Assignment 01

190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

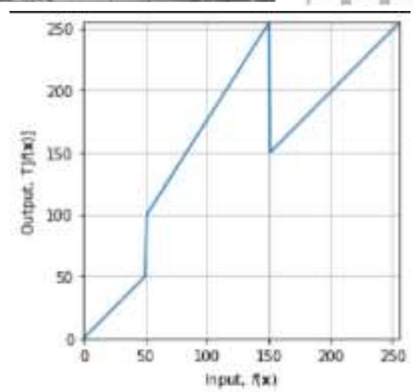
Intensity Transformations and Neighborhood Filtering

The libraries used for this Assignment

```python
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np
%matplotlib inline
import random as rand
from skimage.exposure import rescale_intensity
```

1.

```python
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_xlabel(r'Input, $f(\mathbf {x})$')
ax.set_ylabel('Output, $\mathrm{T}]f(\mathbf {x})]$')
ax.set_xlim (0,255)
ax.set_ylim (0,255)
ax.set_aspect('equal')
plt. savefig('transform.png')
plt.grid()
plt.show()
img_orig = cv.imread('emma_gray.jpg', cv.IMREAD_GRAYSCALE)
image_transformed = cv.LUT(img_orig, transform)


# cv.imshow("Image",image_transformed)
# cv.waitKey(0)
# cv.destroyAllWindows()

fig, ax  = plt.subplots(1,2, sharex='all', sharey='all', figsize=(18,18))
ax[0].imshow(img_orig,cmap='gray')
ax[0].set_title('original image')
ax[1].imshow(image_transformed,cmap='gray')
ax[1].set_title('Transformed image white matter')
```

