190622R_a01

February 27, 2022

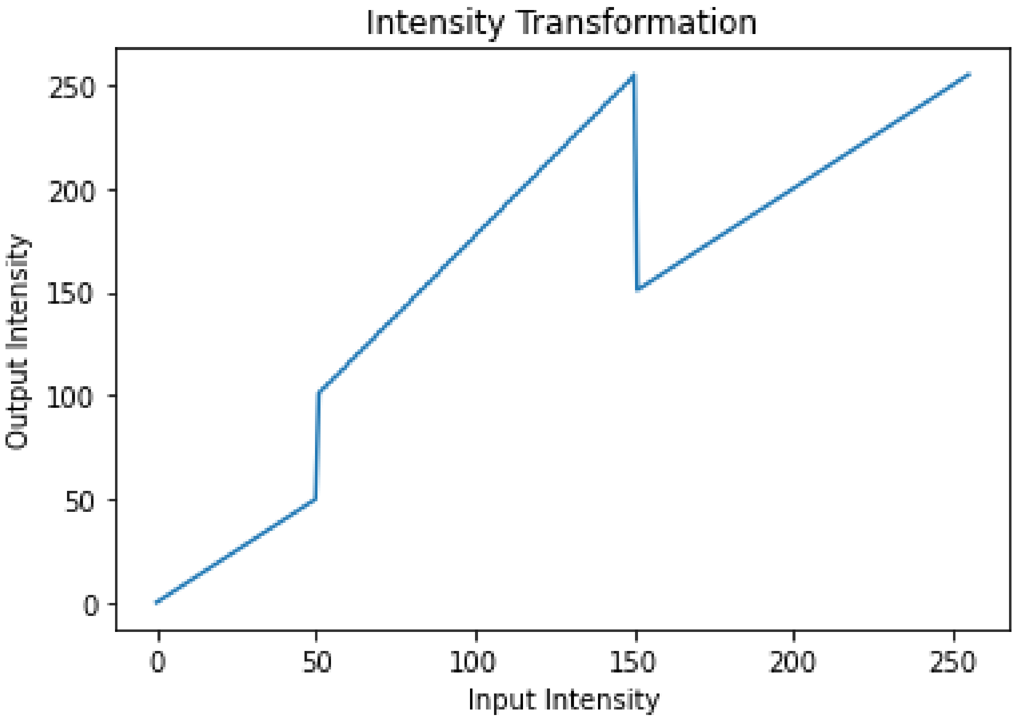
1 Name : Tharindu O.K.D

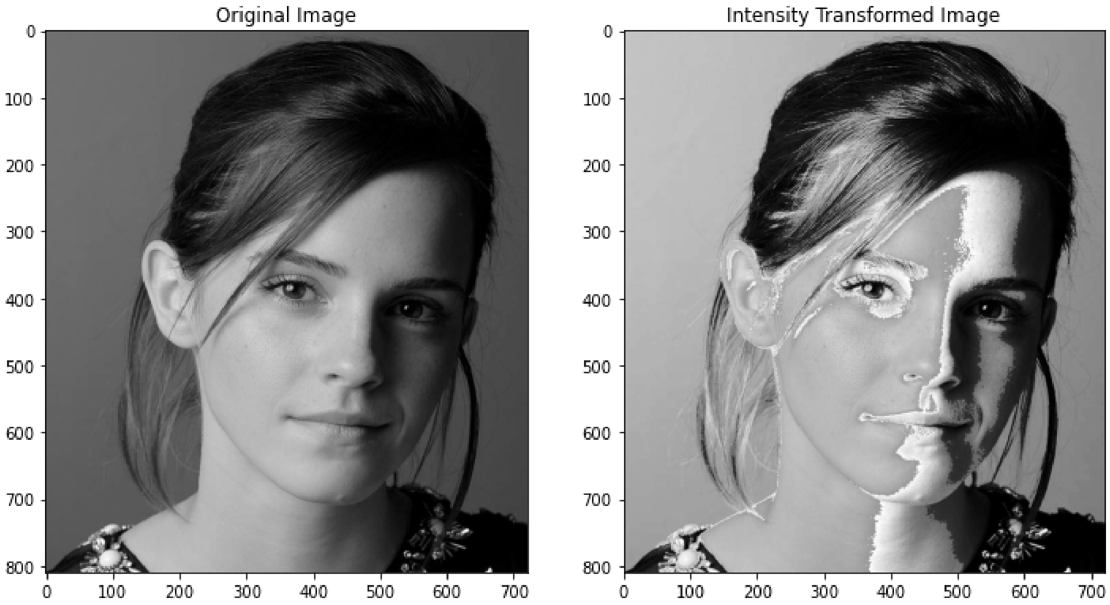
2 Index No. : 190622R

Question 01

[ ]:

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

array_1 = np.array([ i for i in range(0,51)])
array_2 = np.array([ (155 / 100) * (i - 50) + 100 for i in range(51,151)])
array_3 = np.array([i for i in range(151,256)])
transform = np.concatenate((array_1, array_2, array_3),axis=0).astype(np.uint8)

fig, ax = plt.subplots()
ax.plot(transform)
ax.set_title("Intensity Transformation")
ax.set_xlabel("Input Intensity")
ax.set_ylabel("Output Intensity")

im = cv.imread(r"emma_gray.jpg", cv.IMREAD_GRAYSCALE)
assert im is not None

transformed_image = cv.LUT(im, transform)

fig, ax = plt.subplots(1, 2, figsize=(12, 8))
ax[0].imshow(im, cmap="gray", vmin=0, vmax=255)
ax[0].set_title("Original Image")
ax[1].imshow(transformed_image, cmap="gray", vmin=0, vmax=255)
ax[1].set_title("Intensity Transformed Image")
plt.show()
```

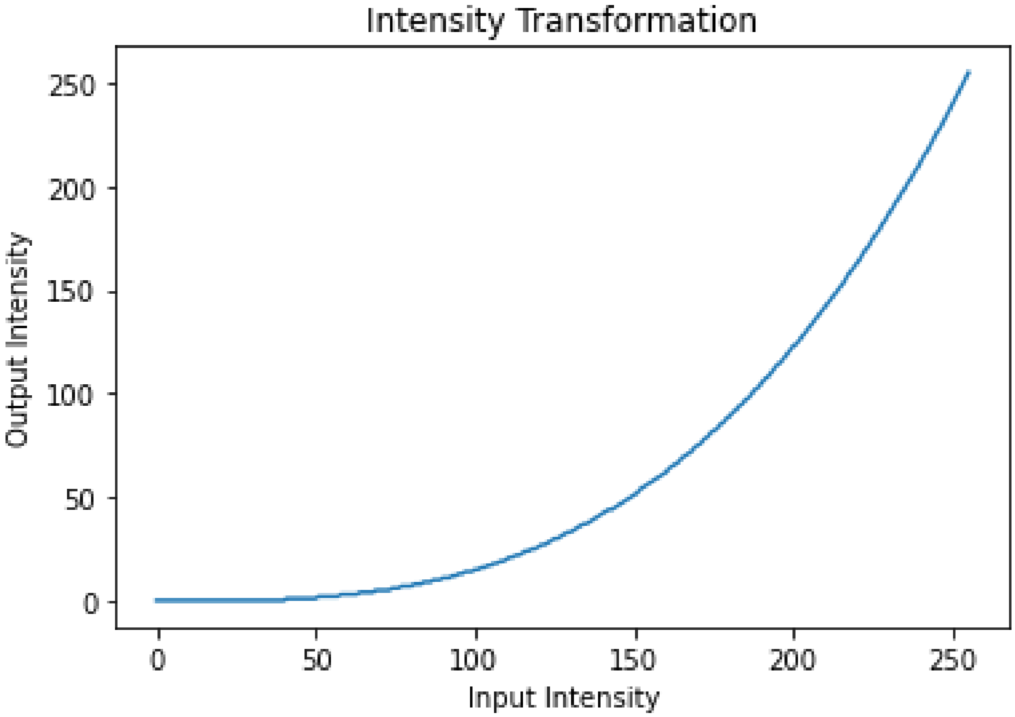

Intensity Transformation
Output Intensity
Input Intensity
0
50
100
150
200
250
0
50
100
150
200
250

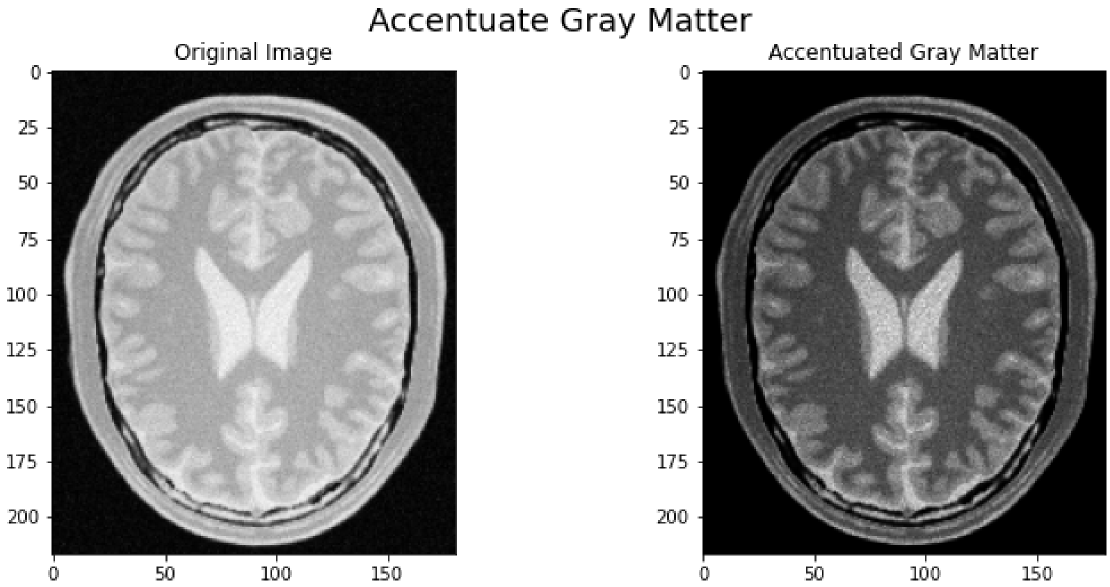

Original Image
Intensity Transformed Image
0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700
0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700

Question 02

[ ]:

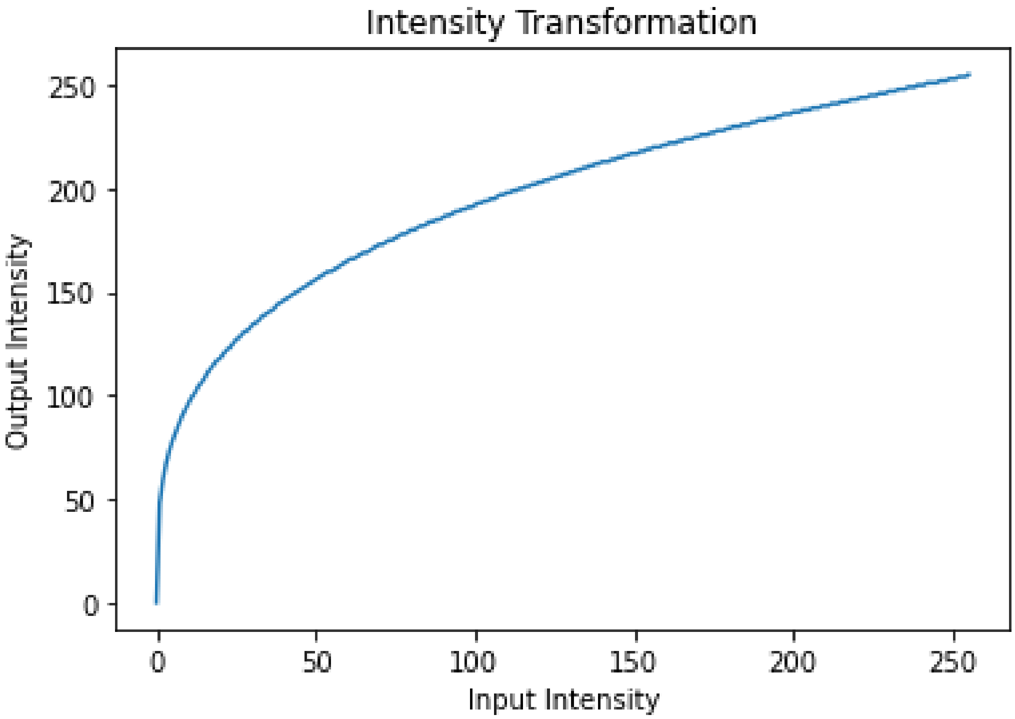
```
im = cv.imread(r"brain_proton_density_slice.png", cv.IMREAD_GRAYSCALE)
assert im is not None

