EN3160 Assignment 1

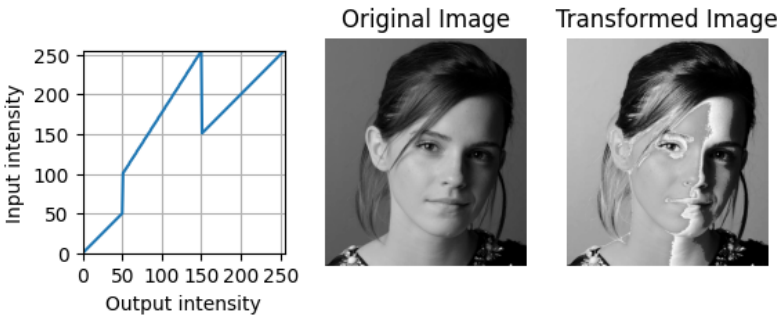
Intensity Transformations and Neighborhood Filtering

Index no: 200417M, Name: NAZAR F.S.

GitHub repository link: https://github.com/Saeedha-N/EN3160-Image-Processing-and-Machine-Vision.git

Question 1

In [12]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np

c = np.array([(50, 50), (50, 100), (150, 255), (150, 150), (255,255)]) # (x, y) coordinates of control points

t1 = np.linspace(0, c[0, 1], c[0, 0] + 1 - 0).astype('uint8') # 0 to 50, 51 elements
t2 = np.linspace(c[1, 1] + 1, c[2,1], c[2, 0] - c[1, 0]).astype('uint8') # 101 to 255, 100 elements
t3 = np.linspace(c[3, 1] + 1, c[4, 1], c[4, 0] - c[3, 0]).astype('uint8') # 151 to 255, 105 elements
transform = np.concatenate((t1, t2), axis=0).astype('uint8') # concatenate arrays
transform = np.concatenate((transform, t3), axis=0).astype('uint8') # concatenate arrays

fig, ax = plt.subplots(1,3)

# plot intensity transformation function
ax[0].plot(transform), ax[0].set_xlabel('Output intensity'), ax[0].set_ylabel('Input intensity'), ax[0].set_xlim([0, 255]), ax[0].set_ylim([0,

img_original = cv.imread('emma.jpg', cv.IMREAD_GRAYSCALE) # read image in grayscale
ax[1].imshow(img_original, cmap='gray'), ax[1].set_title('Original Image'), ax[1].axis('off')

img_transformed = cv.LUT(img_original, transform) # apply intensity transformation function
ax[2].imshow(img_transformed, cmap='gray') # show transformed image
ax[2].set_title('Transformed Image'), ax[2].axis('off')

plt.show() # show plot
```

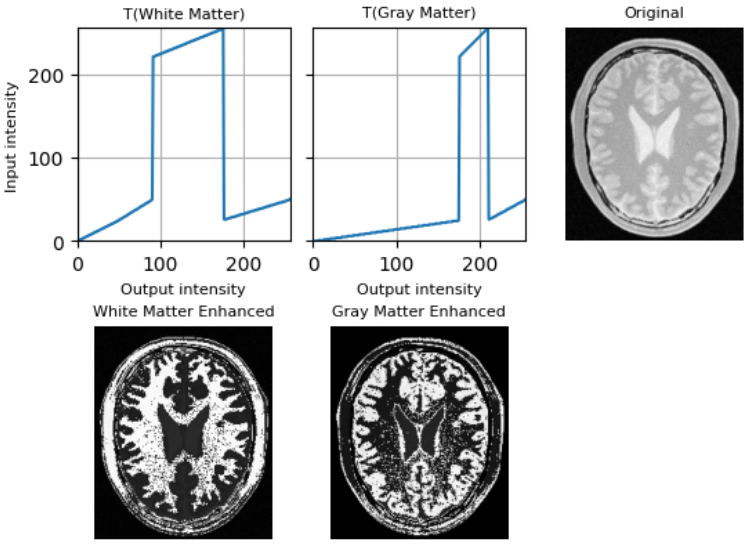


The middle-valued original intensities are enhanced i.e. increased upon intensity while the rest remain the same.

Question 2

In [76]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np

c_white = np.array([(50,25), (90,50), (90,220), (175,255), (175,25), (255,50)]) # control points in white matter
c_gray = np.array([(175,25), (175,50), (175,220), (210,255), (210,25), (255,50)]) # control points in gray matter

t1_white = np.linspace(0, c_white[0, 1], c_white[0, 0] + 1 - 0).astype('uint8')
t2_white = np.linspace(c_white[0, 1] + 1, c_white[1,1], c_white[1, 0] - c_white[0, 0]).astype('uint8')
t3_white = np.linspace(c_white[2, 1] + 1, c_white[3, 1], c_white[3, 0] - c_white[2, 0]).astype('uint8')
t4_white = np.linspace(c_white[4, 1] + 1, c_white[5, 1], c_white[5, 0] - c_white[4, 0]).astype('uint8')
white_transform = np.concatenate((t1_white, t2_white), axis=0).astype('uint8') # concatenate arrays
white_transform = np.concatenate((white_transform, t3_white), axis=0).astype('uint8')
white_transform = np.concatenate((white_transform, t4_white), axis=0).astype('uint8')

t1_gray = np.linspace(0, c_gray[0, 1], c_gray[0, 0] + 1 - 0).astype('uint8')
t2_gray = np.linspace(c_gray[0, 1] + 1, c_gray[1,1], c_gray[1, 0] - c_gray[0, 0]).astype('uint8')
t3_gray = np.linspace(c_gray[2, 1] + 1, c_gray[3, 1], c_gray[3, 0] - c_gray[2, 0]).astype('uint8')
t4_gray = np.linspace(c_gray[4, 1] + 1, c_gray[5, 1], c_gray[5, 0] - c_gray[4, 0]).astype('uint8')
gray_transform = np.concatenate((t1_gray, t2_gray), axis=0).astype('uint8') # concatenate arrays
gray_transform = np.concatenate((gray_transform, t3_gray), axis=0).astype('uint8')
gray_transform = np.concatenate((gray_transform, t4_gray), axis=0).astype('uint8')

fig, ax = plt.subplots(2,3), plt.subplots_adjust(wspace=0.1), plt.subplots_adjust(hspace=0.4)

ax[0,0].plot(white_transform) # plot intensity transformation function of white matter
ax[0,0].set_xlabel('Output intensity', fontsize=8), ax[0,0].set_ylabel('Input intensity', fontsize=8), ax[0,0].set_title('T(White Matter)', fon

ax[0,1].plot(gray_transform) # plot intensity transformation function of gray matter
ax[0,1].set_xlabel('Output intensity', fontsize=8), ax[0,1].set_title('T(Gray Matter)', fontsize=8), ax[0,1].set_xlim([0, 255]), ax[0,1].set_yl

img_orig = cv.imread('brain.png', cv.IMREAD_GRAYSCALE) # read image as grayscale
ax[0,2].imshow(img_orig, cmap='gray'), ax[0,2].set_title('Original', fontsize=8), ax[0,2].axis('off')
```

```
img_white = cv.LUT(img_orig, white_transform)
ax[1,0].imshow(img_white, cmap='gray') # show white matter enhanced image
ax[1,0].set_title('White Matter Enhanced', fontsize=8), ax[1,0].axis('off')

img_gray = cv.LUT(img_orig, gray_transform)
ax[1,1].imshow(img_gray, cmap='gray') # show gray matter enhanced image
ax[1,1].set_title('Gray Matter Enhanced', fontsize=8), ax[1,1].axis('off')

ax[1, 2].remove() # remove empty subplot
plt.show()
```

