Image processing & Machine Vision

Assignment – 01

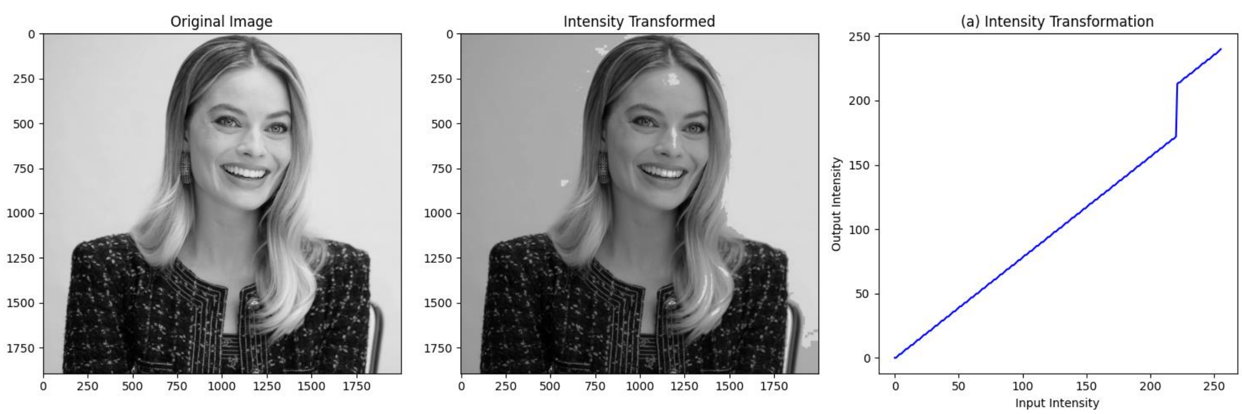
Name: JAS Sanjana

Reg No: D/ENG/22/0001/ET

Git Hub Link: https://github.com/SandaliSanjana/Image-Processing-Assingment

Question 01

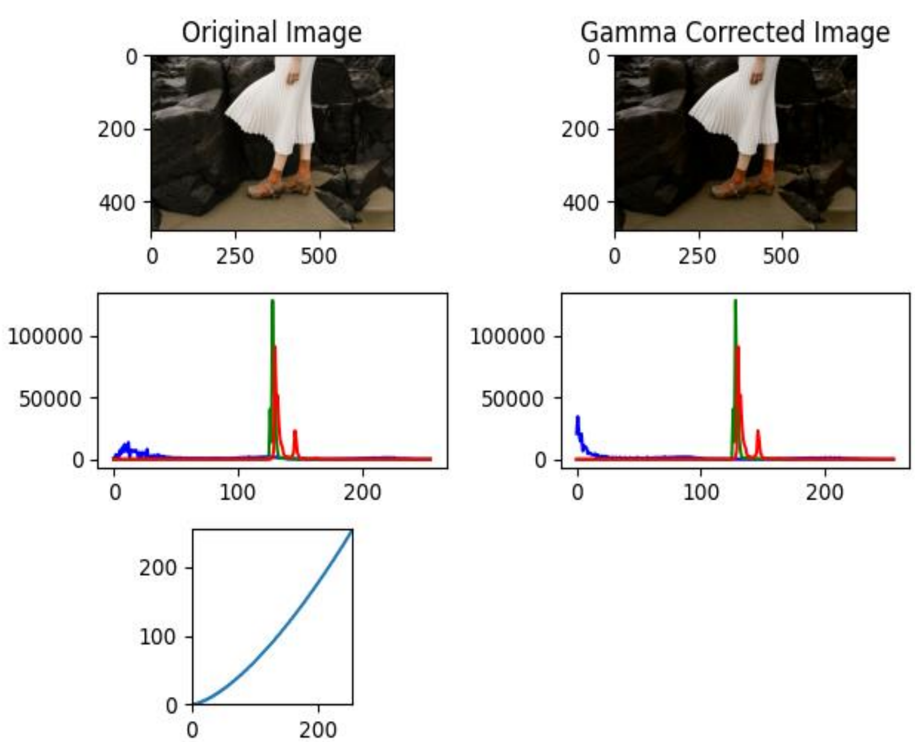
```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

im1 = cv.imread('D:/Images/margot_golden_gray.jpg')
assert im1 is not None

t = np.zeros(256,dtype=np.uint8)
t[0:221] = np.array([int(x*200/255) for x in range(221)])
t[221:256] = np.array([int(x*200/255+40) for x in range(221,256)])
im2 = t[im1]

fig,ax = plt.subplots(1,3, figsize = (15,5))
ax[0].imshow(im1,vmin=0,vmax=255,cmap='gray')
ax[0].set_title('Original Image')

ax[1].imshow(im2,vmin=0,vmax=255,cmap='gray')
ax[1].set_title('Intensity Transformed')
ax[2].plot(t, color='blue')
ax[2].set_title('(a) Intensity Transformation')
ax[2].set_xlabel('Input Intensity')
ax[2].set_ylabel('Output Intensity')
plt.tight_layout()
plt.show()
```



