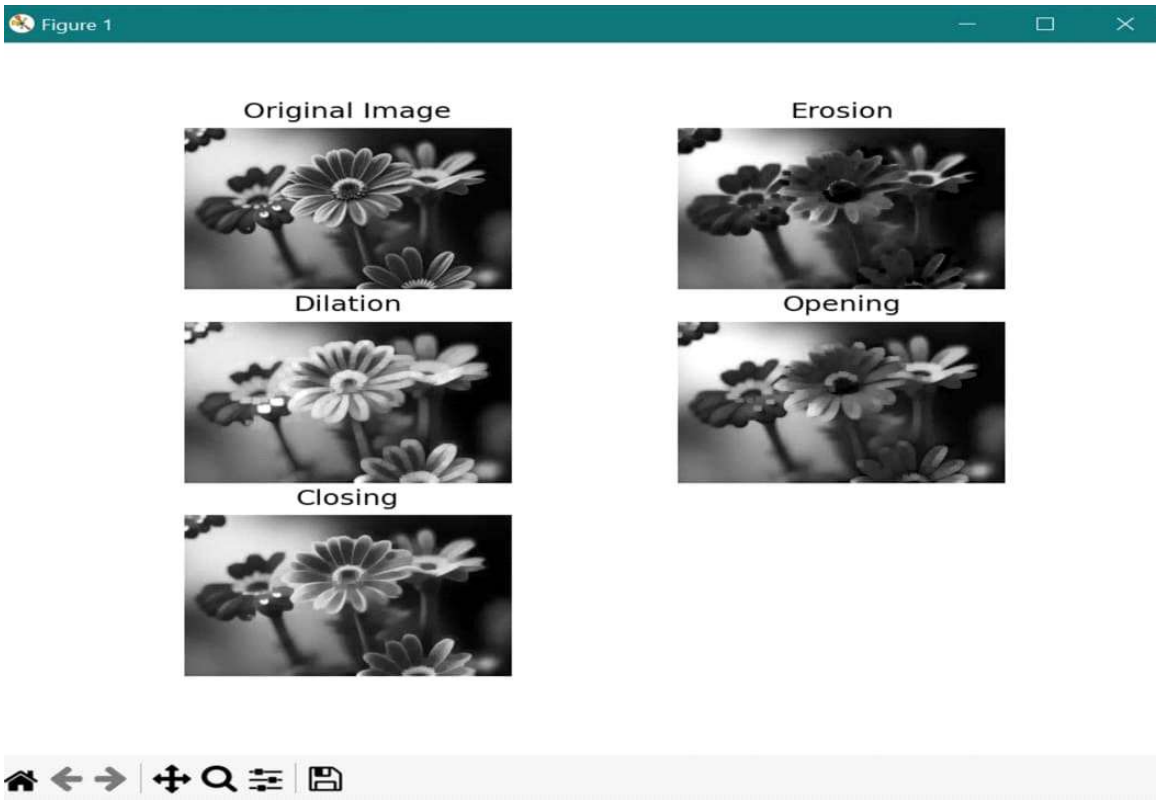
```
def sharpen_image(image):
        laplacian_filter = np.array([[0, 1, 0],
                                     [1, -4, 1],
                                     [0, 1, 0]])
        sharpened_image = cv2.filter2D(image, -1, laplacian_filter)
         sharpened_image = cv2.addWeighted(image, 1.0,
sharpened_image, -0.5, 0)
    return sharpened_image
image_path = 'C:/Users/sharana/OneDrive/Pictures/ysf cirtificate.jpg'
image = cv2.imread(image_path)
if image is None:
    print("Error: Image not found.")
else:
    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    sharpened_image = sharpen_image(grayscale_image
    cv2.imshow('Original Image', grayscale_image)
    cv2.imshow('Sharpened Image', sharpened_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```



11.Morphological filters