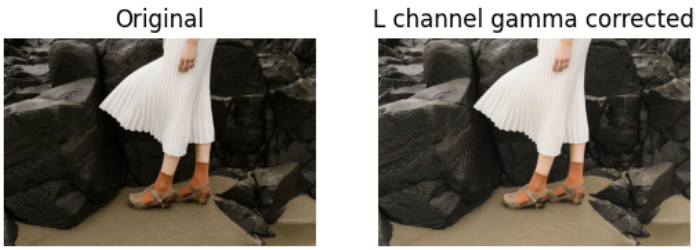


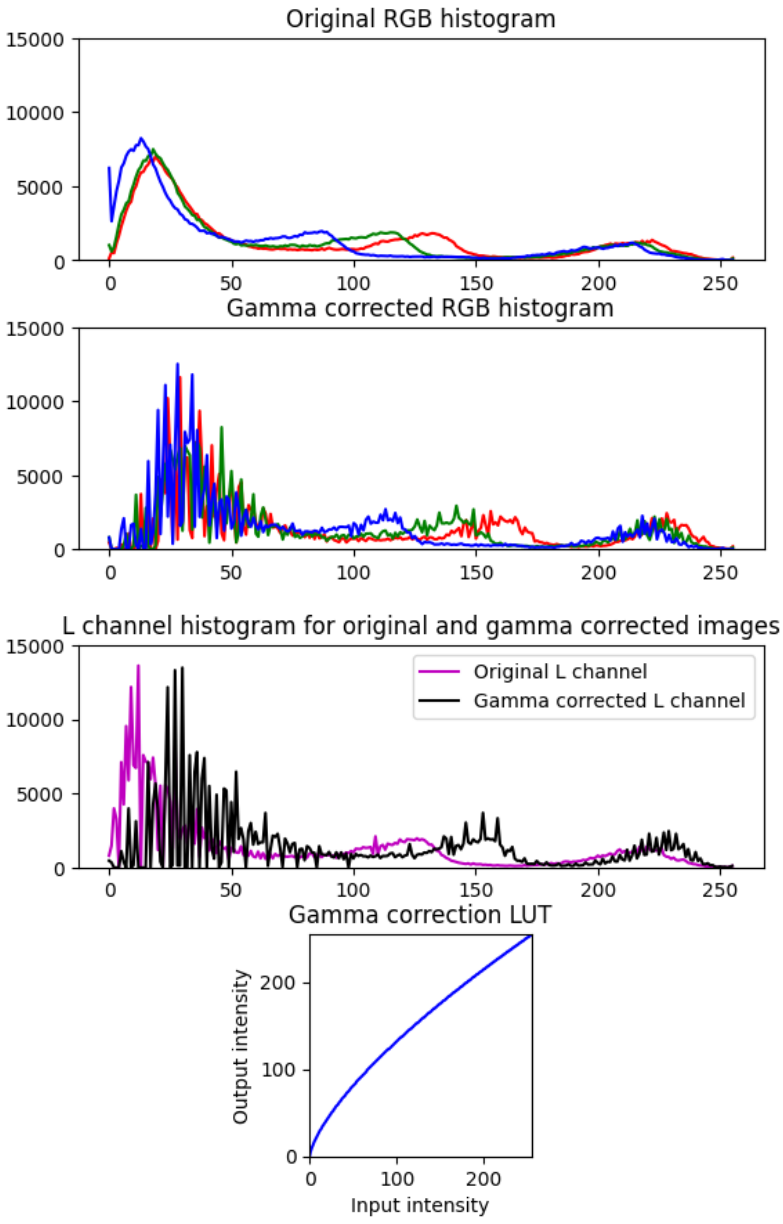
The intensity transformations accentuates white matter and gray matter, thus enhancing the visibility of white matter and gray matter within the image. By adjusting the control points and intensity mapping functions, it selectively increases the intensity the required tissue and darkens the rest.