Intensity transformation adjusts image brightness, contrast, and other parameters. Intensity transformation is a fundamental method in image processing that modifies pixel values. I've provided the code and findings.

Question 02

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

img_orig = cv.imread('D:/Images/highlights_and_shadows.jpg')

img_lab = cv.cvtColor(img_orig, cv.COLOR_BGR2LAB)

L, a, b = cv.split(img_lab)

#apply gamma correction gamma = 1.5

table = np.array([(i/255.0)**(gamma)*255.0 for i in np.arange(0, 256)]).astype('uint8')

L_gamma = cv.LUT(L, table)

#gamma and lab merging img_lab_gamma = cv.merge([L_gamma, a, b])

img_gamma = cv.cvtColor(img_lab_gamma, cv.COLOR_LAB2BGR)

fig, axarr = plt.subplots(3, 2)

axarr[0, 0].imshow(cv.cvtColor(img_orig, cv.COLOR_BGR2RGB))

axarr[0, 0].set_title('Original Image')

axarr[0, 1].imshow(cv.cvtColor(img_gamma, cv.COLOR_BGR2RGB))

axarr[0, 1].set_title('Gamma Corrected Image')

color = ('b', 'g', 'r')

for i, c in enumerate(color):

    hist_orig = cv.calcHist([img_lab], [i], None, [256], [0, 256])

    axarr[1, 0].plot(hist_orig, color=c )

    hist_gamma = cv.calcHist([img_lab_gamma], [i], None, [256], [0, 256])

    axarr[1, 1].plot(hist_gamma, color=c)

axarr[2, 0].plot(table)

axarr[2, 0].set_xlim(0, 255)

axarr[2, 0].set_ylim(0, 255)

axarr[2, 0].set_aspect('equal')

axarr[2, 1].axis('off')

plt.tight_layout()

plt.show()
```

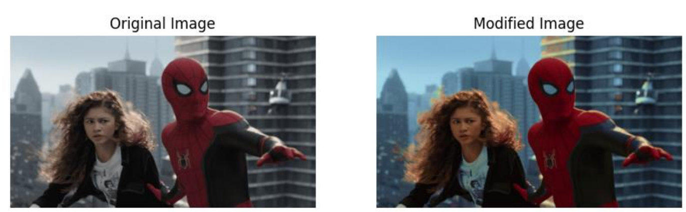


Histograms are visual representations of the distribution of pixel intensities in a picture. Original histogram color spaces correspond to pixel intensities in the L channel. Quantify the frequency of each intensity level in the image. Gamma correction modifies the relationship between input and output pixel intensities, which affects overall brightness and contrast.

Question 03

```python
def f(x, a, sigma):
    return np.minimum(x + a * 128 * np.exp(-(x - 128) ** 2 / (2 * sigma ** 2)), 255)

image = cv.imread("spider.png")
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

saturation_plane = hsv_image[:, :, 1]

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)

hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)
hue_plane, saturation_plane, value_plane = cv.split(cv.cvtColor(modified_image, cv.COLOR_BGR2HSV))

fig, ax = plt.subplots(1, 3, figsize=(10,5), sharey = True)

ax[0].imshow(hue_plane, cmap='gray')
ax[0].set_title('Hue Plane')
ax[0].axis('off')
ax[1].imshow(saturation_plane, cmap='gray')
ax[1].set_title('Saturation Plane')
ax[1].axis('off')

ax[2].imshow(value_plane, cmap='gray')
ax[2].set_title('Value Plane')
ax[2].axis('off')
```