# accentuate white matter
white_transform = np.array([((i / 255) ** 3) * 255 for i in␣
 ↪range(0,256)],dtype=np.uint8)
white_transformed_image = cv.LUT(im, white_transform)

fig, ax = plt.subplots()
ax.plot(white_transform)
ax.set_title("Intensity Transformation")
ax.set_xlabel("Input Intensity")
ax.set_ylabel("Output Intensity")
plt.show()

fig, ax = plt.subplots(1, 2, figsize=(12, 5))
fig.suptitle("Accentuate Gray Matter", fontsize=18)
ax[0].imshow(im, cmap="gray", vmin=0, vmax=255)
ax[0].set_title("Original Image")
ax[1].imshow(white_transformed_image, cmap="gray", vmin=0, vmax=255)
ax[1].set_title("Accentuated Gray Matter")
plt.show()
```

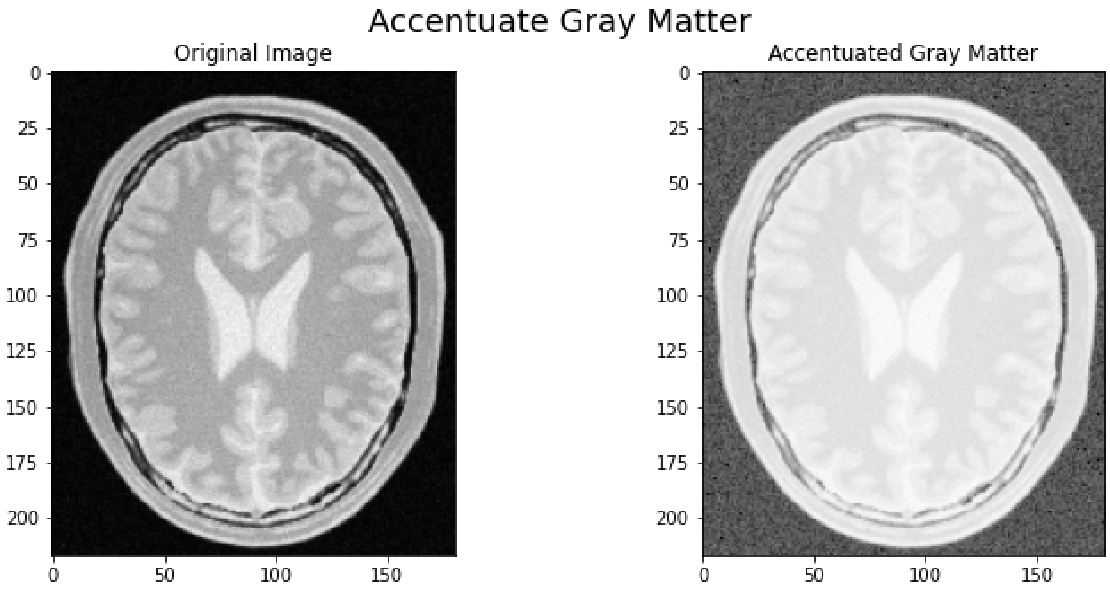




[ ]:

```
gray_transform = np.array([((i / 255) ** 0.3) * 255 for i in range(0,256)],dtype=np.uint8)
gray_transformed_image = cv.LUT(im, gray_transform)

fig, ax = plt.subplots()
ax.plot(gray_transform)
ax.set_title("Intensity Transformation")
ax.set_xlabel("Input Intensity")
ax.set_ylabel("Output Intensity")
plt.show()