```
import cv2

import numpy as np

image = cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf cirtificate.jpg', 0)

kernel = np.ones((5, 5), np.uint8)

erosion = cv2.erode(image, kernel, iterations=1)

dilation = cv2.dilate(image, kernel, iterations=1)

opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
```

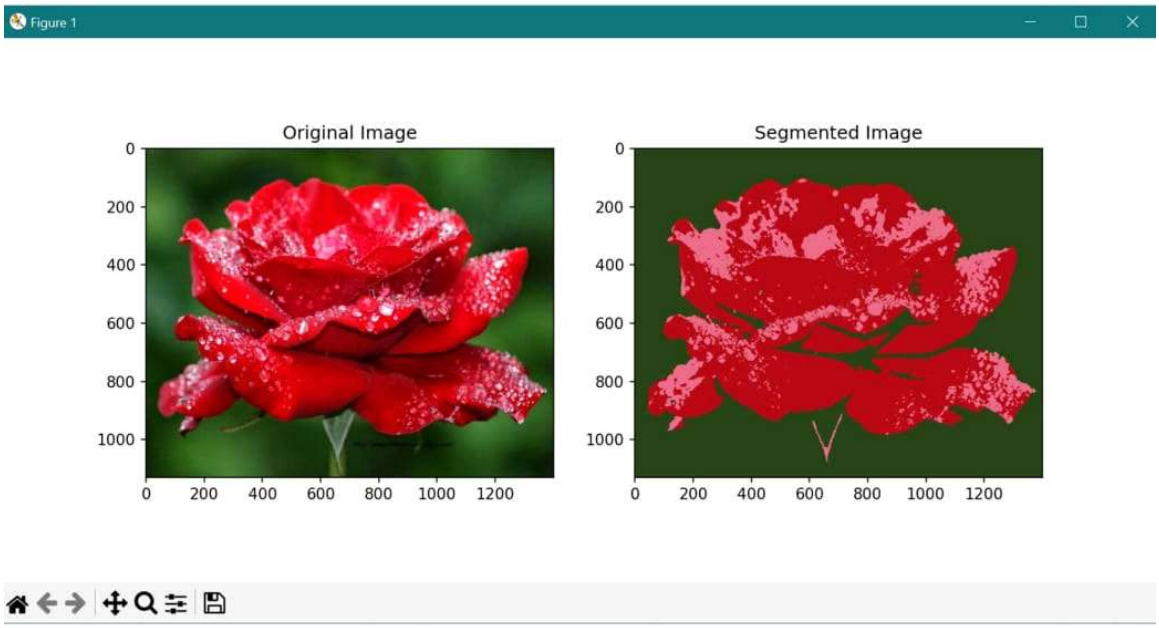
```
closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
cv2.imshow('Original Image', image)
cv2.imshow('Erosion', erosion)
cv2.imshow('Dilation', dilation)
```



12.Segmentation

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('C:/Users/sharana/OneDrive/Pictures/ysf cirtificate.jpg')
```

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

pixels = image.reshape((-1, 3))

pixels = np.float32(pixels)

criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)

k = 3

_, labels, centers = cv2.kmeans(pixels, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)

centers = np.uint8(centers)

segmented_image = centers[labels.flatten()]

segmented_image = segmented_image.reshape(image.shape)

plt.figure(figsize=(12, 6))

plt.subplot(121), plt.imshow(image), plt.title('Original Image')

plt.subplot(122), plt.imshow(segmented_image), plt.title('Segmented
Image')

plt.show()
```



13.Image Watermarking

```
import cv2
import numpy as np  def add_text_watermark(file,
out, mark, size, color, opacity, angle, space):
 # Load the image
image =
cv2.imread(file)
 (h, w) = image.shape[:2]

 # Create a blank image with the same
dimensions for the watermark   watermark =
np.zeros((h, w, 3), dtype="uint8")   # Set the
font, scale, and thickness for the watermark text
 font = cv2.FONT_HERSHEY_SIMPLEX

 scale = size / 100   color = tuple(int(color[i:i+2], 16) for
```

```
i in (1, 3, 5)) # Convert hex color to BGR   thickness =
int(size / 20)
 # Calculate the text size

 (text_w, text_h), baseline = cv2.getTextSize(mark,
font, scale, thickness)   text_h += baseline   # Create
a transparent overlay   overlay = watermark.copy()
 # Draw the watermark text repeatedly
on the overlay   for y in range(0, h,
text_h + space):   for x in range(0, w,
text_w + space):
 cv2.putText(overlay, mark, (x, y), font, scale, color,
thickness, cv2.LINE_AA)