```python
def f(x, a, sigma):
    return np.minimum(x + a * 128 * np.exp(-(x - 128) ** 2 / (2 * sigma ** 2)), 255)

image = cv.imread("spider.png")

hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
saturation_plane = hsv_image[:, :, 1]

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)

hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(cv.cvtColor(modified_image, cv.COLOR_BGR2RGB))
ax[1].set_title('Modified Image')
ax[1].axis('off')
plt.show()
```




```python
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
saturation_plane = hsv_image[:, :, 1]

a = 0.2
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)

hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(cv.cvtColor(modified_image, cv.COLOR_BGR2RGB))
ax[1].set_title(f'Modified Image (a={a})')
ax[1].axis('off')
plt.show()
```

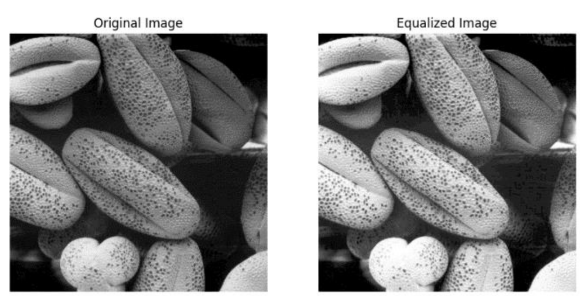
```python
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
hue_plane, saturation_plane, value_plane = cv.split(hsv_image)

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)

hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.merge([hue_plane, hsv_image[:, :, 1], value_plane])
final_modified_image = cv.cvtColor(modified_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(cv.cvtColor(final_modified_image, cv.COLOR_BGR2RGB))
ax[1].set_title(f'Final Modified Image (a={a})')
ax[1].axis('off')
plt.show()
```

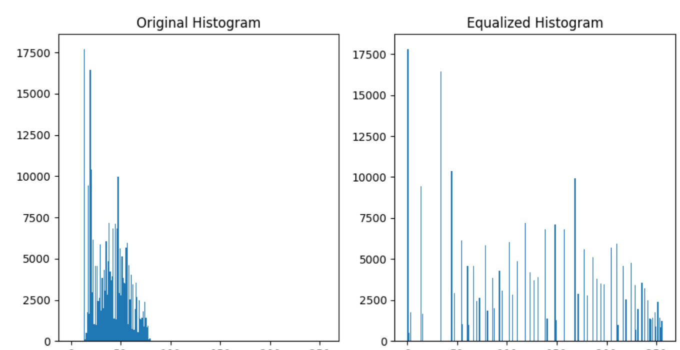

```python
hsv_image = cv.cvtColor(original_image, cv.COLOR_BGR2HSV)

saturation_plane = hsv_image[:, :, 1]

vibrance_factor = 1.5
vibrance_enhanced_image = original_image.copy()
vibrance_enhanced_image[:, :, 1] = np.clip(vibrance_factor * saturation_plane, 0, 255)

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)
hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)

intensity_transformed_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 3, figsize=(10,5), sharey = True)

ax[0].imshow(cv.cvtColor(original_image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(cv.cvtColor(vibrance_enhanced_image, cv.COLOR_BGR2RGB))
ax[1].set_title('Vibrance Enhanced')
ax[1].axis('off')

ax[2].imshow(cv.cvtColor(intensity_transformed_image, cv.COLOR_BGR2RGB)),
ax[2].set_title('Intensity Transformed')
ax[2].axis('off')
plt.show()
```

This topic focuses on image improvement, which involves applying intensity transformations to saturation, HUE, and HSV planes to increase vibrance. We focus on improving and bringing out more vibrant colors throughout these three planes. The intensity transformation includes adjusting a parameter 'a' to obtain a desirable vibrance boost.

Question 04

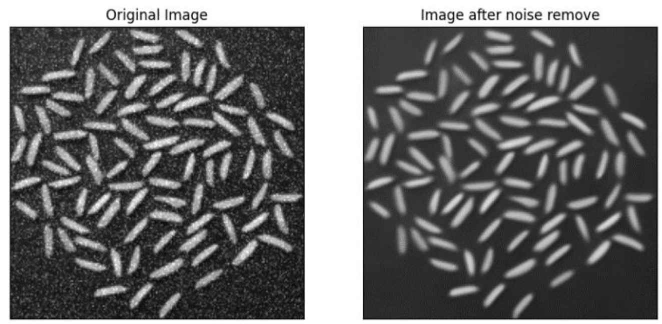
```python
im = cv.imread('shells.tif', cv.IMREAD_GRAYSCALE)

