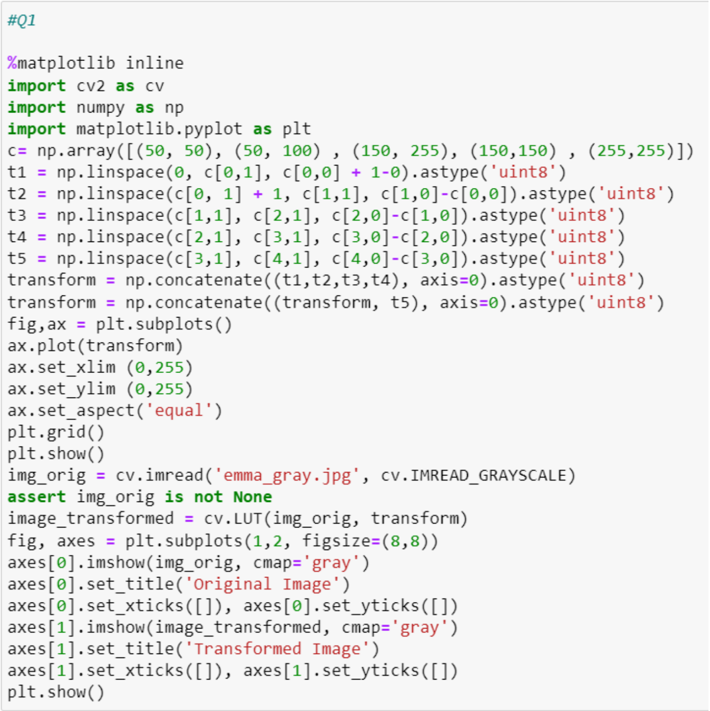
EN2550 Assignment 1 on Intensity Transformations and Neighborhood Filtering

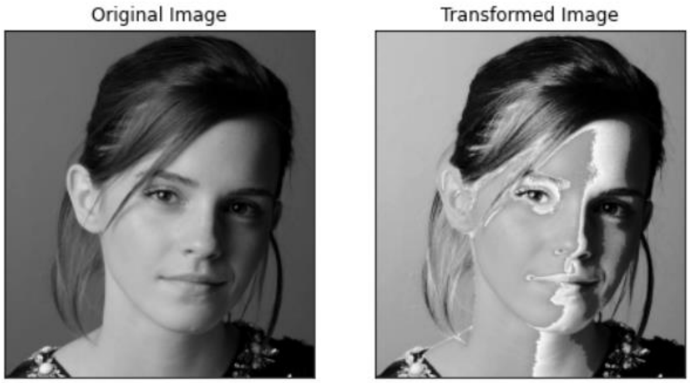
190185D

Gajaanan S.