Question 3

In [59]:

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

image_original = cv.imread('highlights_and_shadows.jpg', cv.IMREAD_COLOR) # Load the image in color
image_lab = cv.cvtColor(image_original, cv.COLOR_BGR2LAB) # Convert the image to LAB color space
L_channel = image_lab[:, :, 0] # Extract the L* channel

gamma = 0.7
table = np.array([((i / 255.0) ** gamma) * 255 for i in range(0,256)], dtype=np.uint8) # Create a lookup table for gamma correction
L_gamma = cv.LUT(L_channel, table) # Apply gamma correction using LUT
image_lab[:, :, 0] = L_gamma # Replace the original L* channel with the corrected one
image_gamma = cv.cvtColor(image_lab, cv.COLOR_LAB2RGB) # Convert the LAB image back to RGB color space
image_original = cv.cvtColor(image_original, cv.COLOR_BGR2RGB) # Convert the original image to RGB color space

# Display the original and corrected images using matplotlib
f, axarr = plt.subplots(1, 2)
axarr[0].imshow(image_original), axarr[0].set_title('Original'), axarr[0].axis('off'), axarr[1].imshow(image_gamma), axarr[1].set_title('L chan
plt.show()

# Plot histograms in RGB
f, axarr = plt.subplots(2, 1)
plt.subplots_adjust(hspace=0.3)
color = ('r', 'g', 'b')
for i, c in enumerate(color):
    hist_orig = cv.calcHist([image_original], [i], None, [256], [0, 256])
    axarr[0].set_ylim(0, 15000), axarr[0].plot(hist_orig, color=c), axarr[0].set_title('Original RGB histogram')
    hist_gamma = cv.calcHist([image_gamma], [i], None, [256], [0, 256])
    axarr[1].set_ylim(0, 15000), axarr[1].plot(hist_gamma, color=c), axarr[1].set_title('Gamma corrected RGB histogram')

# Plot histograms in L*a*b*
image_original_lab = cv.cvtColor(image_original, cv.COLOR_RGB2LAB) # Convert the RGB image to LAB color space
image_gamma_lab = cv.cvtColor(image_gamma, cv.COLOR_RGB2LAB) # Convert the RGB image back to LAB color space

f, axarr = plt.subplots(2, 1), plt.subplots_adjust(hspace=0.3)

hist_orig_L = cv.calcHist([image_original_lab], [0], None, [256], [0, 256])
axarr[0].set_ylim(0, 15000), axarr[0].plot(hist_orig_L, color='m')

hist_gamma_L = cv.calcHist([image_gamma_lab], [0], None, [256], [0, 256])
axarr[0].set_ylim(0, 15000), axarr[0].plot(hist_gamma_L, color='k'), axarr[0].legend(['Original L channel', 'Gamma corrected L channel']), axar

#plot LUT
axarr[1].plot(table, color='b'), axarr[1].set_title('Gamma correction LUT'), axarr[1].set_xlabel('Input intensity'), axarr[1].set_ylabel('Outpu
```



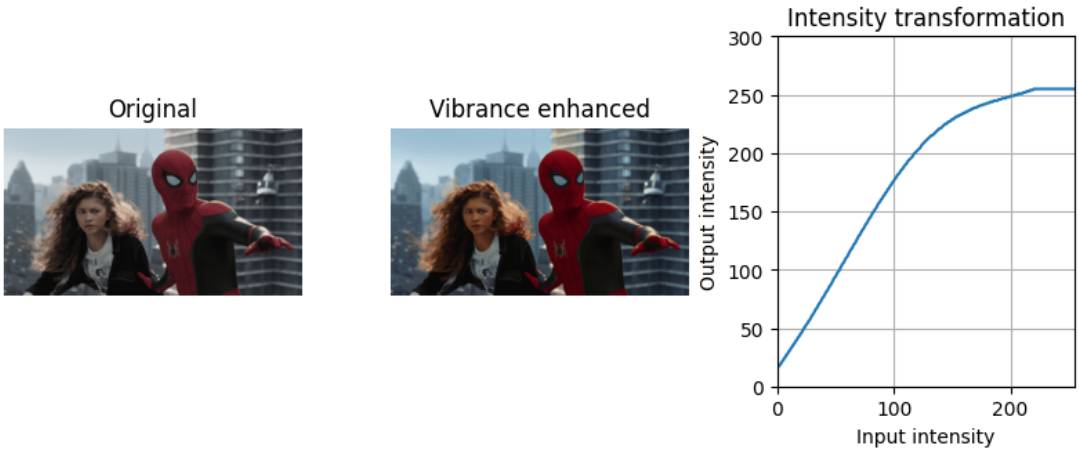


Although the L* channel is from 0-100, the default value OpenCV provides is uint8 thus it ranges from 0-255 and was considered. Gamma correction with a γ value of 0.7 is applied to the L* channel in the LAB color space, which changes the luminescence of the image thus enhances the brightness and contrast of the image. The histograms of the original and gamma-corrected images are plotted both in RGB and LAB color spaces. Gamma correction redistributes pixel values, evident in the change of intensity distribution, particularly in the L* channel.

Question 4

In [20]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np

image_original = cv.imread('spider.png', cv.IMREAD_COLOR) # Load the image in color
image_original_rgb = cv.cvtColor(image_original, cv.COLOR_BGR2RGB) # Convert the image to RGB color space
image_hsv = cv.cvtColor(image_original, cv.COLOR_BGR2HSV) # Convert the image to HSV color space
h, s, v = cv.split(image_hsv) # Split the HSV image into its planes

# Apply intensity transformation to s plane
a = 0.65 # Adjusting a gives a visually pleasing output
sigma = 70 # Given
x_values = np.arange(256) # Create an array of x values in the range [0, 255]
f_x = np.array(np.minimum(x_values + a * 128 * np.exp(-((x_values - 128) ** 2) / (2 * sigma ** 2)), 255), dtype = np.uint8) # Calculate f(x) us
f_s = cv.LUT(s, f_x) # Apply the intensity transformation to the s plane

image_hsv_vibrant = cv.merge((h, f_s, v)) # Merge the planes back into an HSV image
image_rgb_vibrant = cv.cvtColor(image_hsv_vibrant, cv.COLOR_HSV2RGB) # Convert the HSV image back to RGB color space

# Display the original, vibrance-enhanced images and the intensity transformation
```

```
f, axarr = plt.subplots(1, 3, figsize=(10,10))
plt.subplots_adjust(wspace=0.3)
axarr[0].imshow(image_original_rgb), axarr[0].set_title('Original'), axarr[0].axis('off')
axarr[1].imshow(image_rgb_vibrant), axarr[1].set_title('Vibrance enhanced'), axarr[1].axis('off')
axarr[2].plot(f_x), axarr[2].set_title('Intensity transformation'), axarr[2].set_xlabel('Input intensity'),axarr[2].set_ylabel('Output intensit
plt.show()
```