im_equalized = cv.equalizeHist(im)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Histogram')
plt.hist(im.flatten(), bins=256, range=[0, 256])

plt.subplot(1, 2, 2)
plt.title('Equalized Histogram')
plt.hist(im_equalized.flatten(), bins=256, range=[0, 256])

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(im, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(im_equalized, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.show()
```



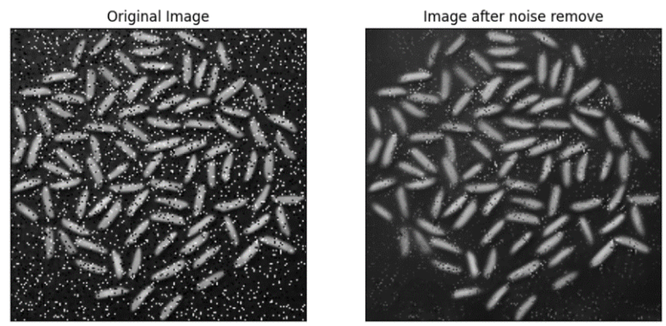


Question 05

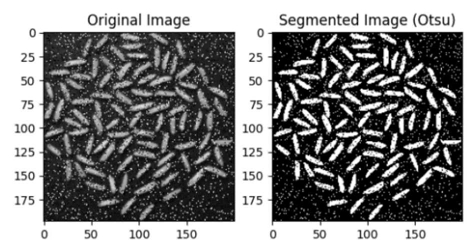
```python
image = cv2.imread('rice_gaussian_noise.png')
noise_removed_img = cv2.fastNlMeansDenoising(image, None,40,7,21)

images = [image, noise_removed_img]

titles = ['Original Image', 'Image after noise remove']

plt.figure(figsize=(10, 8))
for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```



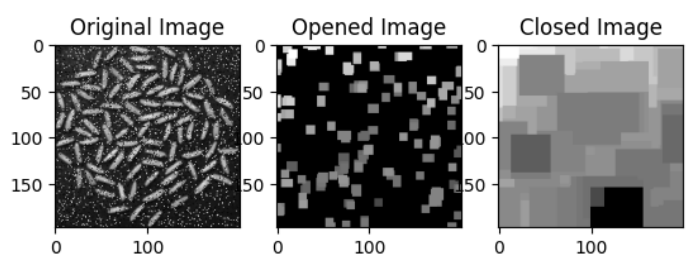

```python
image = cv2.imread('rice_salt_pepper_noise.png')
noise_removed_img = cv2.fastNlMeansDenoising(image, None,40,7,21)

images = [image, noise_removed_img]

titles = ['Original Image', 'Image after noise remove']

plt.figure(figsize=(10, 8))
for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```




```python
image = cv.imread("rice_salt_pepper_noise.png", cv.IMREAD_GRAYSCALE)

_, binary_image = cv.threshold(image, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

plt.subplot(121), plt.imshow(image, cmap='gray'), plt.title('Original Image')
plt.subplot(122), plt.imshow(binary_image, cmap='gray'), plt.title('Segmented Image (Otsu)')
plt.show()
```




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

image = cv.imread('rice_salt_pepper_noise.png', cv.IMREAD_GRAYSCALE)

kernel_open = np.ones((5, 5), np.uint8)
image_opened = cv.morphologyEx(image, cv.MORPH_OPEN, kernel_open, iterations=2)

kernel_close = np.ones((5, 5), np.uint8)
image_closed = cv.morphologyEx(image_opened, cv.MORPH_CLOSE, kernel_close, iterations=10)

plt.subplot(131), plt.imshow(image, cmap='gray'), plt.title('Original Image')
plt.subplot(132), plt.imshow(image_opened, cmap='gray'), plt.title('Opened Image')
plt.subplot(133), plt.imshow(image_closed, cmap='gray'), plt.title('Closed Image')
plt.show()
```



