Electronic and Telecommunication engineering

University of Moratuwa

EN3160 - Image Processing and Machine Vision

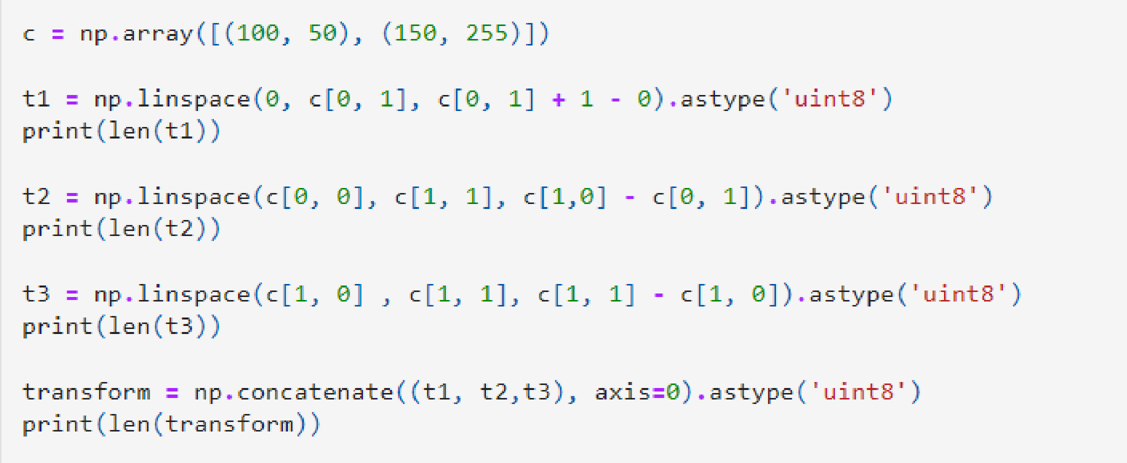
A01 Intensity Transformations and Neighborhood Filtering

RANAVIRAJA R.W.H.M.T.A. - Index No. 200520X

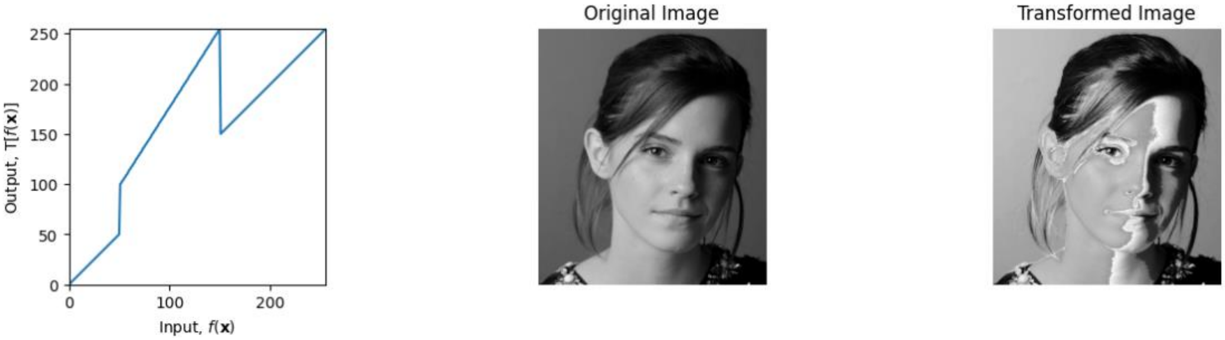
GitHub: GitHub

20th August, 2023

Q1.