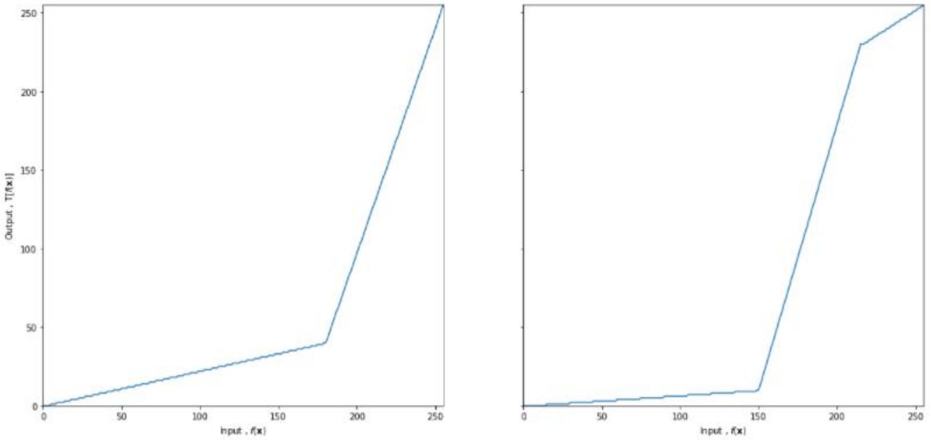
Question1

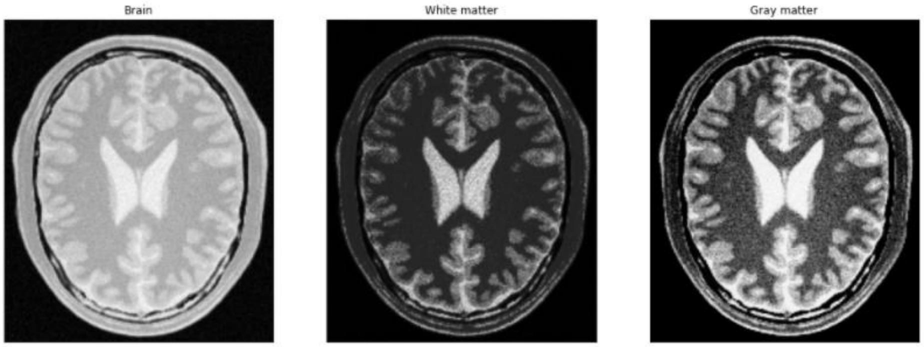


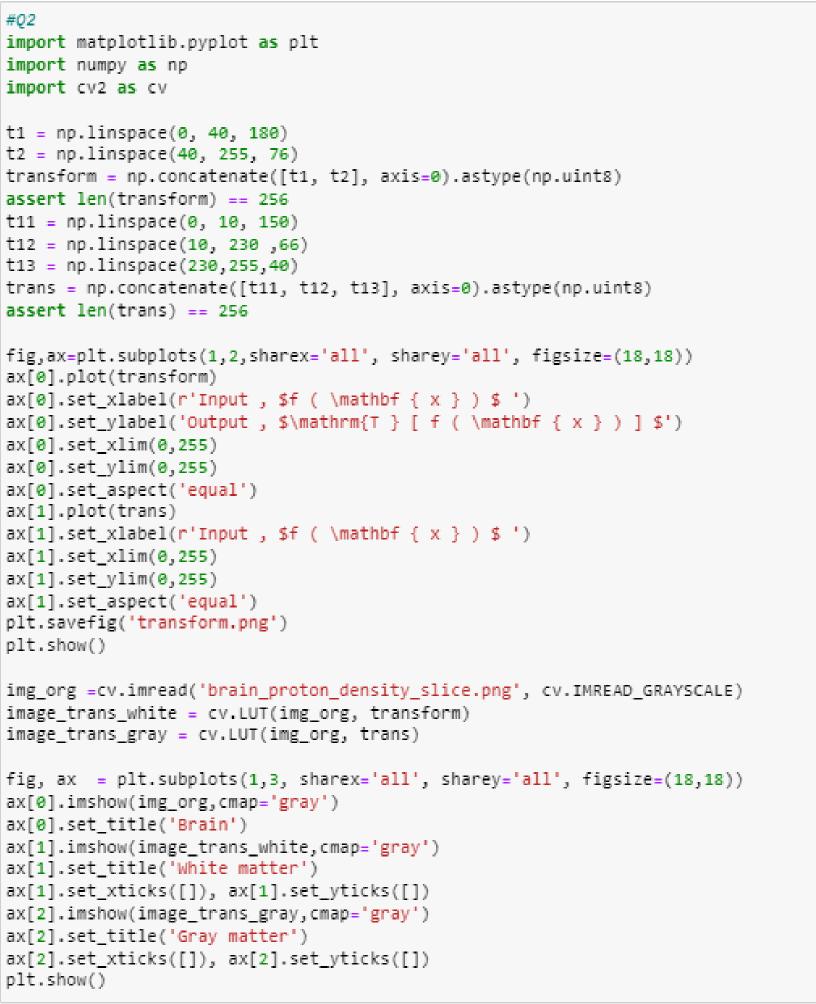
```
#Q1

%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
c= np.array([(50, 50), (50, 100) , (150, 255), (150,150) , (255,255)])
t1 = np.linspace(0, c[0,1], c[0,0] + 1-0).astype('uint8')
t2 = np.linspace(c[0, 1] + 1, c[1,1], c[1,0]-c[0,0]).astype('uint8')
t3 = np.linspace(c[1,1], c[2,1], c[2,0]-c[1,0]).astype('uint8')
t4 = np.linspace(c[2,1], c[3,1], c[3,0]-c[2,0]).astype('uint8')
t5 = np.linspace(c[3,1], c[4,1], c[4,0]-c[3,0]).astype('uint8')
transform = np.concatenate((t1,t2,t3,t4), axis=0).astype('uint8')
transform = np.concatenate((transform, t5), axis=0).astype('uint8')
fig,ax = plt.subplots()
ax.plot(transform)
ax.set_xlim (0,255)
ax.set_ylim (0,255)
ax.set_aspect('equal')
plt.grid()
plt.show()
img_orig = cv.imread('emma_gray.jpg', cv.IMREAD_GRAYSCALE)
assert img_orig is not None
image_transformed = cv.LUT(img_orig, transform)
fig, axes = plt.subplots(1,2, figsize=(8,8))
axes[0].imshow(img_orig, cmap='gray')
axes[0].set_title('Original Image')
axes[0].set_xticks([]), axes[0].set_yticks([])
axes[1].imshow(image_transformed, cmap='gray')
axes[1].set_title('Transformed Image')
axes[1].set_xticks([]), axes[1].set_yticks([])
plt.show()
```