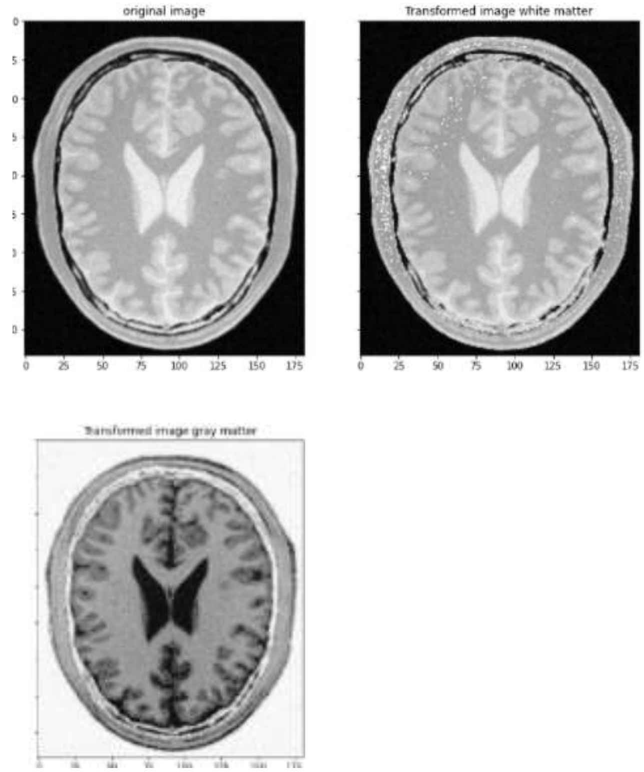


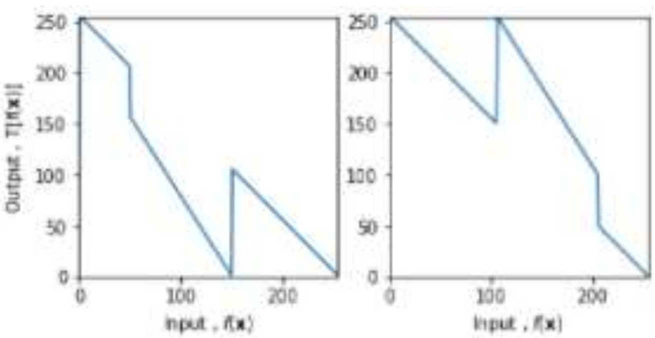
Assignment 01

190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

2.

```python
c=np.array([(50,50),(50,100),(150,150)])

t1=np.linspace(0,c[0,1],c[0,0]+1).astype('uint8')
t2=np.linspace(c[1,1]+1,255,c[2,0]-c[1,0]).astype('uint8')
t3=np.linspace(c[2,1]+1,255,255-c[2,0]).astype('uint8')

transform= np.concatenate((t1,t2),axis=0).astype('uint8')
transform= np.concatenate((transform,t3),axis=0).astype('uint8')

c=np.array([(105,150),(205,100),(205,50)])

t_1=np.linspace(255,c[0,1],c[0,0]+1).astype('uint8')
t_2=np.linspace(255,c[1,1],c[1,0]-c[0,0]).astype('uint8')
t_3=np.linspace(c[2,1],0,c[1,1]-c[2,1]).astype('uint8')


t_w= np.concatenate((t_1,t_2),axis=0).astype('uint8')
t_w= np.concatenate((t_w,t_3),axis=0).astype('uint8')

fig,ax=plt.subplots(1,2)
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(t_w)
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, t_w)


fig, ax  = plt.subplots(1,3, sharex='all', sharey='all', figsize=(18,18))
ax[0].imshow(img_org,cmap='gray')
ax[0].set_title('original image')
ax[1].imshow(image_trans_white,cmap='gray')
ax[1].set_title('Transformed image white matter')
ax[2].imshow(image_trans_gray,cmap='gray')
ax[2].set_title('Transformed image gray matter')
```