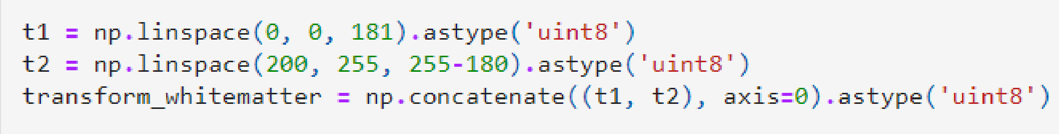


```
c = np.array([(100, 50), (150, 255)])

t1 = np.linspace(0, c[0, 1], c[0, 1] + 1 - 0).astype('uint8')
print(len(t1))

t2 = np.linspace(c[0, 0], c[1, 1], c[1,0] - c[0, 1]).astype('uint8')
print(len(t2))

t3 = np.linspace(c[1, 0] , c[1, 1], c[1, 1] - c[1, 0]).astype('uint8')
print(len(t3))

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

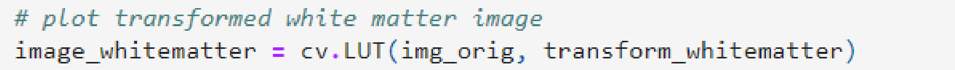


Q2.

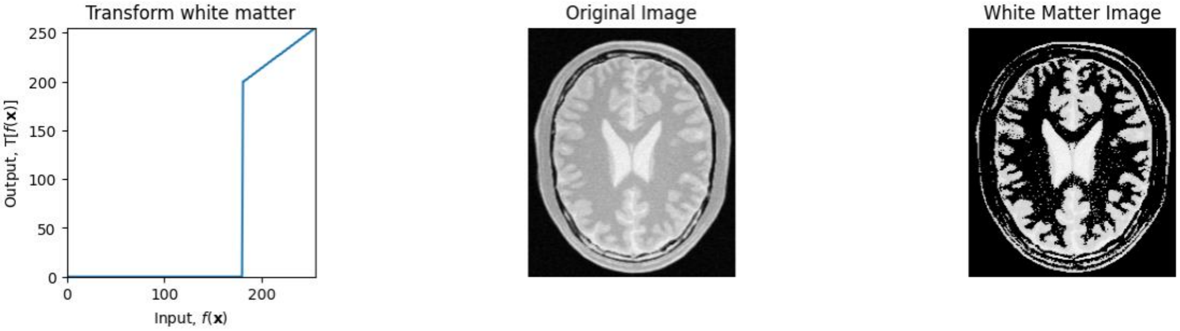
(a)White matter



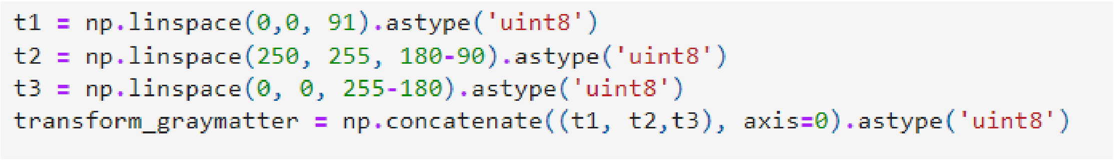
```
t1 = np.linspace(0, 0, 181).astype('uint8')
t2 = np.linspace(200, 255, 255-180).astype('uint8')
transform_whitematter = np.concatenate((t1, t2), axis=0).astype('uint8')
```



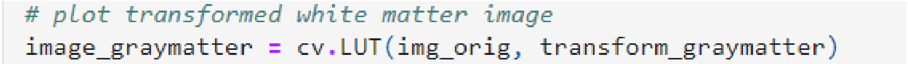
```