Question2

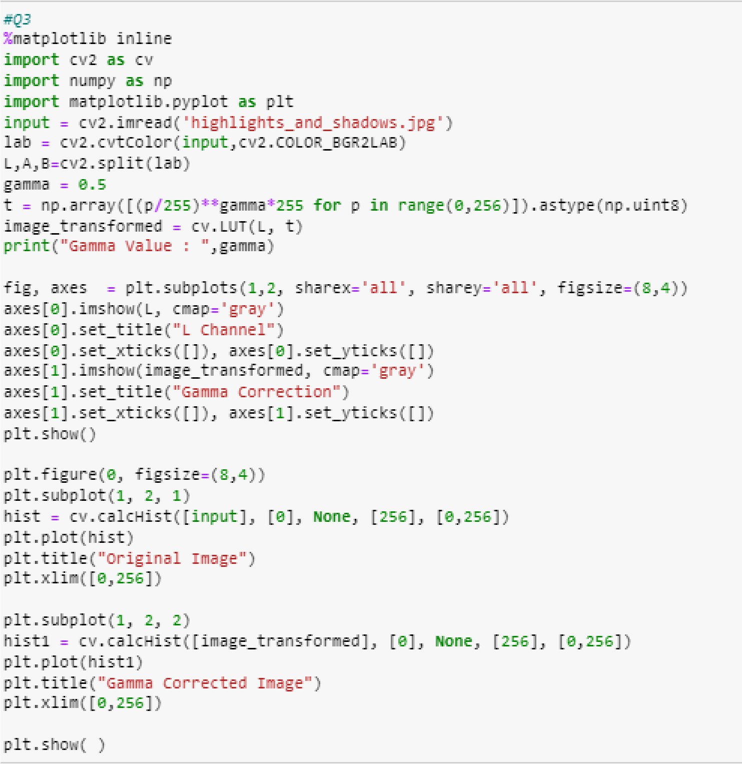




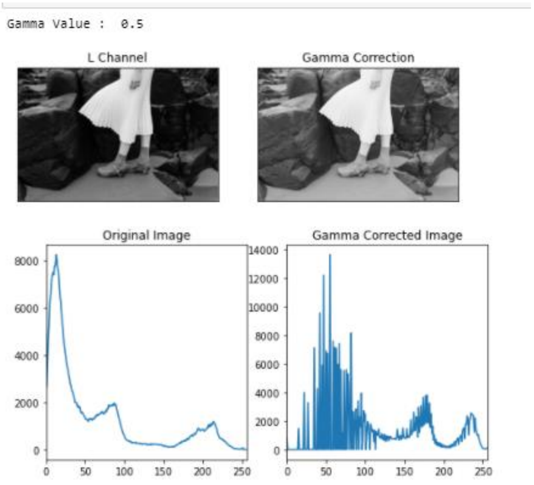


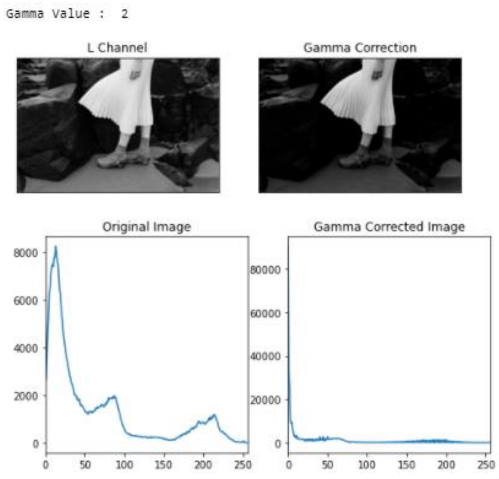
```
#Q2
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv

t1 = np.linspace(0, 40, 180)
t2 = np.linspace(40, 255, 76)
transform = np.concatenate([t1, t2], axis=0).astype(np.uint8)
assert len(transform) == 256
t11 = np.linspace(0, 10, 150)
t12 = np.linspace(10, 230 ,66)
t13 = np.linspace(230,255,40)
trans = np.concatenate([t11, t12, t13], axis=0).astype(np.uint8)
assert len(trans) == 256

fig,ax=plt.subplots(1,2,sharex='all', sharey='all', figsize=(18,18))
ax[0].plot(transform)
ax[0].set_xlabel(r'Input , $f ( \mathbf { x } ) $ ')
ax[0].set_ylabel('Output , $\mathrm{T } [ f ( \mathbf { x } ) ] $')
ax[0].set_xlim(0,255)
ax[0].set_ylim(0,255)
ax[0].set_aspect('equal')
ax[1].plot(trans)
ax[1].set_xlabel(r'Input , $f ( \mathbf { x } ) $ ')
ax[1].set_xlim(0,255)
ax[1].set_ylim(0,255)
ax[1].set_aspect('equal')
plt.savefig('transform.png')
plt.show()

img_org =cv.imread('brain_proton_density_slice.png', cv.IMREAD_GRAYSCALE)
image_trans_white = cv.LUT(img_org, transform)
image_trans_gray = cv.LUT(img_org, trans)

fig, ax  = plt.subplots(1,3, sharex='all', sharey='all', figsize=(18,18))
ax[0].imshow(img_org,cmap='gray')
ax[0].set_title('Brain')
ax[1].imshow(image_trans_white,cmap='gray')
ax[1].set_title('White matter')
ax[1].set_xticks([]), ax[1].set_yticks([])
ax[2].imshow(image_trans_gray,cmap='gray')
ax[2].set_title('Gray matter')
ax[2].set_xticks([]), ax[2].set_yticks([])
plt.show()
```

Question3



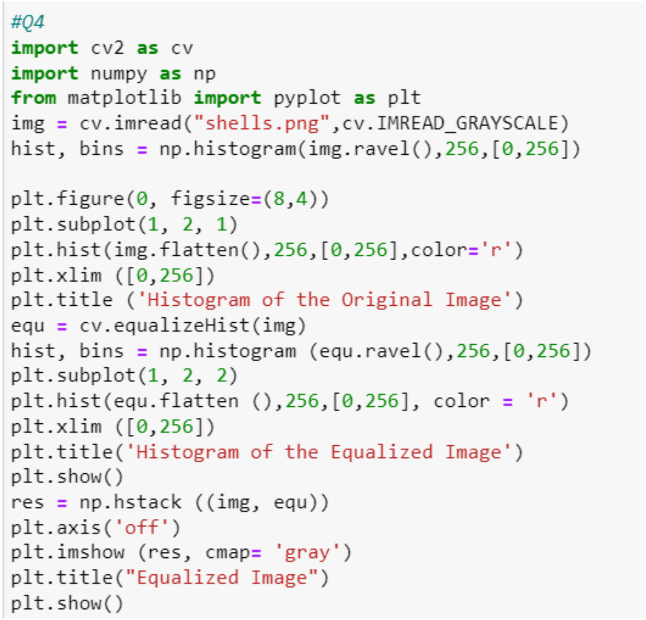
```
#Q3
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
input = cv2.imread('highlights_and_shadows.jpg')
lab = cv2.cvtColor(input,cv2.COLOR_BGR2LAB)
L,A,B=cv2.split(lab)
gamma = 0.5
t = np.array([(p/255)**gamma*255 for p in range(0,256)]).astype(np.uint8)
image_transformed = cv.LUT(L, t)
print("Gamma Value : ",gamma)

fig, axes  = plt.subplots(1,2, sharex='all', sharey='all', figsize=(8,4))
axes[0].imshow(L, cmap='gray')
axes[0].set_title("L Channel")
axes[0].set_xticks([]), axes[0].set_yticks([])
axes[1].imshow(image_transformed, cmap='gray')
axes[1].set_title("Gamma Correction")
axes[1].set_xticks([]), axes[1].set_yticks([])
plt.show()

plt.figure(0, figsize=(8,4))
plt.subplot(1, 2, 1)
hist = cv.calcHist([input], [0], None, [256], [0,256])
plt.plot(hist)
plt.title("Original Image")
plt.xlim([0,256])

plt.subplot(1, 2, 2)
hist1 = cv.calcHist([image_transformed], [0], None, [256], [0,256])
plt.plot(hist1)
plt.title("Gamma Corrected Image")
plt.xlim([0,256])

plt.show( )
```