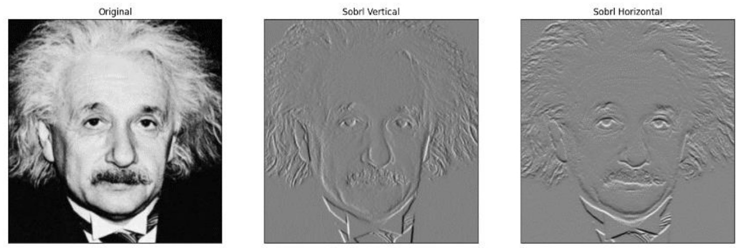

```python
im = cv.imread('rice_gaussian_noise.png', cv.IMREAD_GRAYSCALE)

denoised_im = cv.fastNlMeansDenoising(im, None, h=28, searchWindowSize=10)

_, segmented_image = cv.threshold(denoised_im, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

closed_image = cv.morphologyEx(segmented_image, cv.MORPH_CLOSE, kernel)

opened_image = cv.morphologyEx(closed_image, cv.MORPH_OPEN, kernel)

num_labels, labels = cv.connectedComponents(opened_image)

num_rice_grains = num_labels - 1

print('Number of rice grains : ', num_rice_grains)
```

```
Number of rice grains :  68
```

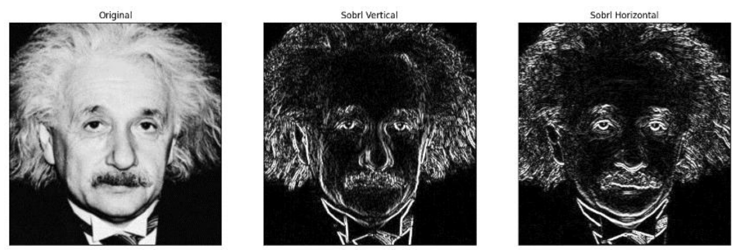
Otsu's approach was used to segment the images, separating rice grains from the backdrop. Morphological operations refine segmentation by removing minor artifacts and filling in gaps.

Question 06

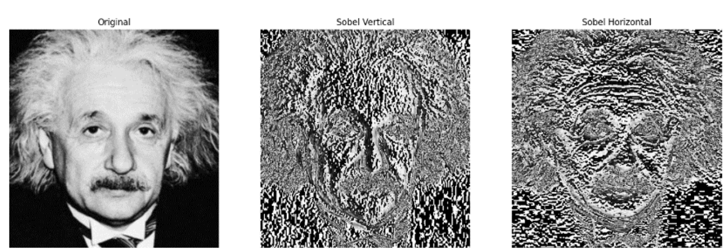
```
img = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE)

kernal = np.ones((11,11),np.float32)/121
imgc =cv. filter2D(img,-1,kernal)

sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.float32)

sobel_x = cv.filter2D(img, cv.CV_64F, sobel_kernel_x)
sobel_y = cv.filter2D(img, cv.CV_64F, sobel_kernel_y)


fig,axes (function) imshow: Any rex='all', sharey='all', figsize=(18,18))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])

axes[1].imshow(sobel_x, cmap='gray')
axes[1].set_title('Sobrl Vertical')
axes[1].set_xticks([]), axes[1].set_yticks([])

axes[2].imshow(sobel_y, cmap='gray')
axes[2].set_title('Sobrl Horizontal')
axes[2].set_xticks([]), axes[0].set_yticks([])
```




```
sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.float32)

sobel_x = cv.filter2D(image, cv.CV_64F, sobel_kernel_x)
sobel_y = cv.filter2D(image, cv.CV_64F, sobel_kernel_y)

sobel_x = cv.convertScaleAbs(sobel_x)
sobel_y = cv.convertScaleAbs(sobel_y)

fig,axes = plt.subplots(1,3, sharex='all', sharey='all', figsize=(18,18))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])

axes[1].imshow(sobel_x, cmap='gray')
axes[1].set_title('Sobrl Vertical')
axes[1].set_xticks([]), axes[1].set_yticks([])

axes[2].imshow(sobel_y, cmap='gray')
axes[2].set_title('Sobrl Horizontal')
axes[2].set_xticks([]), axes[0].set_yticks([])
```




```
def sobel_filter(image):

    sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    sobel_x = convolve2d(image, sobel_kernel_x)
    sobel_y = convolve2d(image, sobel_kernel_y)

    gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    sobel_x = np.abs(sobel_x).astype(np.uint8)
    sobel_y = np.abs(sobel_y).astype(np.uint8)
    gradient_magnitude = gradient_magnitude.astype(np.uint8)

    return sobel_x, sobel_y, gradient_magnitude

def convolve2d(image, kernel):
    height, width = image.shape
    k_height, k_width = kernel.shape

    pad_height = k_height // 2
    pad_width = k_width // 2
    padded_image = np.pad(image, ((pad_height, pad_height), (pad_width, pad_width)), mode='edge')

    result = np.zeros_like(image)
    for i in range(height):
        for j in range(width):
            result[i, j] = np.sum(padded_image[i:i+k_height, j:j+k_width] * kernel)

    return result
image = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE)

if len(image.shape) > 2:
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

sobel_x, sobel_y, gradient_magnitude = sobel_filter(image)
```