# plot transformed white matter image
image_whitematter = cv.LUT(img_orig, transform_whitematter)
```



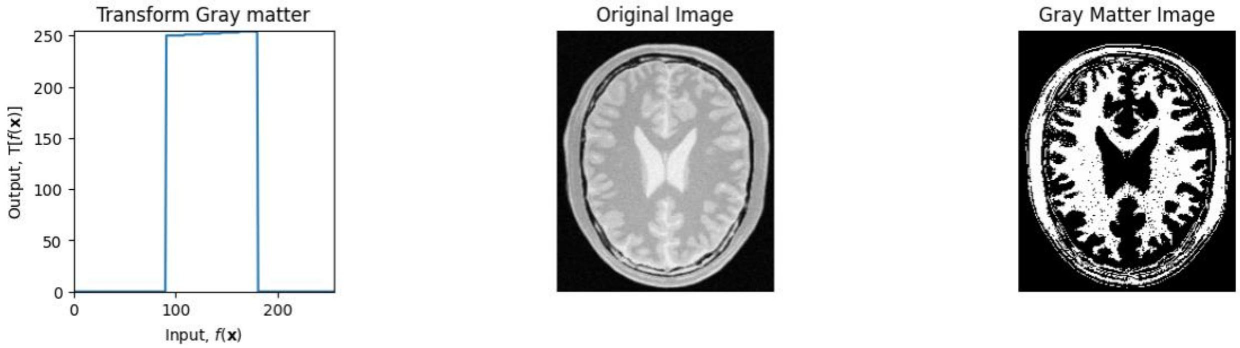
(b)Gray matter



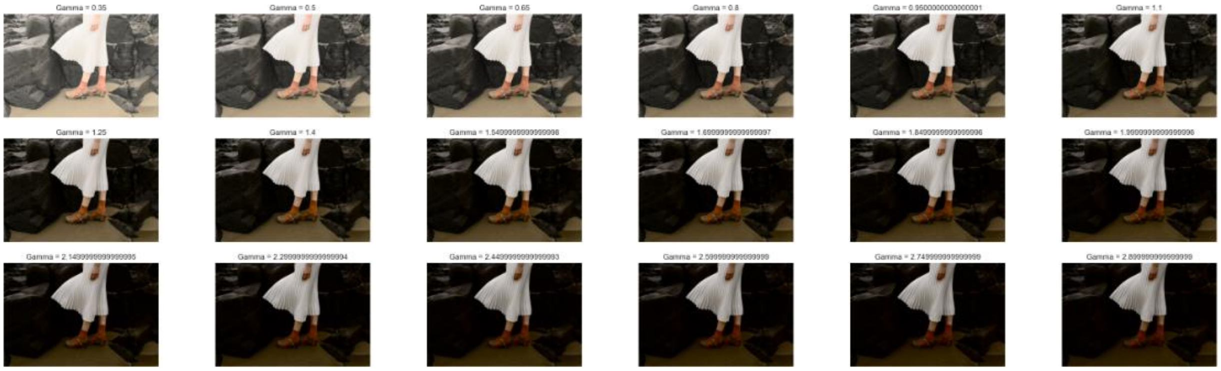
```
t1 = np.linspace(0,0, 91).astype('uint8')
t2 = np.linspace(250, 255, 180-90).astype('uint8')
t3 = np.linspace(0, 0, 255-180).astype('uint8')
transform_graymatter = np.concatenate((t1, t2,t3), axis=0).astype('uint8')
```

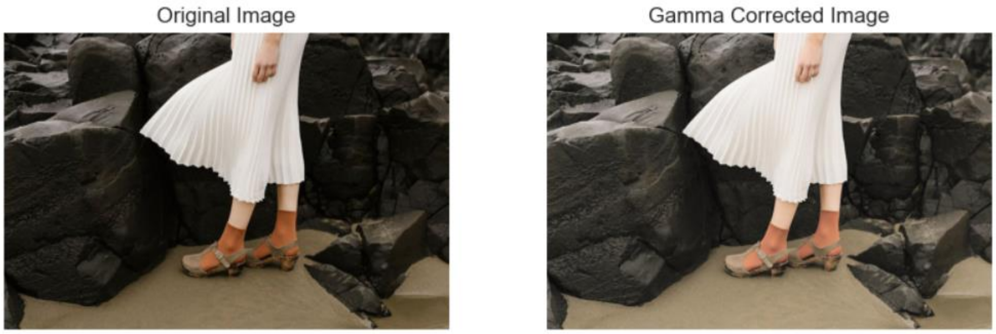


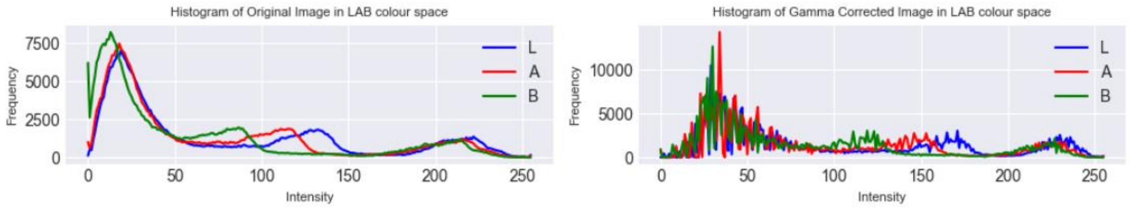
```
# plot transformed white matter image
image_graymatter = cv.LUT(img_orig, transform_graymatter)
```



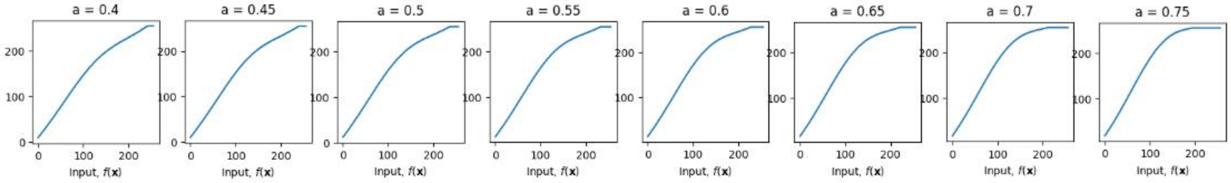
Q3. Selected gamma value = 0.65

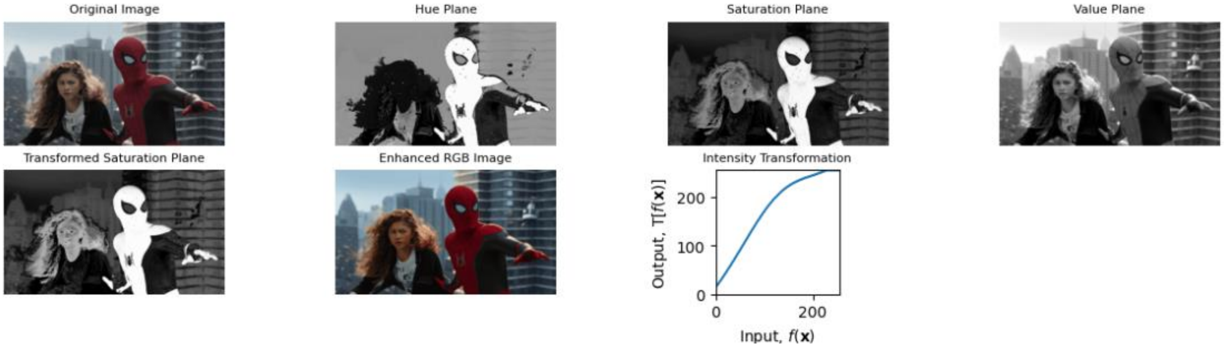

Gamma = 0.35
Gamma = 0.5
Gamma = 0.65