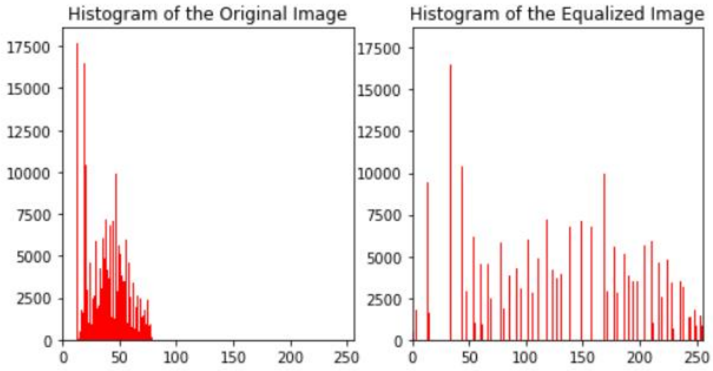

```
Gamma Value :  0.5
```

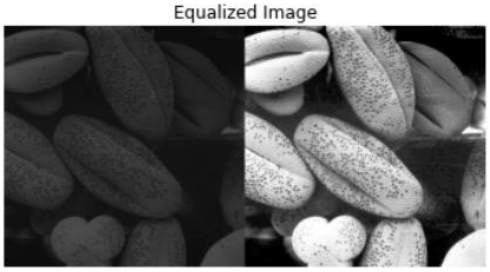

```
Gamma Value :  2
```

- Brightness of an image is controlled by Gamma correction.
- With the increase of Gamma value images darkness of image increases.

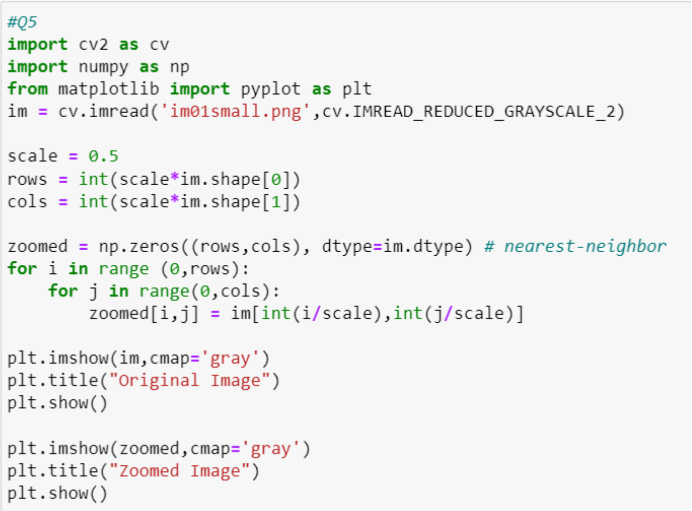
Question4


```
#Q4
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread("shells.png",cv.IMREAD_GRAYSCALE)
hist, bins = np.histogram(img.ravel(),256,[0,256])

plt.figure(0, figsize=(8,4))
plt.subplot(1, 2, 1)
plt.hist(img.flatten(),256,[0,256],color='r')
plt.xlim ([0,256])
plt.title ('Histogram of the Original Image')
equ = cv.equalizeHist(img)
hist, bins = np.histogram (equ.ravel(),256,[0,256])
plt.subplot(1, 2, 2)
plt.hist(equ.flatten (),256,[0,256], color = 'r')
plt.xlim ([0,256])
plt.title('Histogram of the Equalized Image')
plt.show()
res = np.hstack ((img, equ))
plt.axis('off')
plt.imshow (res, cmap= 'gray')
plt.title("Equalized Image")
plt.show()
```