 # Rotate the overlay if an angle
is specified   if angle != 0:
 M = cv2.getRotationMatrix2D((w // 2, h // 2), angle, 1)
 overlay = cv2.warpAffine(overlay, M, (w, h))
# Blend the overlay with the original image
cv2.addWeighted(overlay, opacity, image, 1 -
opacity, 0, image)
 # Save the watermarked image   cv2.imwrite(out,
image)   # Display the watermarked image
cv2.imshow('Watermarked Image', image) # Wait
for a key press and close the image window
cv2.waitKey(0)  cv2.destroyAllWindows() #
Example usage file = 'sunflower.jpg' out =
```

'watermarked_image.png' mark = 'UVCE' size =
80 color = '#ffffff' opacity = 0.2 angle = 30 space
= 40 add_text_watermark(file, out, mark, size,
color, opacity, angle, space)

14.Image restoration

```
import cv2

def restore_image(input_image_path, output_image_path, kernel_size=
3):

    input_image = cv2.imread(input_image_path)


    grayscale_image = cv2.cvtColor(input_image,
```

cv2.COLOR_BGR2GRAY

restored_image = cv2.medianBlur(grayscale_image, kernel_size)

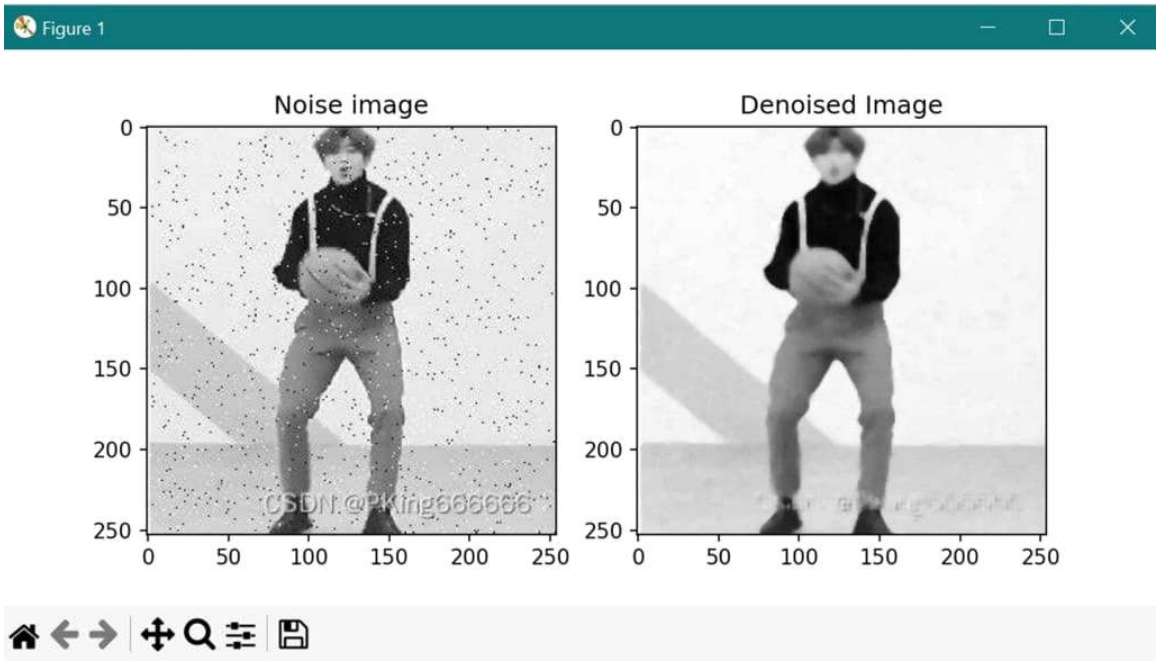
cv2.imwrite(output_image_path, restored_image)

input_image_path = 'noisy_image.jpeg'

output_image_path = 'restored_image.jpg'

kernel_size = 3 # Size of the median filter kernel (odd integer)

restore_image(input_image_path, output_image_path, kernel_size)



15.Block Truncation code

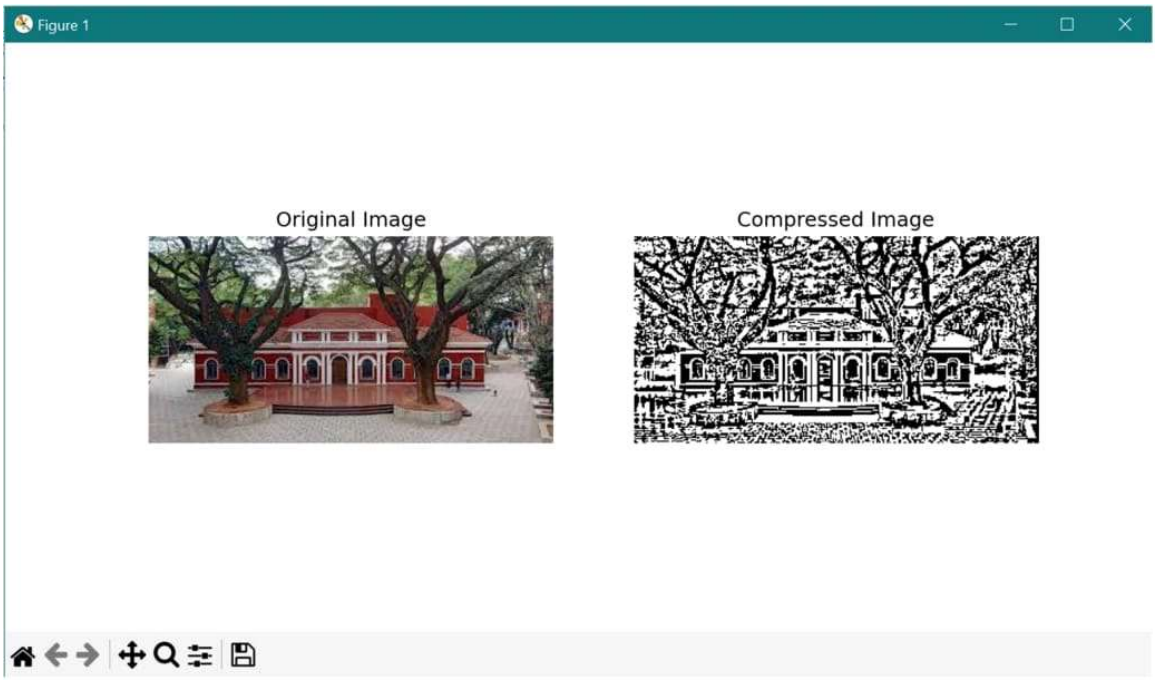