Gamma = 0.8
Gamma = 0.9500000000000001
Gamma = 1.1
Gamma = 1.25
Gamma = 1.4
Gamma = 1.5499999999999998
Gamma = 1.6999999999999997
Gamma = 1.8499999999999996
Gamma = 1.9999999999999996
Gamma = 2.1499999999999995
Gamma = 2.2999999999999994
Gamma = 2.4499999999999993
Gamma = 2.599999999999999
Gamma = 2.749999999999999
Gamma = 2.899999999999999



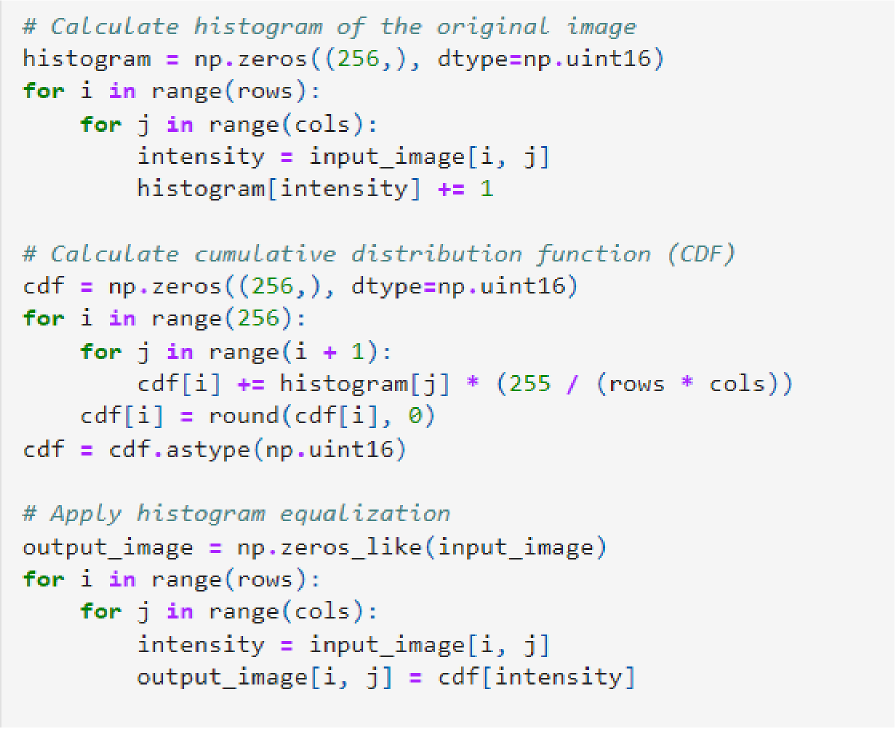


Q4. Selected ‘a’ value =0.6

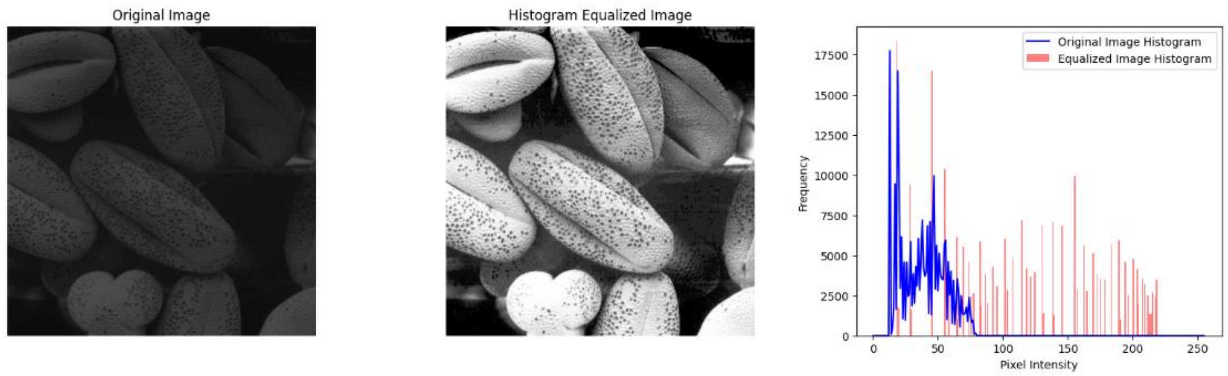




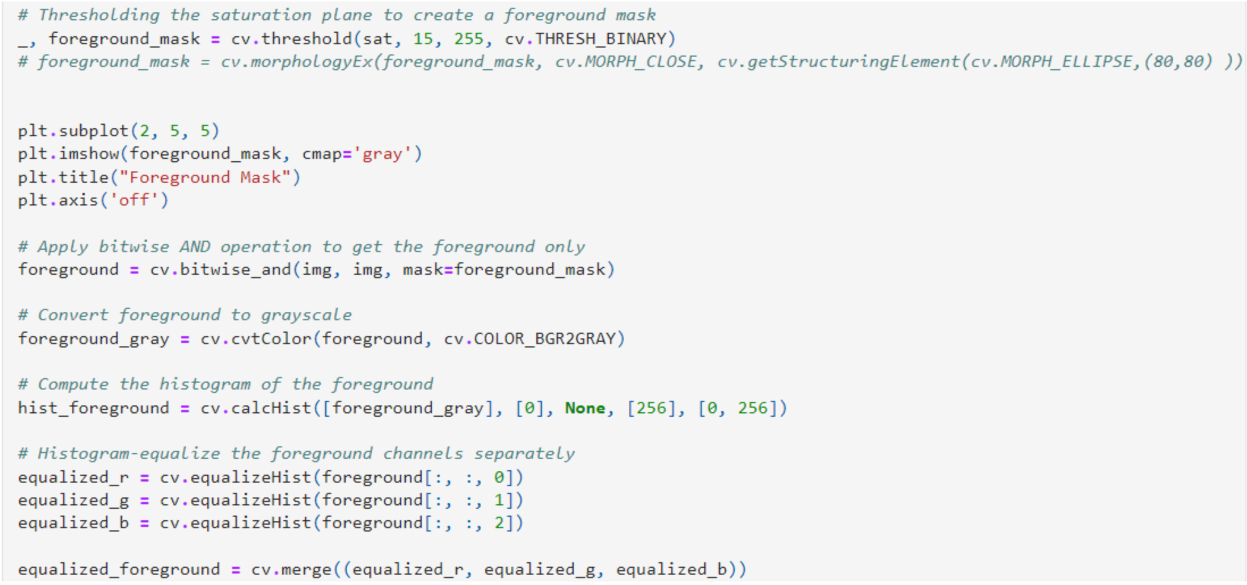
Q5.



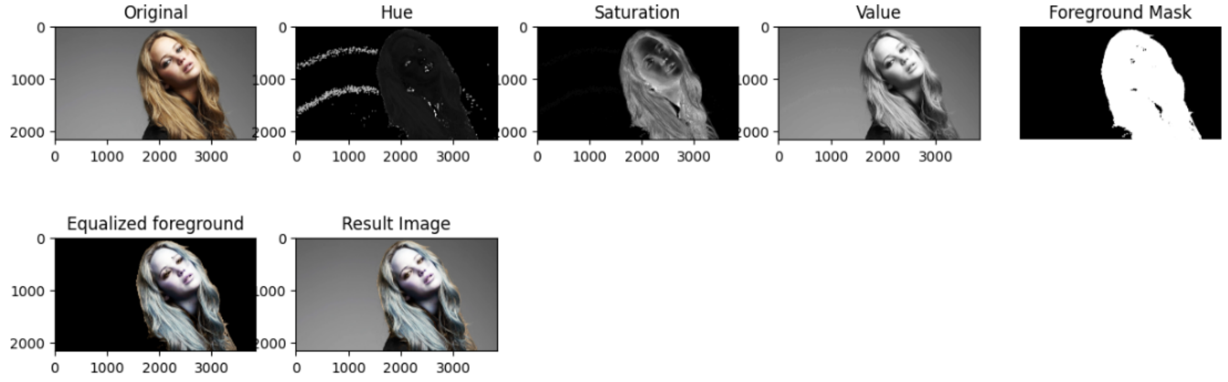
```
# Calculate histogram of the original image
histogram = np.zeros((256,), dtype=np.uint16)
for i in range(rows):
    for j in range(cols):
        intensity = input_image[i, j]
        histogram[intensity] += 1