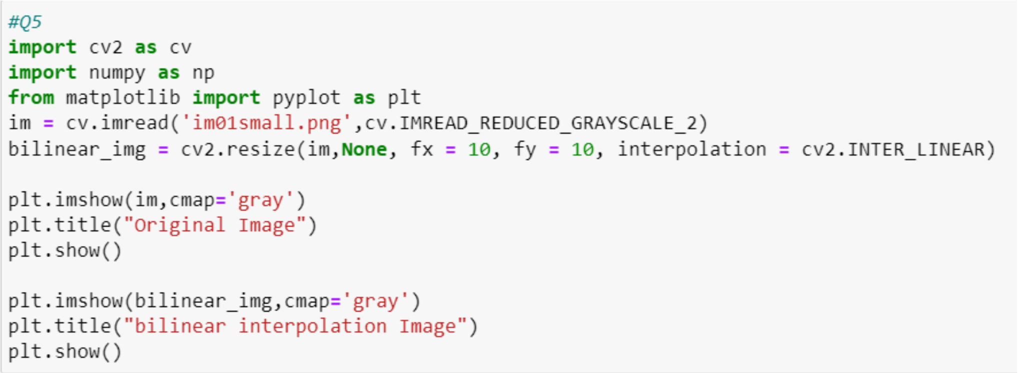




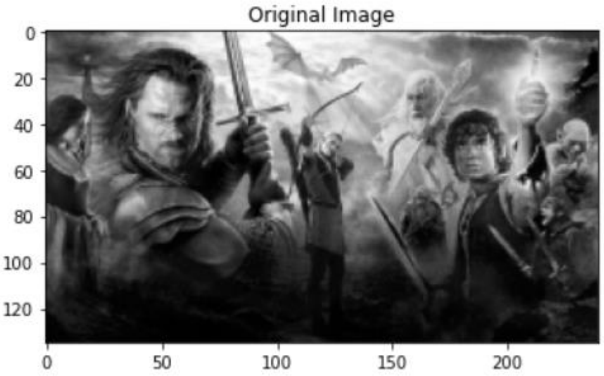
Question5

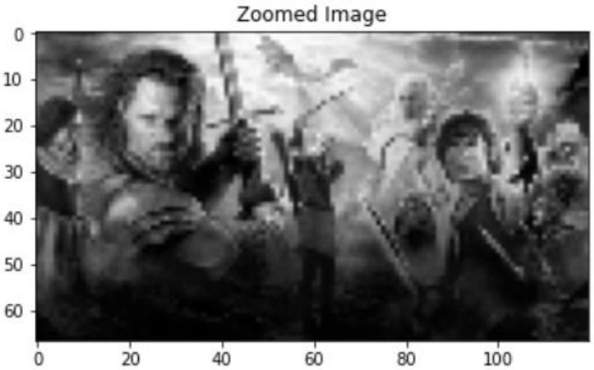


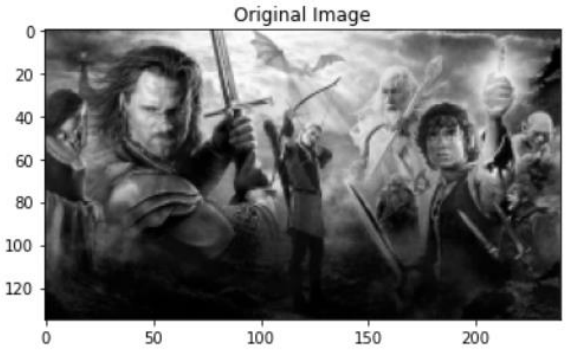
```
#Q5
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
im = cv.imread('im01small.png',cv.IMREAD_REDUCED_GRAYSCALE_2)

scale = 0.5
rows = int(scale*im.shape[0])
cols = int(scale*im.shape[1])

zoomed = np.zeros((rows,cols), dtype=im.dtype) # nearest-neighbor
for i in range (0,rows):
    for j in range(0,cols):
        zoomed[i,j] = im[int(i/scale),int(j/scale)]

plt.imshow(im,cmap='gray')
plt.title("Original Image")
plt.show()

plt.imshow(zoomed,cmap='gray')
plt.title("Zoomed Image")
plt.show()
```

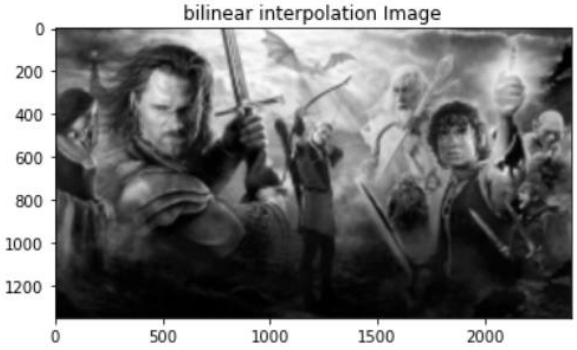


```
#Q5
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
im = cv.imread('im01small.png',cv.IMREAD_REDUCED_GRAYSCALE_2)
bilinear_img = cv2.resize(im,None, fx = 10, fy = 10, interpolation = cv2.INTER_LINEAR)

plt.imshow(im,cmap='gray')
plt.title("Original Image")
plt.show()

plt.imshow(bilinear_img,cmap='gray')
plt.title("bilinear interpolation Image")
plt.show()
```

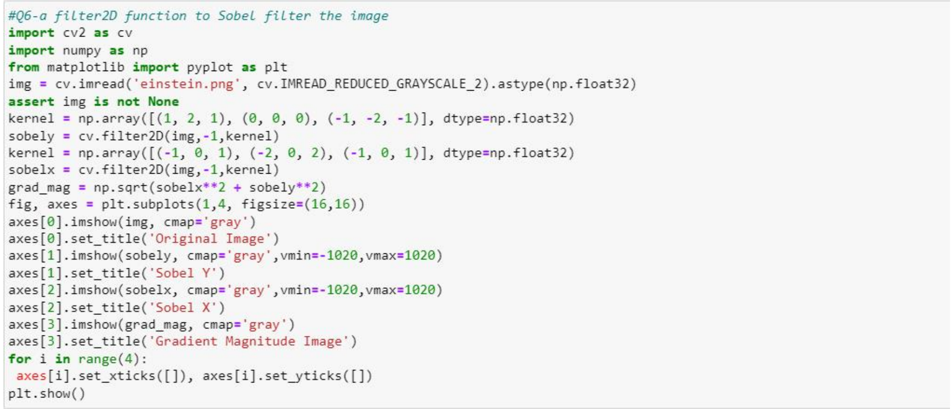




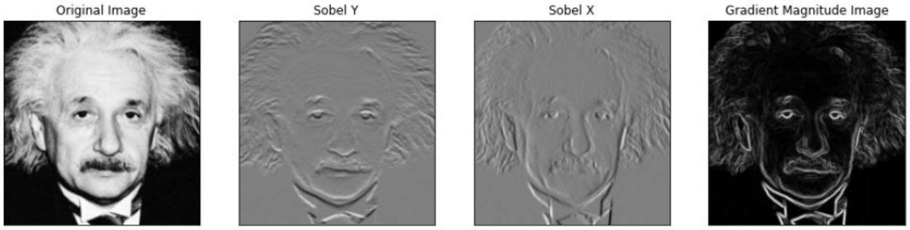




Question6



```
#Q6-a filter2D function to Sobel filter the image
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('einstein.png', cv.IMREAD_REDUCED_GRAYSCALE_2).astype(np.float32)
assert img is not None
kernel = np.array([(1, 2, 1), (0, 0, 0), (-1, -2, -1)], dtype=np.float32)
sobely = cv.filter2D(img,-1,kernel)
kernel = np.array([(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)], dtype=np.float32)
sobelx = cv.filter2D(img,-1,kernel)
grad_mag = np.sqrt(sobelx**2 + sobely**2)
fig, axes = plt.subplots(1,4, figsize=(16,16))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original Image')
axes[1].imshow(sobely, cmap='gray',vmin=-1020,vmax=1020)
axes[1].set_title('Sobel Y')
axes[2].imshow(sobelx, cmap='gray',vmin=-1020,vmax=1020)
axes[2].set_title('Sobel X')
axes[3].imshow(grad_mag, cmap='gray')
axes[3].set_title('Gradient Magnitude Image')
for i in range(4):
 axes[i].set_xticks([]), axes[i].set_yticks([])
plt.show()
```