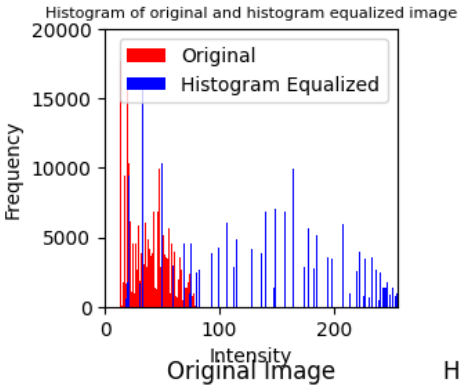


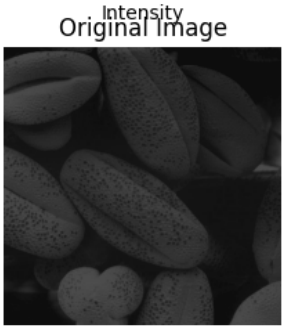
The code splits an image into its hue, saturation, and value (HSV) components and applies the given intensity transformation to the saturation plane (S-plane) to enhance the vibrance of colors while maintaining the overall image quality.The parameter a is selected as 0.65 for adjusting the intensity transformation, allowing to fine-tune the vibrance enhancement and achieve a pleasing output that enhances color vibrancy.

Question 5

In [120…

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

image = cv.imread('shells.tif', cv.IMREAD_GRAYSCALE) # read image in grayscale
fig, ax = plt.subplots(2,2, figsize=(6,6))
plt.subplots_adjust(hspace=0.3)
ax[0,0].hist(image.flatten(), 256, [0, 256], color='r') # plot histogram of original image

ax[1,0].imshow(cv.cvtColor(image, cv.COLOR_GRAY2RGB), cmap='gray') # convert image to RGB color space since it has a tif extension and show ima
ax[1,0].set_title('Original Image'), ax[1,0].axis('off')

def histogram_equalization(img):
    rows = img.shape[0] # get number of rows
    cols = img.shape[1] # get number of columns
    MN = rows * cols # total number of pixels
    histogram_array = np.zeros((256,), dtype = np.uint16) # initialize histogram array
    LUT_array = np.zeros((256,), dtype = np.uint16) # initialize LUT array

    #Calculate histogram
    for i in range(rows):
        for j in range(cols):
            histogram_array[img[i,j]] += 1 # calculate histogram

    #Calculate LUT
    for i in range(256):
        LUT_array[i] = round((255 / MN) * np.sum(histogram_array[0:i])) # calculate LUT

    #Histogram equalization
    for i in range(rows):
        for j in range(cols):
            img[i,j] = LUT_array[img[i,j]] # apply LUT to image
    #Another method to calculate histogram by applying LUT function
    # image = cv.LUT(img, LUT_array, img) # apply LUT to image

    return img

equ = histogram_equalization(image) # apply histogram equalization function
ax[1,1].imshow(cv.cvtColor(equ, cv.COLOR_GRAY2RGB), cmap='gray') # show histogram equalized image
ax[1,1].set_title('Histogram Equalized Image'), ax[1,1].axis('off')

#plot histogram of histogram equalized image on same plot
ax[0,0].hist(equ.flatten(), 256, [0, 256], color='b') # plot histogram of histogram equalized image
ax[0,0].set_title('Histogram of original and histogram equalized image', fontsize=8), ax[0,0].set_xlabel('Intensity'), ax[0,0].set_ylabel('Freq
ax[0,1].remove()
```



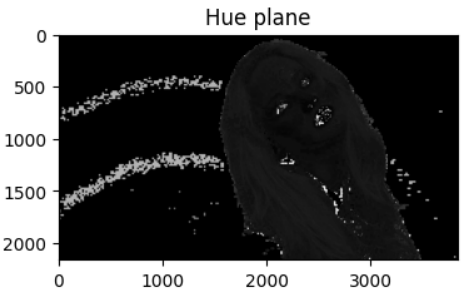


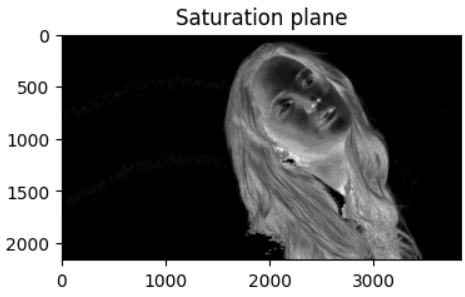
In the original image, only the dark intensities are frequent and bright intensities are nearly zero but after histogram equalization, the histogram is distributed among all intensities, producing an image with all intensities.

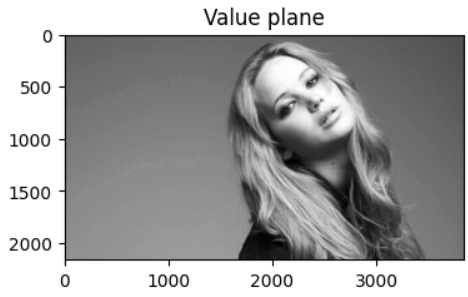
Question 6

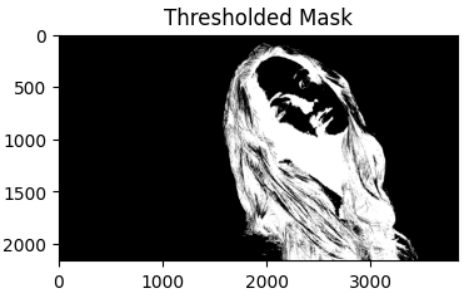
In [121…

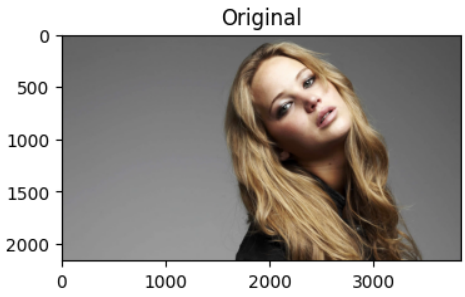
```
%matplotlib inline
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np