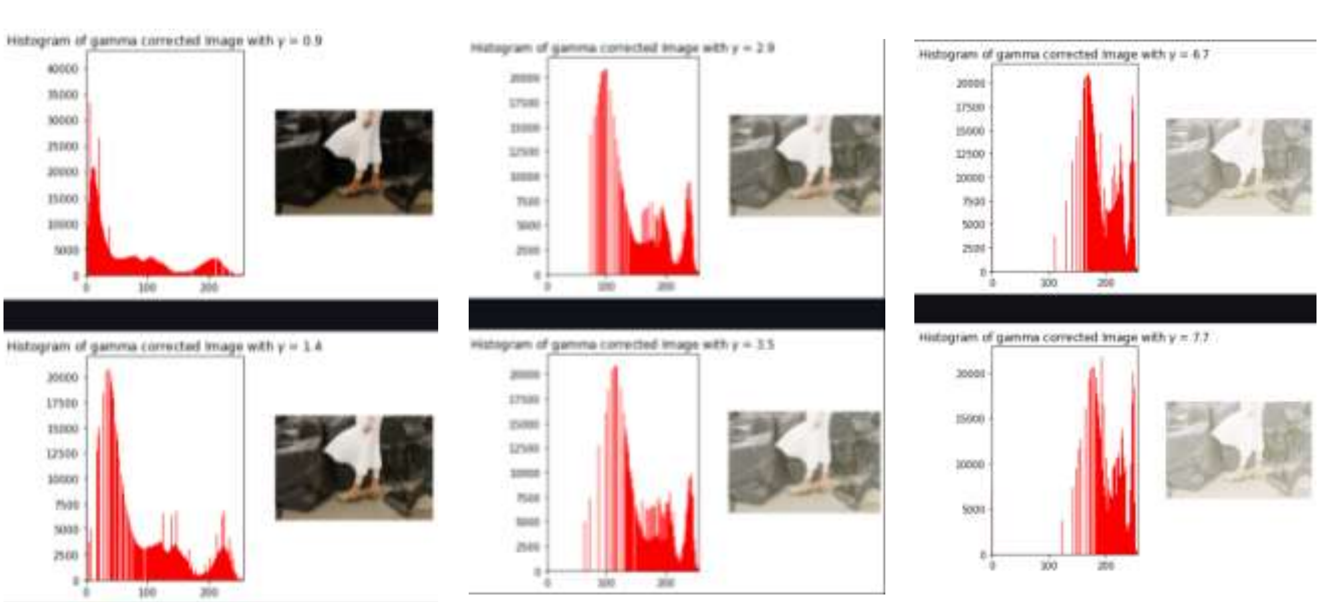


Assignment 01
190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

3.

```python
def gammaCorrection(src, gamma):
    invGamma = 1 / gamma

    table = [((i / 255) ** invGamma) * 255 for i in range(256)]
    table = np.array(table, np.uint8)

    return cv.LUT(src, table)
```

```python
img = cv.imread('highlights_and_shadows.jpg')

for i in range(10):
    gamma = i+rand.randint(0,9)*0.1
    gammaImg = gammaCorrection(img,gamma)
    fig,ax=plt.subplots(1,2)
    ax[0].hist(gammaImg.flatten(),256,[0,256],color = 'r')
    ax[0].set_xlim([0,256])
    ax[0].set_title('Histogram of gamma corrected Image with $\gamma$ = {0}'.format(gamma))
    ax[1].imshow(cv.cvtColor(gammaImg,cv.COLOR_BGR2RGB))
    ax[1].axis('off')
    plt.show()
```

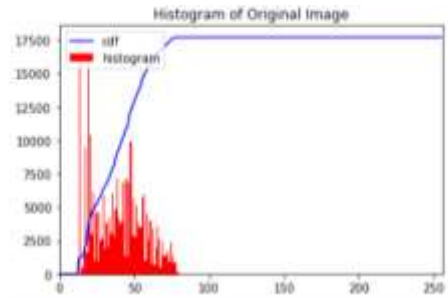


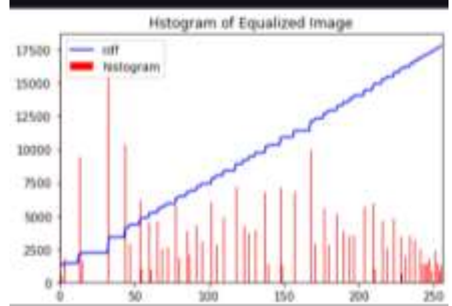
The images are created with the different gamma values. We can understand the result of the change in gamma value in images. As gamma increases the histogram's area moves towards right side.

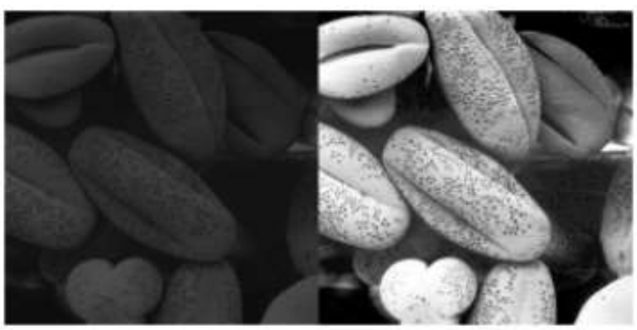
Assignment 01
190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

4.

```
hist,bins = np.histogram(img.ravel(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf*hist.max()/cdf.max()
plt.plot(cdf_normalized, color ='b')
plt.hist(img.flatten(),256,[0,256],color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'),loc = 'upper left')
plt.title('Histogram of Original Image')
plt.show()
img=cv.imread('shells.png',cv.IMREAD_GRAYSCALE)
equ = cv.equalizeHist(img)
hist,bins = np.histogram(equ.ravel(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized =cdf*hist.max()/cdf.max()
plt.plot(cdf_normalized ,color = 'b')
plt.hist(equ.flatten(),356,[0,256],color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'),loc = 'upper left')
plt.title('Hstogram of Equalized Image')
plt.show()
res = np.hstack((img,equ))
plt.axis('off')
plt.imshow(res,cmap='gray')
```



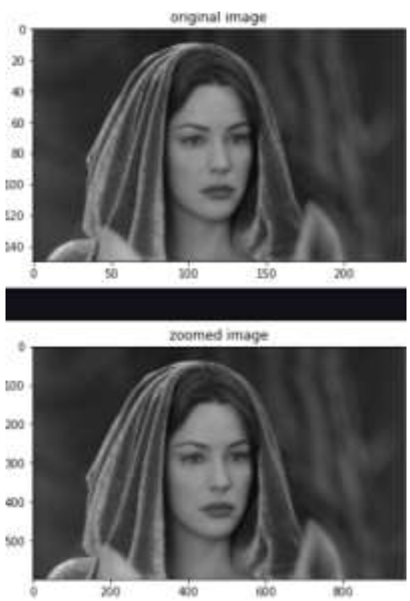




Assignment 01
190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

5.