fig, ax = plt.subplots(1, 2, figsize=(12, 5))
fig.suptitle("Accentuate Gray Matter", fontsize=18)
ax[0].imshow(im, cmap="gray", vmin=0, vmax=255)
ax[0].set_title("Original Image")
ax[1].imshow(gray_transformed_image, cmap="gray", vmin=0, vmax=255)
ax[1].set_title("Accentuated Gray Matter")
plt.show()
```



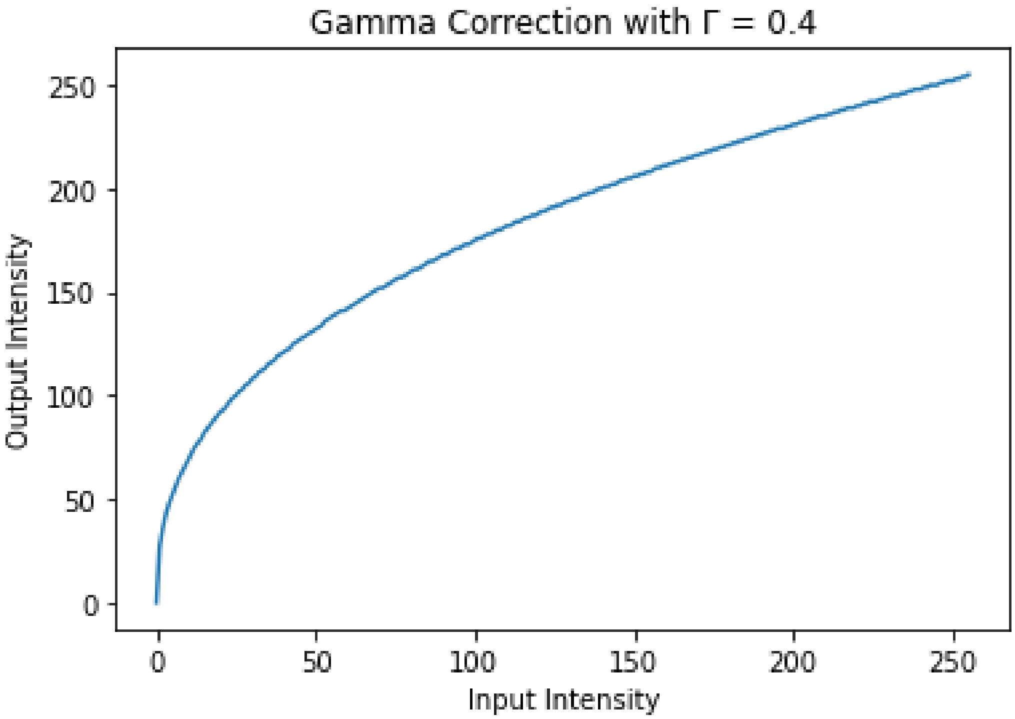


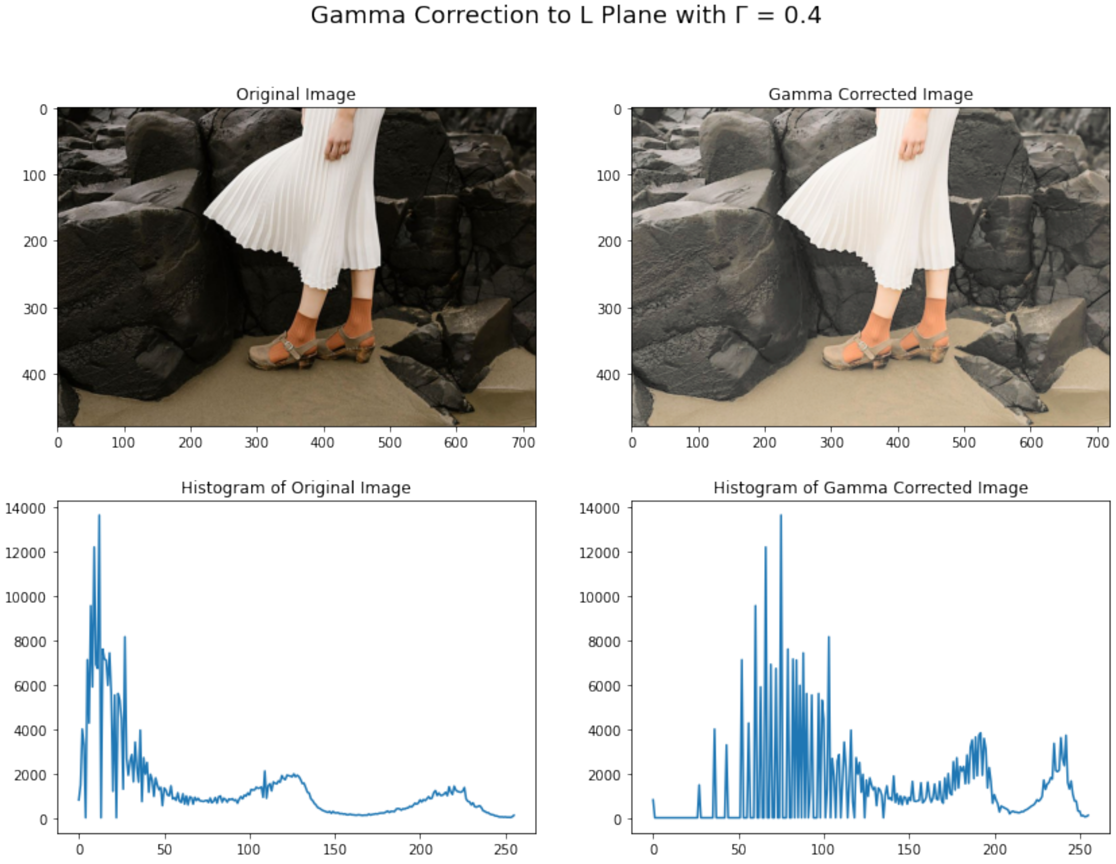
Question 03

[ ]:

```
im = cv.imread(r"highlights_and_shadows.jpg")
assert im is not None
```

```
LAB_image = cv.cvtColor(im, cv.COLOR_BGR2LAB)
original_hist = cv.calcHist([LAB_image], [0], None, [256], [0, 256])
gamma = 0.4
gamma_correction = np.array([((i / 255) ** gamma) * 255 for i in␣
 ↪range(0,256)],dtype=np.uint8)
LAB_image[:, :, 0] = cv.LUT(LAB_image[:, :, 0], gamma_correction)

fig, ax = plt.subplots()
ax.plot(gamma_correction)
ax.set_title("Gamma Correction with Γ = {}".format(gamma))
ax.set_xlabel("Input Intensity")
ax.set_ylabel("Output Intensity")
plt.show()