#Q6(a)
image_original = cv.imread('jeniffer.jpg', cv.IMREAD_COLOR) # Load the image in color
h, s, v = cv.split(cv.cvtColor(image_original, cv.COLOR_BGR2HSV)) # Convert the image to HSV color space and split into its planes
fig, ax = plt.subplots(1, 3, figsize=(14, 14))
ax[0].imshow(h, cmap='gray'), ax[0].set_title("Hue plane"), ax[1].imshow(s, cmap='gray'), ax[1].set_title("Saturation plane"), ax[2].imshow(v,

# Q6(b) Selected Saturation (s) plane for thresholding
_, thresholded_mask = cv.threshold(s, 90, 255, cv.THRESH_BINARY) # Threshold the saturation plane
# Q6(c)
foreground = cv.bitwise_and(v, thresholded_mask) # Apply the mask to the value plane
# Q6(d)
hist_foreground = cv.calcHist([foreground], [0], thresholded_mask, [256], [0, 256]) # Calculate the histogram of the foreground
cumulative_hist = np.cumsum(hist_foreground) # Calculate the cumulative histogram
# Q6(e)
equalized_foreground = cv.equalizeHist(foreground) # Equalize the foreground
# Q6(f)
background = cv.bitwise_not(thresholded_mask) # Invert the mask
background_image = cv.bitwise_and(v, background) # Apply the inverted mask to the value plane
# Add the equalized foreground with the background
result_value = cv.add(equalized_foreground, background_image)
# Display results
fig, ax = plt.subplots(1, 3, figsize=(14, 14))
ax[0].imshow(thresholded_mask, cmap='gray'), ax[0].set_title("Thresholded Mask")
ax[1].imshow(cv.cvtColor(image_original, cv.COLOR_BGR2RGB)), ax[1].set_title("Original")
ax[2].imshow(result_value, cmap='gray'), ax[2].set_title("Histogram-equalized foreground")
plt.show()
```

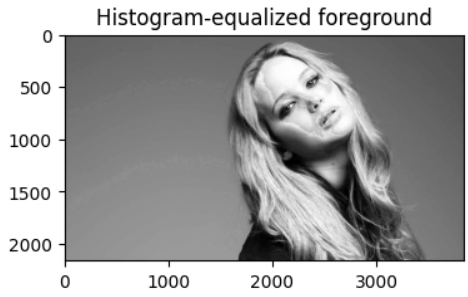










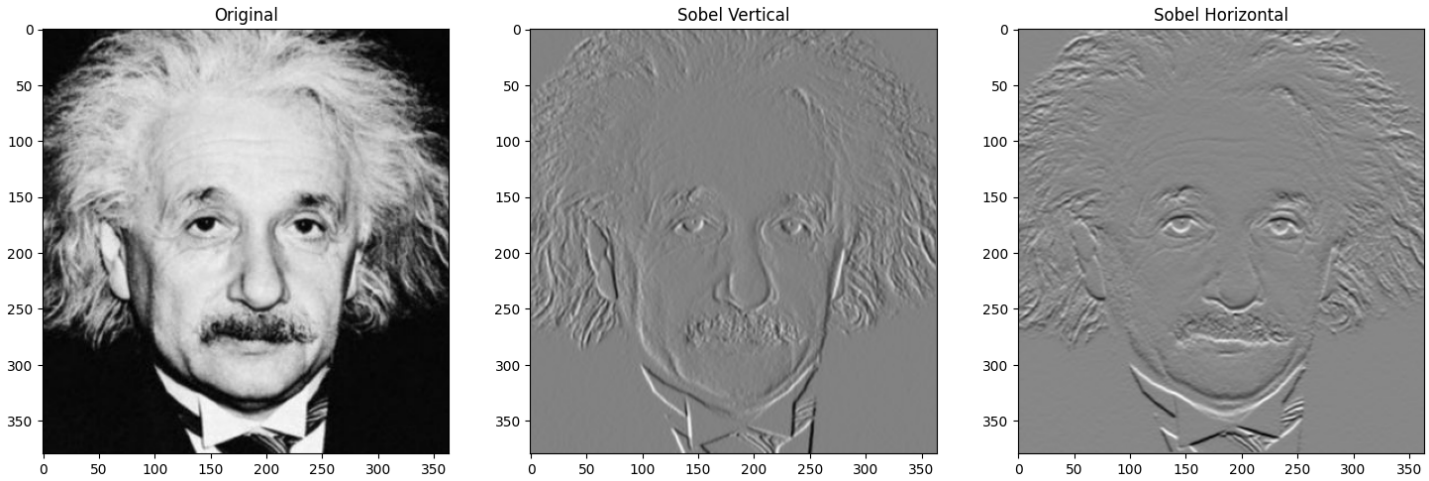


The Saturation plane is selected for thresholding and selectively applies histogram equalization only to the foreground of an image, enhancing the contrast and details in the foreground while preserving the background. It visually presents the Hue, Saturation, and Value planes of the image, the thresholded mask, the original image, and the result with the histogram-equalized foreground.

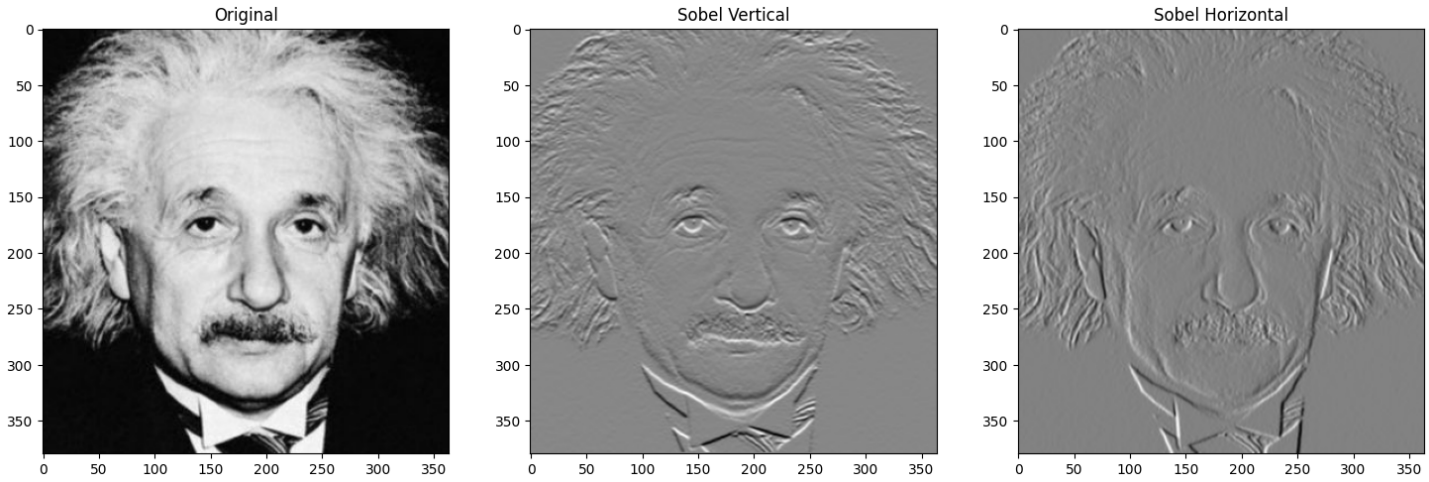
The concept of enhancing the foreground of an image by isolating it using a thresholded mask and then applying histogram equalization is used. This selective enhancement technique improves the visibility and quality of specific regions in an image while keeping other areas intact.

Question 7

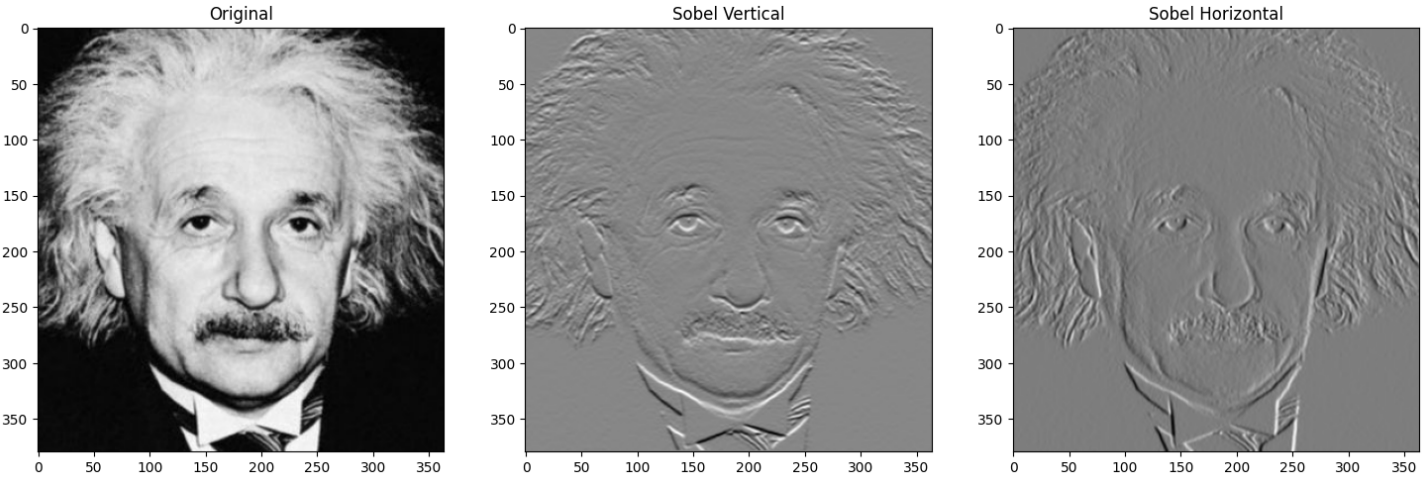
In [98]:
```
%matplotlib inline
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