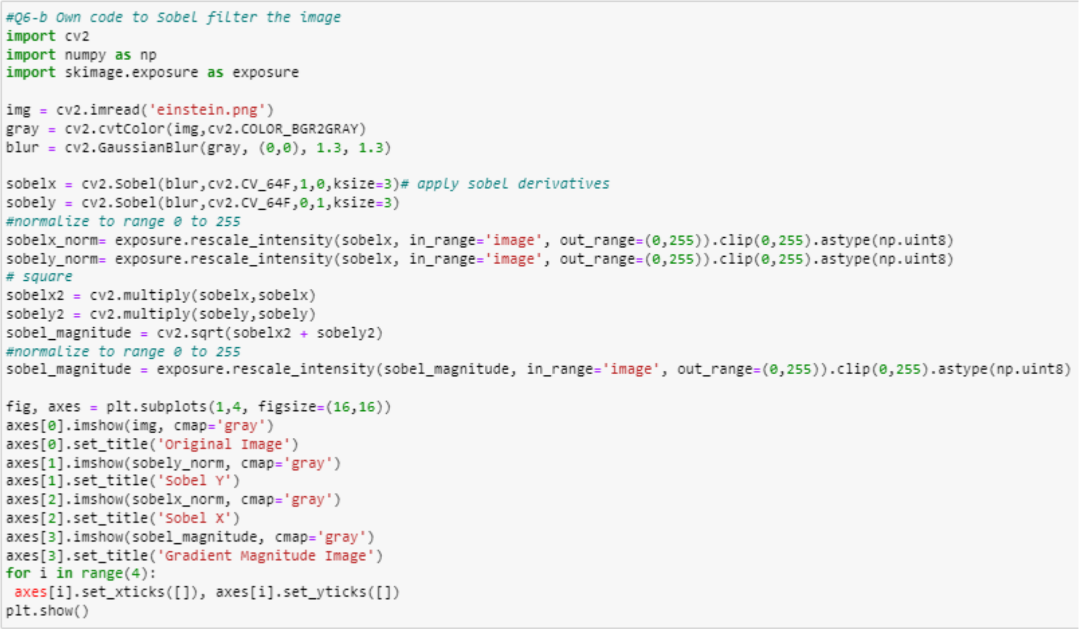


Original Image

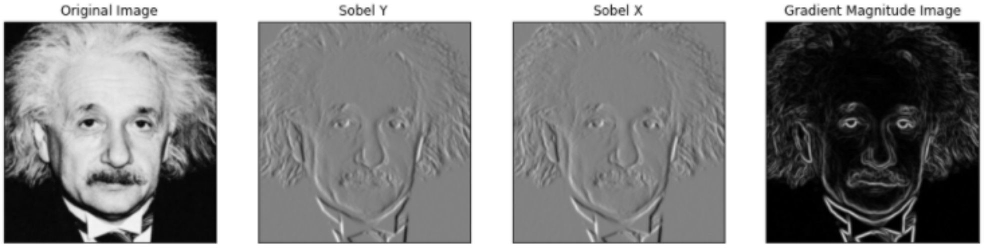
Sobel Y

Sobel X

Gradient Magnitude Image



```
#Q6-b Own code to Sobel filter the image
import cv2
import numpy as np
import skimage.exposure as exposure

img = cv2.imread('einstein.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (0,0), 1.3, 1.3)

sobelx = cv2.Sobel(blur,cv2.CV_64F,1,0,ksize=3)# apply sobel derivatives
sobely = cv2.Sobel(blur,cv2.CV_64F,0,1,ksize=3)
#normalize to range 0 to 255
sobelx_norm= exposure.rescale_intensity(sobelx, in_range='image', out_range=(0,255)).clip(0,255).astype(np.uint8)
sobely_norm= exposure.rescale_intensity(sobelx, in_range='image', out_range=(0,255)).clip(0,255).astype(np.uint8)
# square
sobelx2 = cv2.multiply(sobelx,sobelx)
sobely2 = cv2.multiply(sobely,sobely)
sobel_magnitude = cv2.sqrt(sobelx2 + sobely2)
#normalize to range 0 to 255
sobel_magnitude = exposure.rescale_intensity(sobel_magnitude, in_range='image', out_range=(0,255)).clip(0,255).astype(np.uint8)

fig, axes = plt.subplots(1,4, figsize=(16,16))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original Image')
axes[1].imshow(sobely_norm, cmap='gray')
axes[1].set_title('Sobel Y')
axes[2].imshow(sobelx_norm, cmap='gray')
axes[2].set_title('Sobel X')
axes[3].imshow(sobel_magnitude, cmap='gray')
axes[3].set_title('Gradient Magnitude Image')
for i in range(4):
 axes[i].set_xticks([]), axes[i].set_yticks([])
plt.show()
```