import cv2

import numpy as np

def block_truncation_coding(input_image_path, output_image_path, block_size=8)

input_image = cv2.imread(input_image_path,

```
cv2.IMREAD_GRAYSCALE)
    height, width = input_image.shape
    num_blocks_h = height // block_size
    num_blocks_w = width // block_size
    compressed_image = np.zeros((height, width), dtype=np.uint8)
    for i in range(num_blocks_h):
        for j in range(num_blocks_w):
            block = input_image[i*block_size:(i+1)*block_size,
j*block_size:(j+1)*block_size]
            block_mean = np.mean(block)
            block_std = np.std(block)
            threshold = block_mean
            compressed_block = np.where(block <= threshold, 0, 255)
            compressed_image[i*block_size:(i+1)*block_size,
j*block_size:(j+1)*block_size] = compressed_block
    cv2.imwrite(output_image_path, compressed_image)
input_image_path = 'uvce.jpeg'
output_image_path = 'compressed_image.jpg'
block_size = 8  # Size of the block (in pixels)
block_truncation_coding(input_image_path, output_image_path,
block_size)
```



16.Edge detection

```
import cv2
from matplotlib import pyplot as plt
image = cv2.imread('your_image.jpg', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(image, 100, 200)
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
```

```
plt.imshow(edges, cmap='gray')
plt.title('Edges')
plt.axis('off')
plt.show()
```