# (a)
img = cv.imread('einstein.png',cv.IMREAD_GRAYSCALE).astype(np.float32) # read image in grayscale
# Sobel vertical
kv = np.array([(-1, -2, -1), (0, 0, 0), (1, 2, 1)], dtype='float') # kernel
img_sobel_v = cv.filter2D(img, -1, kv) # apply kernel to image
# Sobel horizontal
kh = np.array([(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)], dtype='float') # kernel
img_sobel_h = cv.filter2D(img, -1, kh) # apply kernel to image
fig, axes  = plt.subplots(1, 3, figsize=(18,9)) # create figure with 3 subplots
axes[0].imshow(img, cmap='gray'), axes[0].set_title('Original') # show original image
axes[1].imshow(img_sobel_h, cmap='gray'), axes[1].set_title('Sobel Vertical') # show sobel vertical image
axes[2].imshow(img_sobel_v, cmap='gray'), axes[2].set_title('Sobel Horizontal') # show sobel horizontal image
plt.show()
```



In [100…
```
# (b)
img_sobel_v = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=3) # apply sobel vertical kernel to image
img_sobel_h = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=3) # apply sobel horizontal kernel to image

fig, axes  = plt.subplots(1, 3, figsize=(18,9)) # create figure with 3 subplots
axes[0].imshow(img, cmap='gray'), axes[0].set_title('Original') # show original image
axes[1].imshow(img_sobel_v, cmap='gray'), axes[1].set_title('Sobel Vertical') # show sobel vertical image
axes[2].imshow(img_sobel_h, cmap='gray'), axes[2].set_title('Sobel Horizontal') # show sobel horizontal image
plt.show()
```



In [101…
```
# (c)
kh = np.array([1,2,1], dtype=np.float32) # kernel
kv = np.array([1,0,-1], dtype=np.float32) # kernel
img_sobel_vimg_sobel_v = cv.sepFilter2D(img, -1, kh, kv) # apply kernel to image
img_sobel_h = cv.sepFilter2D(img, -1, kv, kh) # apply kernel to image