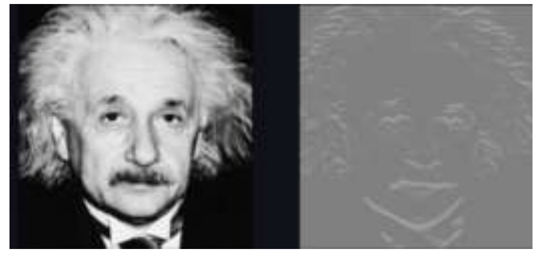
```python
im = cv.imread('im01small.png',cv.IMREAD_REDUCED_GRAYSCALE_2)

scale = 4
rows = int(scale*im.shape[0])
cols= int(scale*im.shape[1])

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


plt.imshow(cv.cvtColor(im,cv.COLOR_RGB2BGR))
plt.title("original image")
plt.show()
plt.imshow(cv.cvtColor(zoomed,cv.COLOR_RGB2BGR))
plt.title("zoomed image")
plt.show()
```



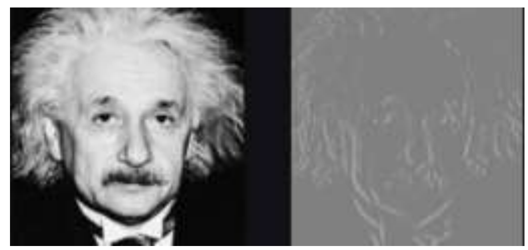
Assignment 01
190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

6. With the 2dfilter

```
img = cv.imread('einstein.png', cv.IMREAD_REDUCED_GRAYSCALE_2)

# Sobel vertical
sobel_ver_kernel = np.array([(-1, -2, -1), (0, 0, 0), (1, 2, 1)], dtype='float32')
img_x = cv.filter2D(img,-1,sobel_ver_kernel)


fig, axes  = plt.subplots(1,2, sharex='all', sharey='all', figsize=(18,18))


axes[0].imshow(img, cmap='gray',vmin=0,vmax=255)
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])
axes[1].imshow(img_x, cmap='gray',vmin=-1020,vmax=1020)
axes[1].set_title('Sobel Vertical')
axes[1].set_xticks([]), axes[1].set_yticks([])


plt.show()
```

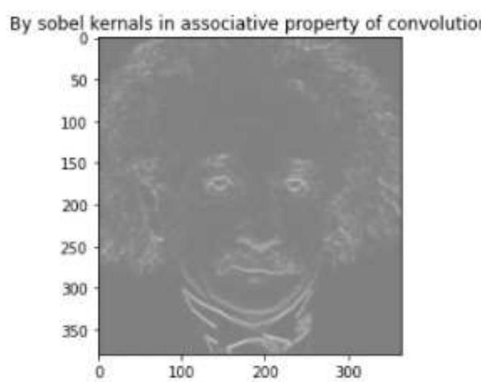


```
img = cv.imread('einstein.png', cv.IMREAD_REDUCED_GRAYSCALE_2)

# Sobel vertical
sobel_ver_kernel = np.array([(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)], dtype='float32')
img_x = cv.filter2D(img,-1,sobel_ver_kernel)


fig, axes  = plt.subplots(1,2, sharex='all', sharey='all', figsize=(18,18))


axes[0].imshow(img, cmap='gray',vmin=0,vmax=255)
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])
axes[1].imshow(img_x, cmap='gray',vmin=-1020,vmax=1020)
axes[1].set_title('Sobel Vertical')
axes[1].set_xticks([]), axes[1].set_yticks([])


plt.show()
```