# Calculate cumulative distribution function (CDF)
cdf = np.zeros((256,), dtype=np.uint16)
for i in range(256):
    for j in range(i + 1):
        cdf[i] += histogram[j] * (255 / (rows * cols))
    cdf[i] = round(cdf[i], 0)
cdf = cdf.astype(np.uint16)

# Apply histogram equalization
output_image = np.zeros_like(input_image)
for i in range(rows):
    for j in range(cols):
        intensity = input_image[i, j]
        output_image[i, j] = cdf[intensity]
```

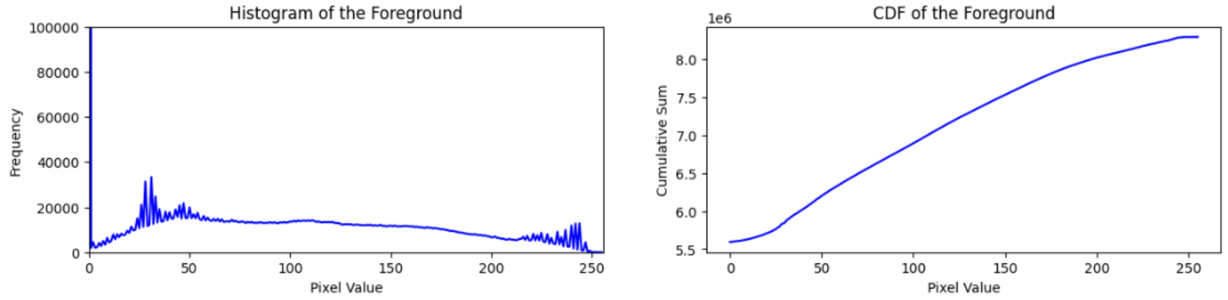


Q6.

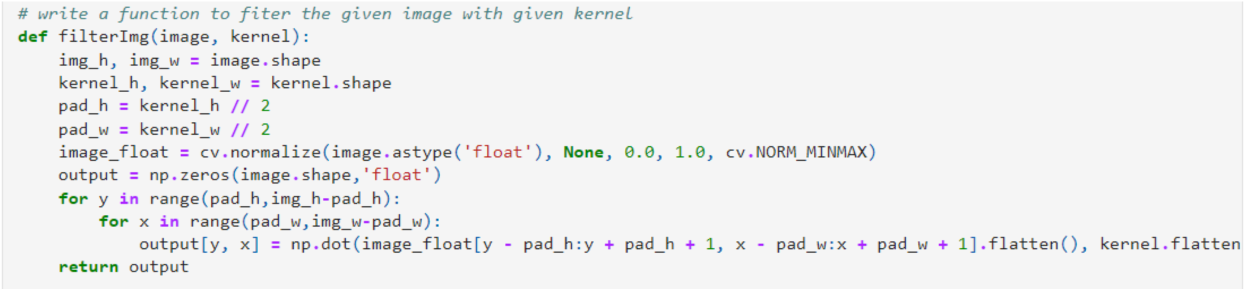