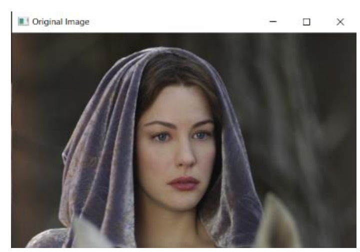


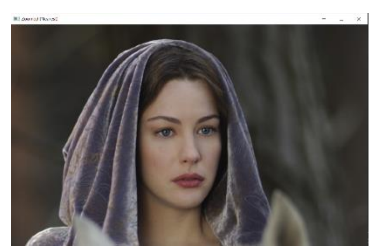
Sobel filtering is a significant approach in image processing that uses intensity fluctuations to find edges. Figure 6 illustrates three approaches. The filter2D function allows for effective and fast Sobel filtering. Second, a custom Sobel filter implementation allows for interactive exploration of the underlying computations.

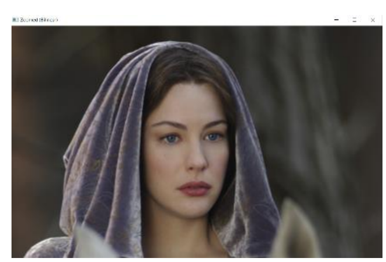
Question 07

```python
def zoom_image(image, zoom_factor, interpolation='nearest'):
    height, width = image.shape[:2]

    new_height = int(height * zoom_factor)
    new_width = int(width * zoom_factor)

    if interpolation == 'nearest':
        interpolation_method = cv2.INTER_NEAREST
    elif interpolation == 'bilinear':
        interpolation_method = cv2.INTER_LINEAR
    else:
        raise ValueError("Invalid interpolation method. Use 'nearest' or 'bilinear'.")

    zoomed_image = cv2.resize(image, (new_width, new_height), interpolation=interpolation_method)

    return zoomed_image

def compute_normalized_ssd(image1, image2):
    image1_resized = cv2.resize(image1, (image2.shape[1], image2.shape[0]))

    ssd = np.sum((image1_resized - image2)**2)
    normalized_ssd = ssd / (np.prod(image1_resized.shape) * np.max(image1_resized)**2)
    return normalized_ssd

image_path = 'im01small.png'
original_image = cv2.imread(image_path)

zoomed_nearest = zoom_image(original_image, 4, interpolation='nearest')
zoomed_bilinear = zoom_image(original_image, 4, interpolation='bilinear')

ssd_nearest = compute_normalized_ssd(original_image, zoomed_nearest)
print(f"Normalized SSD (Nearest): {ssd_nearest}")

ssd_bilinear = compute_normalized_ssd(original_image, zoomed_bilinear)
print(f"Normalized SSD (Bilinear): {ssd_bilinear}")
```

```python
    if interpolation == 'nearest':
        interpolation_method = cv2.INTER_NEAREST
    elif interpolation == 'bilinear':
        interpolation_method = cv2.INTER_LINEAR
    else:
        raise ValueError("Invalid interpolation method. Use 'nearest' or 'bilinear'.")

    zoomed_image = cv2.resize(image, (new_width, new_height), interpolation=interpolation_method)

    return zoomed_image

def compute_normalized_ssd(image1, image2):
    image1_resized = cv2.resize(image1, (image2.shape[1], image2.shape[0]))

    ssd = np.sum((image1_resized - image2)**2)
    normalized_ssd = ssd / (np.prod(image1_resized.shape) * np.max(image1_resized)**2)
    return normalized_ssd

image_path = 'im02small.png'
original_image = cv2.imread(image_path)

zoomed_nearest = zoom_image(original_image, 2, interpolation='nearest')
zoomed_bilinear = zoom_image(original_image, 2, interpolation='bilinear')

ssd_nearest = compute_normalized_ssd(original_image, zoomed_nearest)
print(f"Normalized SSD (Nearest): {ssd_nearest}")

ssd_bilinear = compute_normalized_ssd(original_image, zoomed_bilinear)
print(f"Normalized SSD (Bilinear): {ssd_bilinear}")

cv2.imshow('Original Image', original_image)
cv2.imshow('Zoomed (Nearest)', zoomed_nearest)
cv2.imshow('Zoomed (Bilinear)', zoomed_bilinear)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

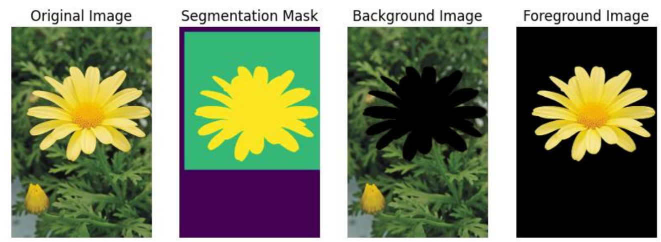






Question 08