fig, axes  = plt.subplots(1, 3, figsize=(18,9)) # create figure with 3 subplots
axes[0].imshow(img, cmap='gray'), axes[0].set_title('Original') # show original image
axes[1].imshow(img_sobel_v, cmap='gray'), axes[1].set_title('Sobel Vertical') # show sobel vertical image
axes[2].imshow(img_sobel_h, cmap='gray'), axes[2].set_title('Sobel Horizontal') # show sobel horizontal image
plt.show()
```

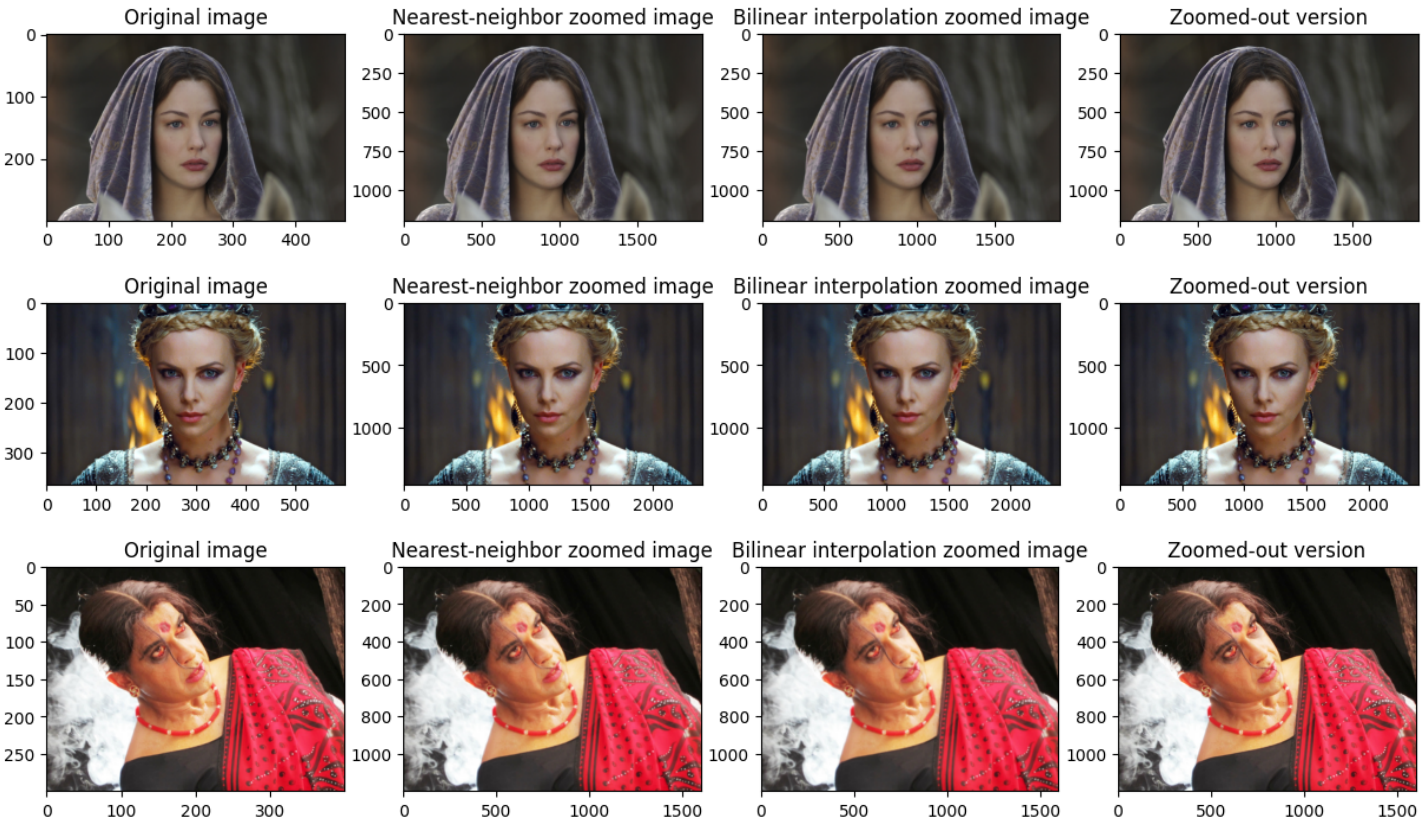


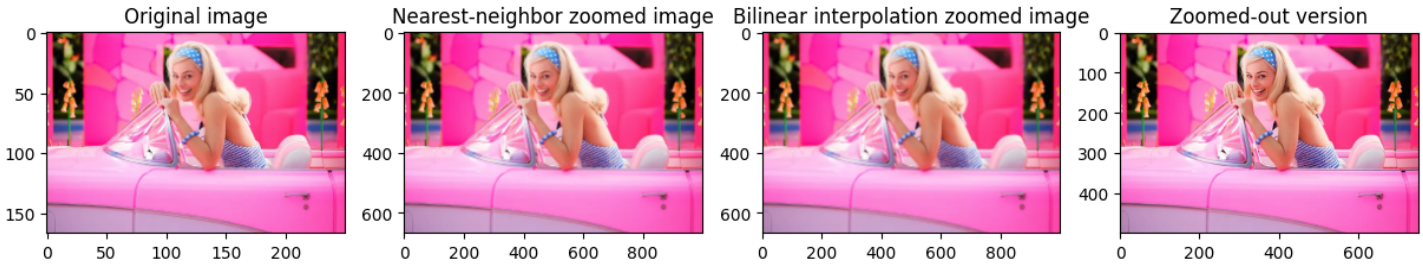
The Sobel operator is applied to the image using different techniques, including using the existing filter2D function, built-in Sobel functions, and custom Sobel filtering. It generates Sobel-filtered images in both vertical and horizontal directions. The program shows various methods for Sobel filtering, demonstrating that the same Sobel operation can be achieved using different approaches, including manual kernel convolution and built-in functions. It highlights the flexibility of Sobel filtering techniques in computing image gradients.

Question 8

In [103…

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

originals = ["im02small.png", "im03small.png", "im09small.png","im11small.png"] # original images
zoomed_outs = ["im02.png", "im03.png", "im09.png", "im11.png"] # zoomed-out images

for i in range(4): # loop through images
    image = cv.imread(originals[i])
    image_zoomed_out = cv.imread(zoomed_outs[i])

    image_near = cv.resize(image, None, fx=4, fy=4, interpolation=cv.INTER_NEAREST) # nearest-neighbor interpolation
    image_bilinear = cv.resize(image, None, fx=4, fy=4, interpolation=cv.INTER_LINEAR) # bilinear interpolation

    fig, ax = plt.subplots(1,4, figsize=(15,15)) # create figure with 4 subplots
    ax[0].imshow(cv.cvtColor(image,cv.COLOR_BGR2RGB)), ax[0].set_title("Original image") # show original image
    ax[1].imshow(cv.cvtColor(image_near,cv.COLOR_BGR2RGB)), ax[1].set_title("Nearest-neighbor zoomed image") # show nearest-neighbor zoomed ima
    ax[2].imshow(cv.cvtColor(image_bilinear,cv.COLOR_BGR2RGB)), ax[2].set_title("Bilinear interpolation zoomed image") # show bilinear interpol
    ax[3].imshow(cv.cvtColor(image_zoomed_out ,cv.COLOR_BGR2RGB)), ax[3].set_title("Zoomed-out version") # show zoomed-out image
    plt.show()
```