```
# Thresholding the saturation plane to create a foreground mask
_, foreground_mask = cv.threshold(sat, 15, 255, cv.THRESH_BINARY)
# foreground_mask = cv.morphologyEx(foreground_mask, cv.MORPH_CLOSE, cv.getStructuringElement(cv.MORPH_ELLIPSE,(80,80) ))


plt.subplot(2, 5, 5)
plt.imshow(foreground_mask, cmap='gray')
plt.title("Foreground Mask")
plt.axis('off')

# Apply bitwise AND operation to get the foreground only
foreground = cv.bitwise_and(img, img, mask=foreground_mask)

# Convert foreground to grayscale
foreground_gray = cv.cvtColor(foreground, cv.COLOR_BGR2GRAY)

# Compute the histogram of the foreground
hist_foreground = cv.calcHist([foreground_gray], [0], None, [256], [0, 256])

# Histogram-equalize the foreground channels separately
equalized_r = cv.equalizeHist(foreground[:, :, 0])
equalized_g = cv.equalizeHist(foreground[:, :, 1])
equalized_b = cv.equalizeHist(foreground[:, :, 2])

equalized_foreground = cv.merge((equalized_r, equalized_g, equalized_b))
```

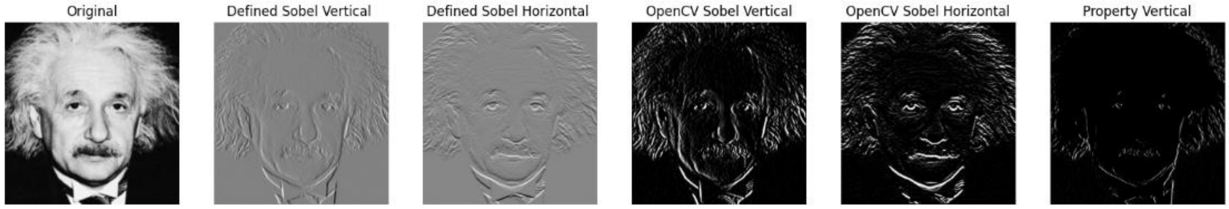




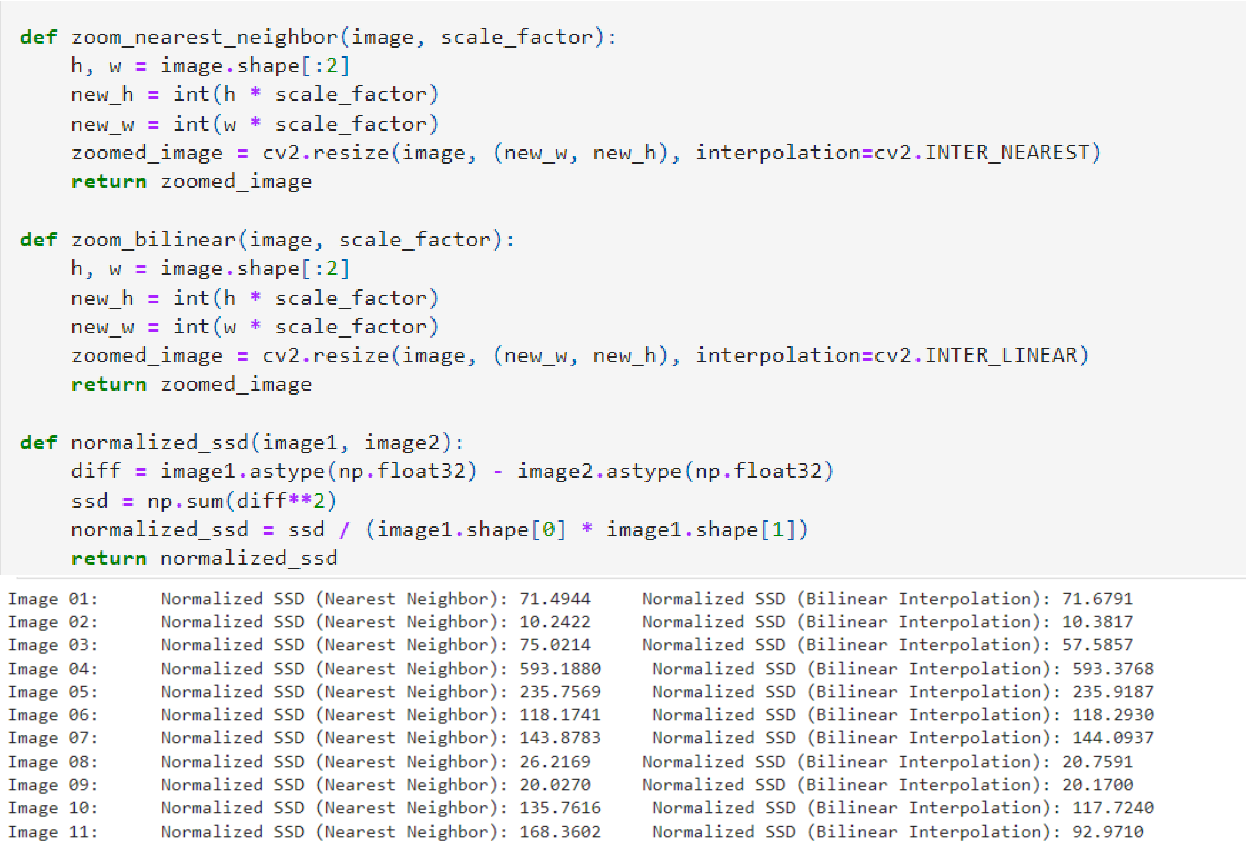
Q7.