By Associative property of Convolution

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




kernel_y1 = np.array([-1, 0, 1], dtype = np.float32)
kernel_y2 = np.array([[-1], [2], [1]], dtype = np.float32)
image_y2 = cv.filter2D(einstein, -1, kernel_y1)
image_y2 = cv.filter2D(image_y2, -1, kernel_y2)

kernel_x1 = np.array([-1, -2, -1], dtype = np.float32)
kernel_x2 = np.array([[1], [0], [-1]], dtype = np.float32)
image_x2 = cv.filter2D(einstein, -1, kernel_x1)
image_x2 = cv.filter2D(image_x2, -1, kernel_x2)

imag2 = np.sqrt(image_y2**2 + image_x2**2)

plt.show()
plt.imshow(imag2, cmap = 'gray', vmin =-1020, vmax=1020)
plt.title('By sobel kernals in associative property of convolution')
```

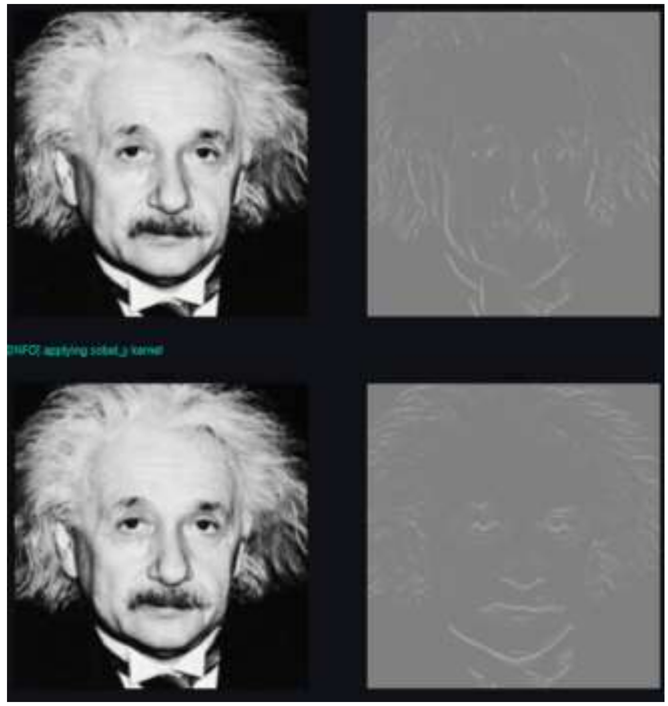


Assignment 01

190428D https://github.com/pankajan-T/Image-Processing-Fundamental.git

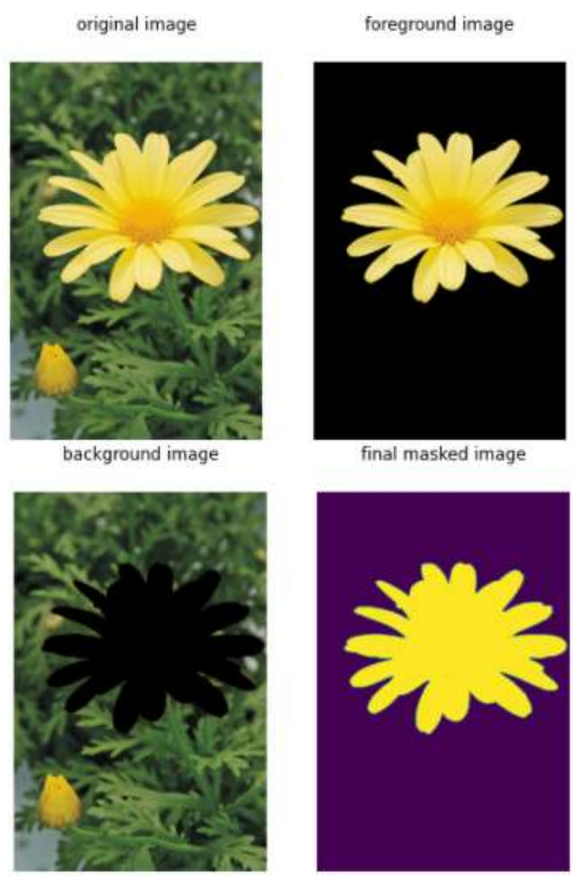
By direct convolution method