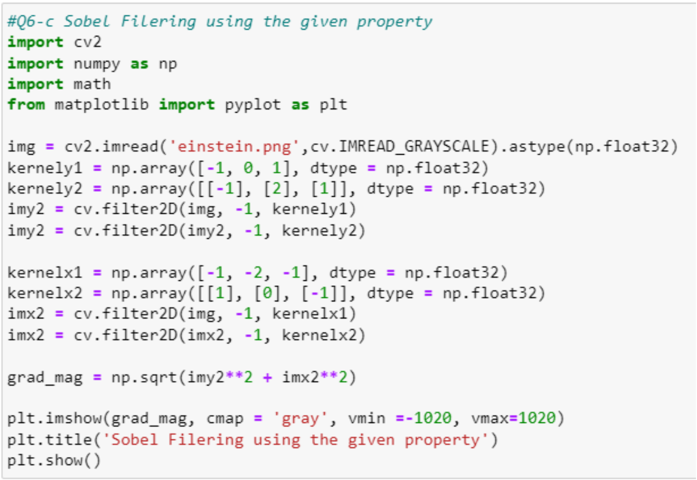


Original Image

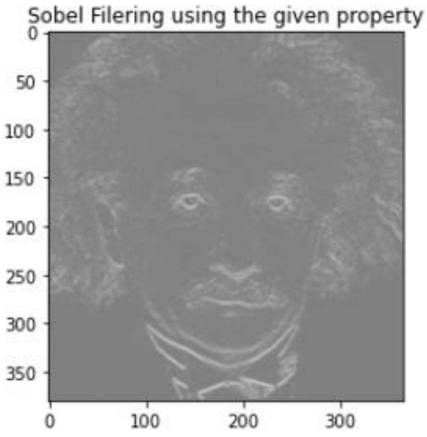
Sobel Y

Sobel X

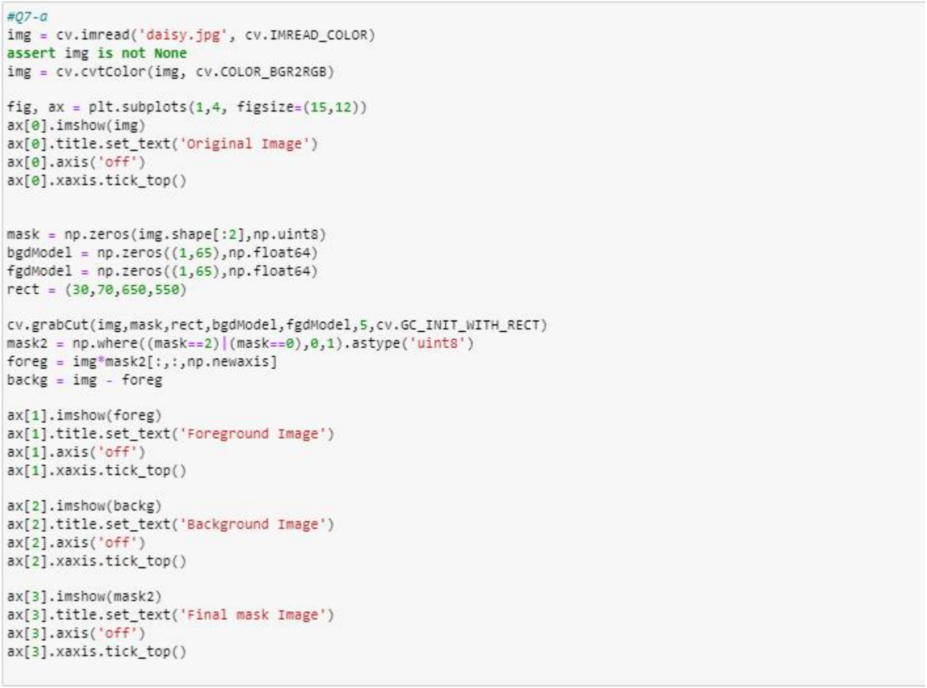
Gradient Magnitude Image



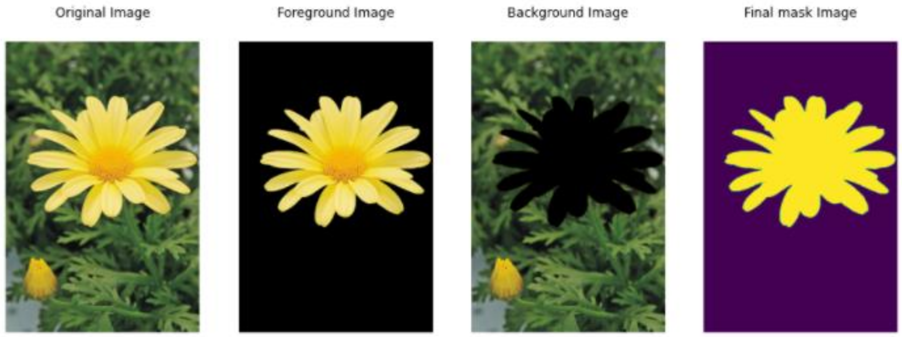
```
#Q6-c Sobel Filering using the given property
import cv2
import numpy as np
import math
from matplotlib import pyplot as plt

img = cv2.imread('einstein.png',cv.IMREAD_GRAYSCALE).astype(np.float32)
kernely1 = np.array([-1, 0, 1], dtype = np.float32)
kernely2 = np.array([[-1], [2], [1]], dtype = np.float32)
imy2 = cv.filter2D(img, -1, kernely1)
imy2 = cv.filter2D(imy2, -1, kernely2)

kernelx1 = np.array([-1, -2, -1], dtype = np.float32)
kernelx2 = np.array([[1], [0], [-1]], dtype = np.float32)
imx2 = cv.filter2D(img, -1, kernelx1)
imx2 = cv.filter2D(imx2, -1, kernelx2)

grad_mag = np.sqrt(imy2**2 + imx2**2)

plt.imshow(grad_mag, cmap = 'gray', vmin =-1020, vmax=1020)
plt.title('Sobel Filering using the given property')
plt.show()
```