```python
image = cv.imread('daisy.jpg')

mask = np.zeros(image.shape[:2], np.uint8)
background = np.zeros((1,65), np.float64)

rect = (20, 20, 550, 550)

cv.grabCut(image, mask, rect, None, None, 5, cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask == cv.GC_FGD) | (mask == cv.GC_PR_FGD), 1, 0).astype('uint8')

foreground = cv.bitwise_and(image, image, mask=mask2)
background = cv.bitwise_and(image, image, mask=1 - mask2)

segmentation_mask = np.where(mask2[:, :, np.newaxis] == 1, 255, 0).astype('uint8')

fig, ax = plt.subplots(1, 4, figsize=(10,10), sharey = True)

ax[0].imshow(image[:,:,::-1]),
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(mask)
ax[1].set_title('Segmentation Mask')
ax[1].axis('off')

ax[2].imshow(background[:,:,::-1]),
ax[2].set_title('Background Image')
ax[2].axis('off')

ax[3].imshow(foreground[:,:,::-1])
ax[3].set_title('Foreground Image')
ax[3].axis('off')
```




```python
mask = np.zeros(image.shape[:2], np.uint8)
background = np.zeros((1,65), np.float64)

rect = (20, 20, 550, 550)

cv.grabCut(image, mask, rect, None, None, 5, cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask == cv.GC_FGD) | (mask == cv.GC_PR_FGD), 1, 0).astype('uint8')

foreground = cv.bitwise_and(image, image, mask=mask2)
background = cv.bitwise_and(image, image, mask=1 - mask2)
segmentation_mask = np.where(mask2[:, :, np.newaxis] == 1, 255, 0).astype('uint8')

blurred_bg = cv.GaussianBlur(background, (31, 31), 0)
enhanced_img = cv.addWeighted(foreground, 1, blurred_bg, 0.8, 0)

fig, ax = plt.subplots(1, 2, figsize=(12,6), sharey = True)

ax[0].imshow(image[:,:,::-1])
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(enhanced_img[:,:,::-1])
ax[1].set_title('Enhanced Image')
ax[1].axis('off')
```

c. The darker backdrop in the improved image is primarily due to a Gaussian blur applied to the background, extending beyond the flower's boundary. Grab Cut divides the image into foreground (flower) and background. The background is then smoothed using a Gaussian blur with a kernel size of (15, 15). The smoothing effect averages pixel values, making the backdrop appear darker. To improve the image, combine the sharp foreground and blurring backdrop. Variables like kernel size can adjust blurring and darkness in the backdrop.