corrected_hist = cv.calcHist([LAB_image], [0], None, [256], [0, 256])

fig, ax = plt.subplots(2, 2, figsize=(14,10))
fig.suptitle("Gamma Correction to L Plane with Γ = {}".format(gamma),␣
 ↪fontsize=18)
ax[0, 0].imshow(cv.cvtColor(im, cv.COLOR_BGR2RGB))
ax[0, 0].set_title("Original Image")
ax[0, 1].imshow(cv.cvtColor(LAB_image, cv.COLOR_LAB2RGB))
ax[0, 1].set_title("Gamma Corrected Image")
ax[1, 0].plot(original_hist)
ax[1, 0].set_title("Histogram of Original Image")
ax[1, 1].plot(corrected_hist)
ax[1, 1].set_title("Histogram of Gamma Corrected Image")
plt.show()
```

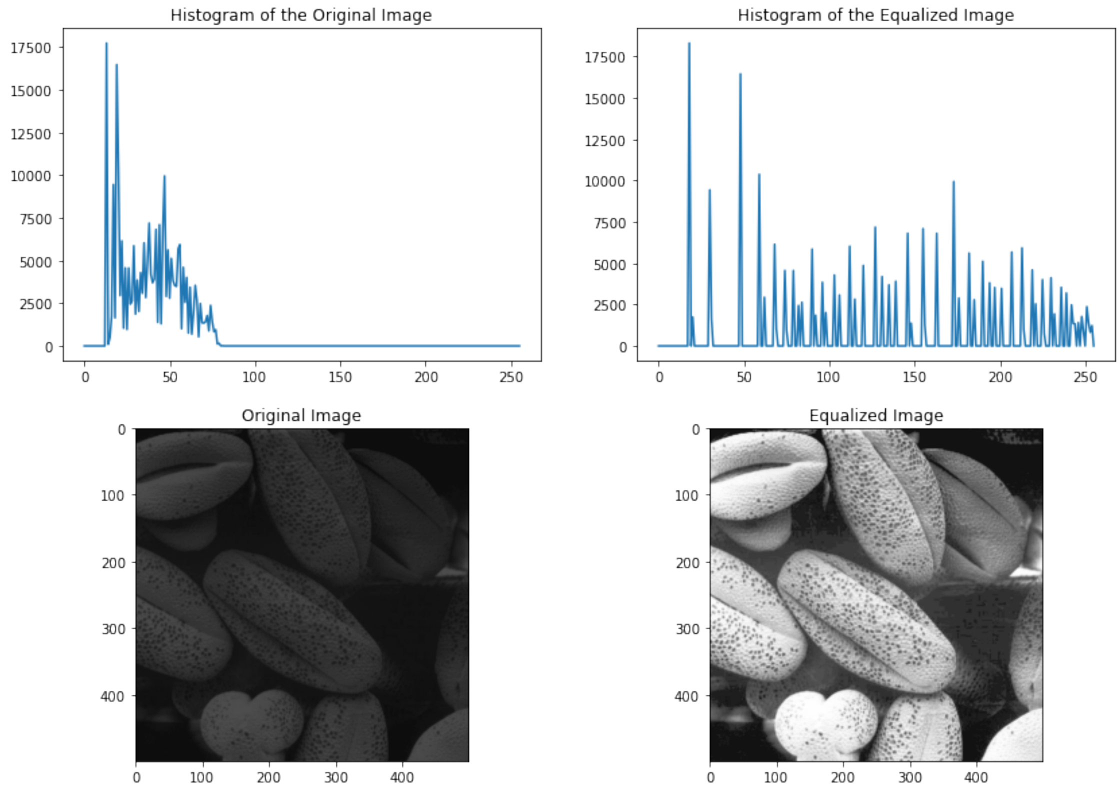

Gamma Correction with Γ = 0.4
Output Intensity
0
50
100
150
200
250
Input Intensity
0
50
100
150
200
250


Gamma Correction to L Plane with Γ = 0.4
Original Image
Gamma Corrected Image
0
100
200
300
400
0
100
200
300
400
500
600
700
Histogram of Original Image
Histogram of Gamma Corrected Image
0
2000
4000
6000
8000
10000
12000
14000
0
50
100
150
200
250

Question 04

[ ]:

```
img = cv.imread(r"shells.png", cv.IMREAD_GRAYSCALE)
assert im is not None

original_hist = cv.calcHist([img], [0], None, [256], [0, 256])
cdf = original_hist.cumsum()
MN = img.shape[0] * img.shape[1]
equalize_transformation = np.array((cdf * 255) / MN, dtype=np.uint8)
equalize_img = cv.LUT(img, equalize_transformation)
equalize_hist = cv.calcHist([equalize_img], [0], None, [256], [0, 256])


fig, ax = plt.subplots(2, 2, figsize=(14,10))
ax[0, 0].plot(original_hist)
ax[0, 0].set_title("Histogram of the Original Image")
ax[0, 1].plot(equalize_hist)
ax[0, 1].set_title("Histogram of the Equalized Image")
ax[1, 0].imshow(img, cmap="gray", vmin=0, vmax=255)
ax[1, 0].set_title("Original Image")
ax[1, 1].imshow(equalize_img, cmap="gray", vmin=0, vmax=255)
ax[1, 1].set_title("Equalized Image")
plt.show()
```



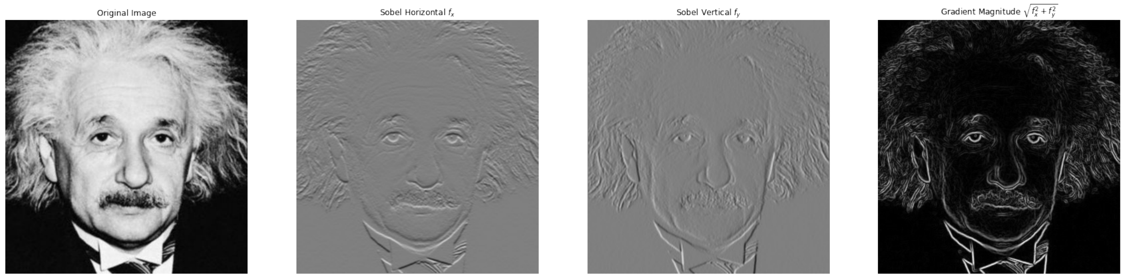
[ ]:

```python
import math
def nearestNeighborInterpolation(img, scale):
    scaled_dimentions = (round(img.shape[0] * scale), round(img.shape[1] * scale), img.shape[2])
    scaled_img = np.zeros(scaled_dimentions,dtype=np.uint8)

    for i in range(scaled_dimentions[0]):
        for j in range(scaled_dimentions[1]):
            scaled_img[i, j] = img[min(round(i / scale), img.shape[0] - 1), min(round(j / scale), img.shape[1] - 1)]

    return scaled_img

def bilinearInterpolation(img, scale):
    scaled_dimentions = (round(img.shape[0] * scale), round(img.shape[1] * scale), img.shape[2])
    scaled_img = np.zeros(scaled_dimentions,dtype=np.uint8)

    x_ratio = (img.shape[1] - 1) / (scaled_dimentions[1] - 1)
    y_ratio = (img.shape[0] - 1) / (scaled_dimentions[0] - 1)
    for i in range(scaled_dimentions[0]):
        for j in range(scaled_dimentions[1]):
            x_floor = math.floor(j * x_ratio)
            x_ceil = min(math.ceil(j * x_ratio), img.shape[1] - 1)
            y_floor = math.floor(i * y_ratio)
            y_ceil = min(math.ceil(i * y_ratio), img.shape[0] - 1)

            x_weight = (j * x_ratio) - x_floor
            y_weight = (i * y_ratio) - y_floor

            pixel_value = img[y_floor, x_floor] * (1 - x_weight) * (1 -␣
 ↪y_weight) + \
                img[y_ceil, x_floor] * (x_weight) * (1 - y_weight) + \
                    img[y_floor, x_ceil] * (1 - x_weight) * (y_weight) + \
                        img[y_ceil, x_ceil] * (x_weight) * (y_weight)

            scaled_img[i, j] = pixel_value

    return scaled_img

def scale_image(img, scale, method):
    if method == "NEAREST_NEIGHBOR":
        return nearestNeighborInterpolation(img, scale)
    elif method == "BILINEAR":
        return bilinearInterpolation(img, scale)


img = cv.imread(r"a1q5images/a1q5images/im02small.png")
assert im is not None
```

[ ]:

```
img = cv.imread(r"einstein.png", cv.IMREAD_GRAYSCALE).astype(np.float32)
assert im is not None

sobel_v_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.
 ↪float32)
f_x = cv.filter2D(img, -1, sobel_v_kernel)
sobel_h_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.
 ↪float32)
f_y = cv.filter2D(img, -1, sobel_h_kernel)
grad_mag_img = np.sqrt(f_x**2 + f_y**2)