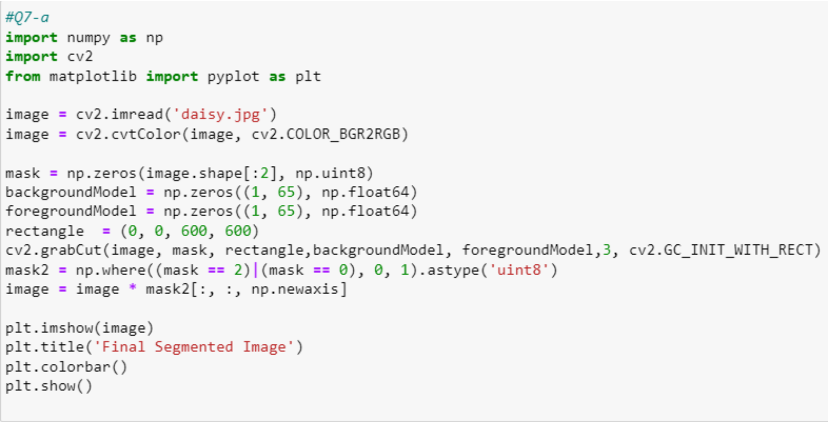


Question7

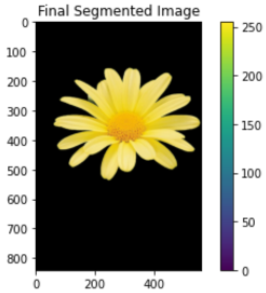


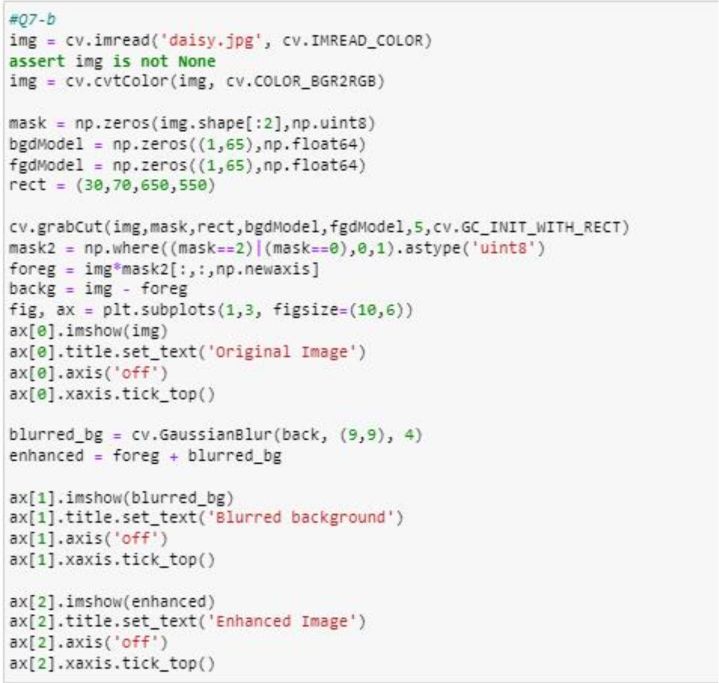
```
#Q7-a
img = cv.imread('daisy.jpg', cv.IMREAD_COLOR)
assert img is not None
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,4, figsize=(15,12))
ax[0].imshow(img)
ax[0].title.set_text('Original Image')
ax[0].axis('off')
ax[0].xaxis.tick_top()


mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (30,70,650,550)

cv.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
foreg = img*mask2[:,:,np.newaxis]
backg = img - foreg

ax[1].imshow(foreg)
ax[1].title.set_text('Foreground Image')
ax[1].axis('off')
ax[1].xaxis.tick_top()

ax[2].imshow(backg)
ax[2].title.set_text('Background Image')
ax[2].axis('off')
ax[2].xaxis.tick_top()

ax[3].imshow(mask2)
ax[3].title.set_text('Final mask Image')
ax[3].axis('off')
ax[3].xaxis.tick_top()
```



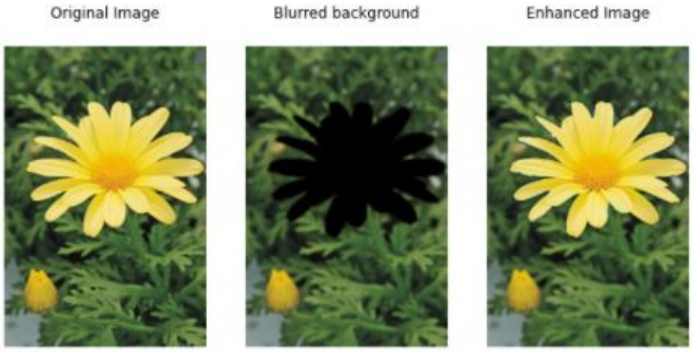


```
#Q7-a
import numpy as np
import cv2
from matplotlib import pyplot as plt

image = cv2.imread('daisy.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

mask = np.zeros(image.shape[:2], np.uint8)
backgroundModel = np.zeros((1, 65), np.float64)
foregroundModel = np.zeros((1, 65), np.float64)
rectangle  = (0, 0, 600, 600)
cv2.grabCut(image, mask, rectangle,backgroundModel, foregroundModel,3, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2)|(mask == 0), 0, 1).astype('uint8')
image = image * mask2[:, :, np.newaxis]

plt.imshow(image)
plt.title('Final Segmented Image')
plt.colorbar()
plt.show()
```





```
#Q7-b
img = cv.imread('daisy.jpg', cv.IMREAD_COLOR)
assert img is not None
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (30,70,650,550)

cv.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
foreg = img*mask2[:,:,np.newaxis]
backg = img - foreg
fig, ax = plt.subplots(1,3, figsize=(10,6))
ax[0].imshow(img)
ax[0].title.set_text('Original Image')
ax[0].axis('off')
ax[0].xaxis.tick_top()

blurred_bg = cv.GaussianBlur(back, (9,9), 4)
enhanced = foreg + blurred_bg

ax[1].imshow(blurred_bg)
ax[1].title.set_text('Blurred background')
ax[1].axis('off')
ax[1].xaxis.tick_top()

ax[2].imshow(enhanced)
ax[2].title.set_text('Enhanced Image')
ax[2].axis('off')
ax[2].xaxis.tick_top()
```



c) when using Gaussian kernel for blurring the image, background near the edge of the flower is affected by the darker pixels in the image. There fore in the blurred background, flower is replaced by dark pixels.

GitHub Profile: https://github.com/Gajaan08/FUNDAMENTALS-OF-IMAGE-PROCESSING.git