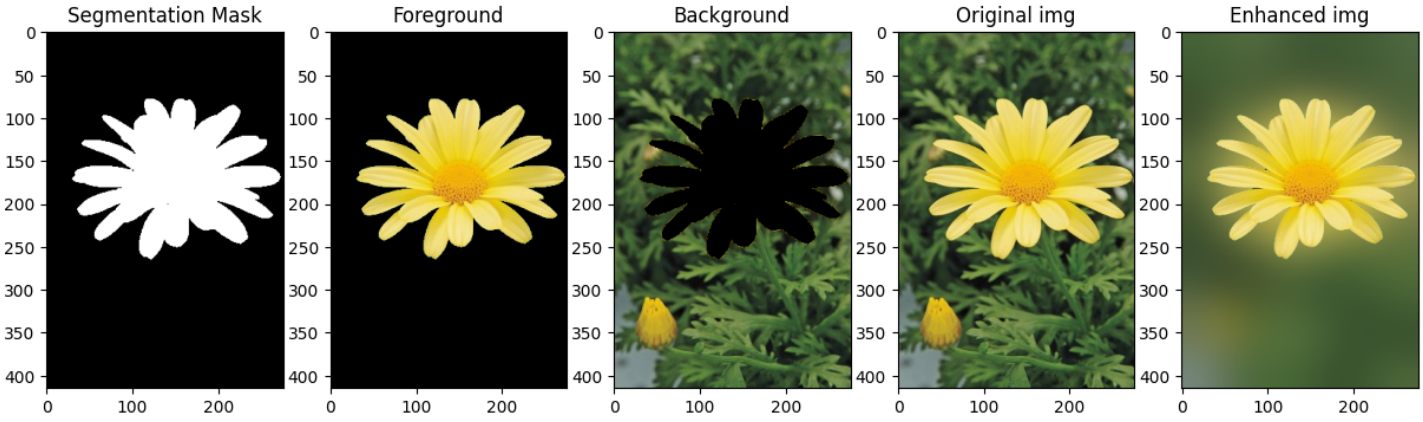


The program visually presents the zoomed images and their comparison with the original and zoomed-out versions to assess the zooming techniques' performance. The Normalized Sum of Squared Differences (SSD) is used to measure the accuracy of the zooming techniques in scaling up small images.

Question 9

In [112…

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

img = cv.imread('flower.jpg')
# Q9(a)
segment_mask = np.zeros(img.shape[:2], np.uint8) # create mask
rect = (30, 30, img.shape[1] - 30, img.shape[0] - 150) # create rectangle
background = np.zeros((1, 65), np.float64) # create background
foreground = np.zeros((1, 65), np.float64) # create foreground
cv.grabCut(img, segment_mask, rect, background, foreground, 5, cv.GC_INIT_WITH_RECT) # apply grabcut
mask2 = np.where((segment_mask == 2) | (segment_mask == 0), 0, 1).astype('uint8') # create mask
segmented_img = img * mask2[:, :, np.newaxis] # apply mask to image
fig, ax = plt.subplots(1,5, figsize = (15,15)) # create figure with 5 subplots
ax[0].imshow(mask2, cmap='gray'), ax[0].set_title('Segmentation Mask') # show segmentation mask
ax[1].imshow(cv.cvtColor(segmented_img,cv.COLOR_BGR2RGB)), ax[1].set_title('Foreground') # show foreground
ax[2].imshow(cv.cvtColor(img - segmented_img,cv.COLOR_BGR2RGB)), ax[2].set_title('Background') # show background

# Q9(b)
blurred_background = cv.GaussianBlur(img, (0, 0), 30) # apply gaussian blur to background
enhanced_img = np.where(mask2[:, :, np.newaxis] == 1, img, blurred_background) # apply mask to image
ax[3].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB)), ax[3].set_title('Original img') # show original image
ax[4].imshow(cv.cvtColor(enhanced_img, cv.COLOR_BGR2RGB)), ax[4].set_title('Enhanced img') # show enhanced image
plt.show()
```



GrabCut is used to segment an input image into foreground and background regions, creating a binary mask (mask2) to distinguish between them. Gaussian blur is applied to the background of the original image and combined with the foreground, resulting in an enhanced image where the background is blurred while the foreground remains unchanged.