```python
def convolve(image, kernel):
    (iH, iW) = image.shape[:2]
    (kH, kW) = kernel.shape[:2]
    pad = (kW - 1) // 2
    image = cv.copyMakeBorder(image, pad, pad, pad, pad,cv.BORDER_REPLICATE)
    output = np.zeros((iH, iW), dtype="float32")
    for y in np.arange(pad, iH + pad):
        for x in np.arange(pad, iW + pad):

            roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]

            k = (roi * kernel).sum()

            output[y - pad, x - pad] = k
    output = rescale_intensity(output, in_range=(0, 255))
    output = (output * 255).astype("uint8")
    # return the output image
    return output


sobelX = np.array((
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]), dtype="int")
# construct the Sobel y-axis kernel
sobelY = np.array((
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]), dtype="int")

kernelBank = (

    ("sobel_x", sobelX),
    ("sobel_y", sobelY)
)

image = cv.imread("einstein.png")
gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
# loop over the kernels
for (kernelName, kernel) in kernelBank:
    print("[INFO] applying {} kernel".format(kernelName))
    convoleOutput = convolve(gray, kernel)
```

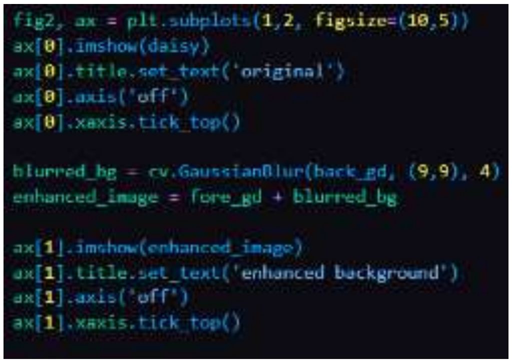


7. a.

```python
daisy = cv.imread(r'daisy.jpg', cv.IMREAD_COLOR)
daisy = cv.cvtColor(daisy, cv.COLOR_BGR2RGB)

mask = np.zeros(daisy.shape[:2],np.uint8)
back_gdModel = np.zeros((1,65),np.float64)
fore_gdModel = np.zeros((1,65),np.float64)
rect = (30,70,650,550)

cv.grabCut(daisy,mask,rect,back_gdModel,fore_gdModel,5,cv.GC_INIT_WITH_RECT)
mask_2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
fore_gd = daisy*mask_2[:,:,np.newaxis]
back_gd = daisy - fore_gd

fig, ax = plt.subplots(1,4, figsize=(15,10))


ax[0].imshow(daisy)
ax[0].title.set_text('original image')
ax[0].axis('off')
ax[0].xaxis.tick_top()

ax[1].imshow(fore_gd)
ax[1].title.set_text('foreground image')
ax[1].axis('off')
ax[1].xaxis.tick_top()

ax[2].imshow(back_gd)
ax[2].title.set_text('background image')
ax[2].axis('off')
ax[2].xaxis.tick_top()

ax[3].imshow(mask_2)
ax[3].title.set_text('final masked image')
ax[3].axis('off')
ax[3].xaxis.tick_top()
```



b.



```
fig2, ax = plt.subplots(1,2, figsize=(10,5))
ax[0].imshow(daisy)
ax[0].title.set_text('original')
ax[0].axis('off')
ax[0].xaxis.tick_top()

blurred_bg = cv.GaussianBlur(back_gd, (9,9), 4)
enhanced_image = fore_gd + blurred_bg

ax[1].imshow(enhanced_image)
ax[1].title.set_text('enhanced background')
ax[1].axis('off')
ax[1].xaxis.tick_top()
```

c.

For this que we can take the background image of 31th cell the daisy flower is replaced by black or dark palate that covers all the flower. when we apply the gaussian blur effect to this image the neighborhood of the edge of the flower is influenced by these dark pixels causing such effect in the last enhanced image(even the gaussian blurred background image will have this same effect)