```
# write a function to fiter the given image with given kernel
def filterImg(image, kernel):
    img_h, img_w = image.shape
    kernel_h, kernel_w = kernel.shape
    pad_h = kernel_h // 2
    pad_w = kernel_w // 2
    image_float = cv.normalize(image.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
    output = np.zeros(image.shape,'float')
    for y in range(pad_h,img_h-pad_h):
        for x in range(pad_w,img_w-pad_w):
            output[y, x] = np.dot(image_float[y - pad_h:y + pad_h + 1, x - pad_w:x + pad_w + 1].flatten(), kernel.flatten
    return output
```



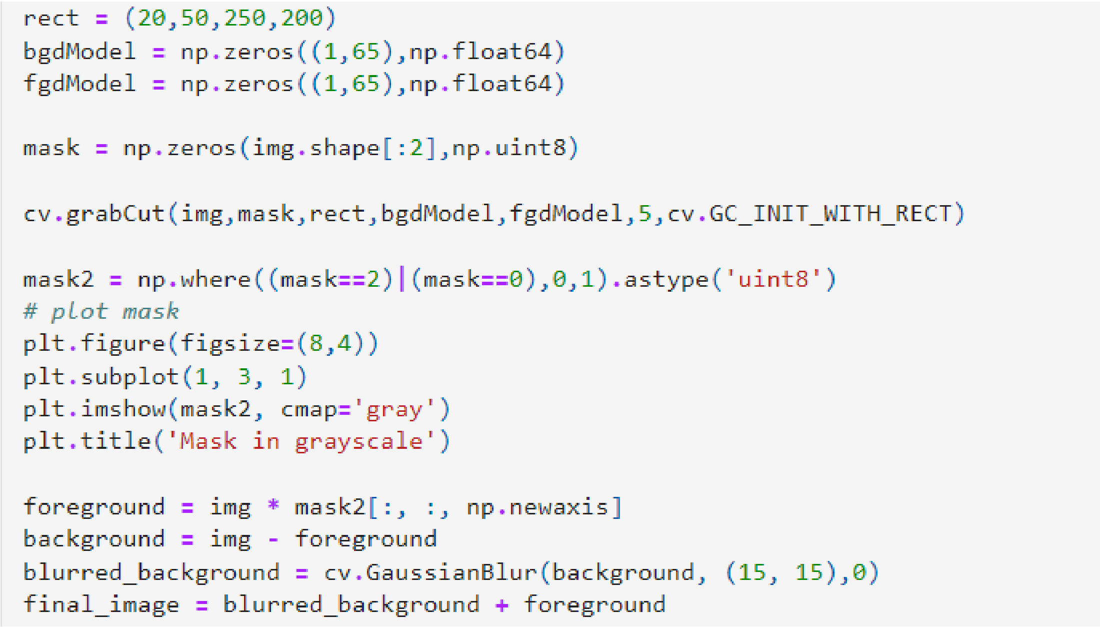
Q8.



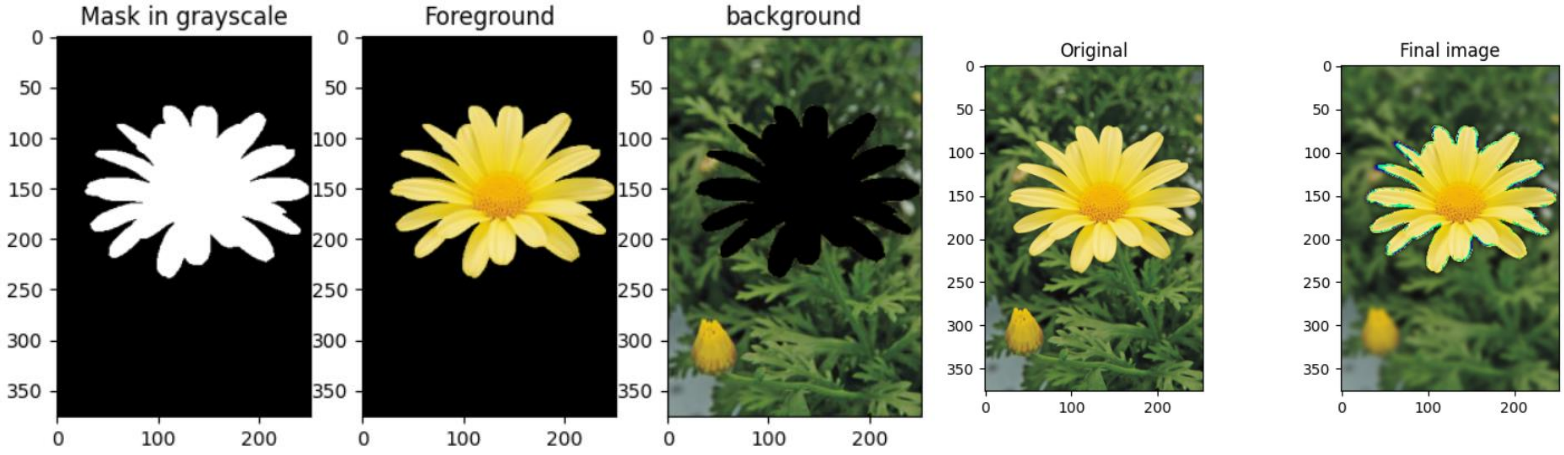
```
def zoom_nearest_neighbor(image, scale_factor):
    h, w = image.shape[:2]
    new_h = int(h * scale_factor)
    new_w = int(w * scale_factor)
    zoomed_image = cv2.resize(image, (new_w, new_h), interpolation=cv2.INTER_NEAREST)
    return zoomed_image