fig, ax = plt.subplots(1, 4, figsize=(30, 18))
ax[0].imshow(img, cmap="gray", vmin=0, vmax=255)
ax[0].set_title("Original Image")
ax[1].imshow(f_x, cmap="gray", vmin=-1020, vmax=1020)
ax[1].set_title(r"Sobel Horizontal $f_x$")
```

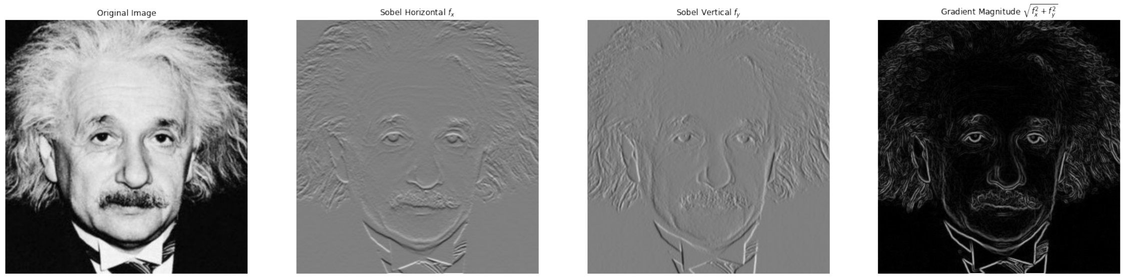
```
ax[2].imshow(f_y, cmap="gray", vmin=-1020, vmax=1020)
ax[2].set_title(r"Sobel Vertical $f_y$")
ax[3].imshow(grad_mag_img, cmap="gray")
ax[3].set_title(r"Gradient Magnitude $\sqrt{f_x^2+f_y^2}$")
for i in range(4):
    ax[i].axis("off")
plt.show()
```



[ ]:

```
def spacial_filter(img, kernel):
    assert kernel.shape[0]%2 == 1 and kernel.shape[1]%2 == 1
    kernel = np.rot90(np.rot90(kernel)) # rotate 180 for convolution
    v_padding = int(kernel.shape[0] / 2)
    h_padding = int(kernel.shape[1] / 2)
    padded_img = np.zeros((img.shape[0] + 2 * v_padding, img.shape[1] + 2 *␣
 ↪h_padding ), dtype=np.float32)
    padded_img[v_padding:padded_img.shape[0] - v_padding, h_padding:padded_img.
 ↪shape[1] - h_padding] = img
    filtered_image = np.zeros(img.shape, dtype=np.float32)
    for i in range(filtered_image.shape[0]):
        for j in range(filtered_image.shape[1]):
            filtered_image[i, j] = (kernel * padded_img[i: i + kernel.shape[0],␣
 ↪j: j + kernel.shape[1]]).sum()
    return filtered_image

f_x = spacial_filter(img, np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]],␣
 ↪dtype=np.float32))
f_y = spacial_filter(img, np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],␣
 ↪dtype=np.float32))
grad_mag_img = np.sqrt(f_x**2 + f_y**2)

fig, ax = plt.subplots(1, 4, figsize=(30, 18))
ax[0].imshow(img, cmap="gray", vmin=0, vmax=255)
ax[0].set_title("Original Image")
ax[1].imshow(f_x, cmap="gray", vmin=-1020, vmax=1020)
ax[1].set_title(r"Sobel Horizontal $f_x$")
ax[2].imshow(f_y, cmap="gray", vmin=-1020, vmax=1020)
```

```
ax[2].set_title(r"Sobel Vertical $f_y$")
ax[3].imshow(grad_mag_img, cmap="gray")
ax[3].set_title(r"Gradient Magnitude $\sqrt{f_x^2+f_y^2}$")
for i in range(4):
    ax[i].axis("off")
plt.show()
```



[ ]:

```
f_y = spacial_filter(img, np.array([[1], [2], [1]], dtype=np.float32))
f_y = spacial_filter(f_y, np.array([[1, 0, -1]], dtype=np.float32))

f_x = spacial_filter(img, np.array([[1], [0], [-1]], dtype=np.float32))
f_x = spacial_filter(f_x, np.array([[1, 2, 1]], dtype=np.float32))
grad_mag_img = np.sqrt(f_x**2 + f_y**2)

fig, ax = plt.subplots(1, 4, figsize=(30, 18))
ax[0].imshow(img, cmap="gray", vmin=0, vmax=255)
ax[0].set_title("Original Image")
ax[1].imshow(f_x, cmap="gray", vmin=-1020, vmax=1020)
ax[1].set_title(r"Sobel Horizontal $f_x$")
ax[2].imshow(f_y, cmap="gray", vmin=-1020, vmax=1020)
ax[2].set_title(r"Sobel Vertical $f_y$")
ax[3].imshow(grad_mag_img, cmap="gray")
ax[3].set_title(r"Gradient Magnitude $\sqrt{f_x^2+f_y^2}$")
for i in range(4):
    ax[i].axis("off")
plt.show()
```