def zoom_bilinear(image, scale_factor):
    h, w = image.shape[:2]
    new_h = int(h * scale_factor)
    new_w = int(w * scale_factor)
    zoomed_image = cv2.resize(image, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
    return zoomed_image

def normalized_ssd(image1, image2):
    diff = image1.astype(np.float32) - image2.astype(np.float32)
    ssd = np.sum(diff**2)
    normalized_ssd = ssd / (image1.shape[0] * image1.shape[1])
    return normalized_ssd
```

```
Image 01:      Normalized SSD (Nearest Neighbor): 71.4944    Normalized SSD (Bilinear Interpolation): 71.6791
Image 02:      Normalized SSD (Nearest Neighbor): 10.2422    Normalized SSD (Bilinear Interpolation): 10.3817
Image 03:      Normalized SSD (Nearest Neighbor): 75.0214    Normalized SSD (Bilinear Interpolation): 57.5857
Image 04:      Normalized SSD (Nearest Neighbor): 593.1880    Normalized SSD (Bilinear Interpolation): 593.3768
Image 05:      Normalized SSD (Nearest Neighbor): 235.7569    Normalized SSD (Bilinear Interpolation): 235.9187
Image 06:      Normalized SSD (Nearest Neighbor): 118.1741    Normalized SSD (Bilinear Interpolation): 118.2930
Image 07:      Normalized SSD (Nearest Neighbor): 143.8783    Normalized SSD (Bilinear Interpolation): 144.0937
Image 08:      Normalized SSD (Nearest Neighbor): 26.2169    Normalized SSD (Bilinear Interpolation): 20.7591
Image 09:      Normalized SSD (Nearest Neighbor): 20.0270    Normalized SSD (Bilinear Interpolation): 20.1700
Image 10:      Normalized SSD (Nearest Neighbor): 135.7616    Normalized SSD (Bilinear Interpolation): 117.7240
Image 11:      Normalized SSD (Nearest Neighbor): 168.3602    Normalized SSD (Bilinear Interpolation): 92.9710
```

Q9.



```
rect = (20,50,250,200)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)

mask = np.zeros(img.shape[:2],np.uint8)

cv.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
# plot mask
plt.figure(figsize=(8,4))
plt.subplot(1, 3, 1)
plt.imshow(mask2, cmap='gray')
plt.title('Mask in grayscale')

foreground = img * mask2[:, :, np.newaxis]
background = img - foreground
blurred_background = cv.GaussianBlur(background, (15, 15),0)
final_image = blurred_background + foreground
```



Q9. (c)The dark (black) pixels are added to the blurred image when the foreground image and the blurred background image are layered on top of one another. Consequently, the flower's margin is a little bit black.