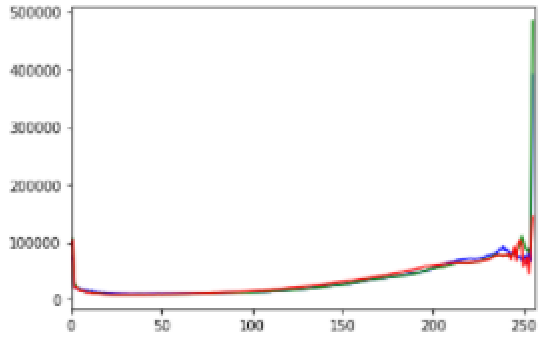
B.M.D.S.Karunarathna

180308C

<u>Assignment 01</u>

01) (a) Histogram Computation

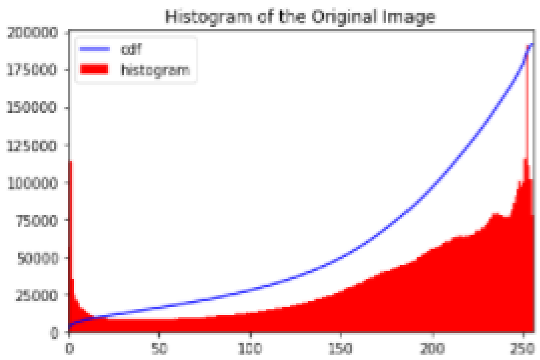
```python
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
img=cv.imread('../a01images/im04.png',cv.IMREAD_COLOR)
color=('b','g','r')
for i,c in enumerate(color):
    hist=cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(hist,color=c)
```

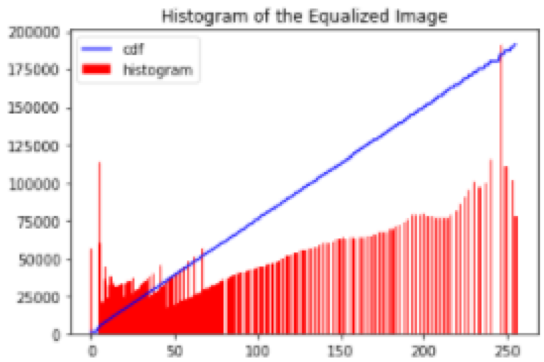


(b) Histogram Equalization

```python
hist=cv.calcHist([img],[0],None,[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')
```

```python
equ=cv.equalizeHist(img)
hist=cv.calcHist([equ],[0],None,[256],[0,256])
cdf=hist.cumsum()
cdf_normalized=cdf*hist.max()/cdf.max()
plt.plot(cdf_normalized,color='b')
plt.hist(equ.flatten(),256,[0,256],color='r')
```

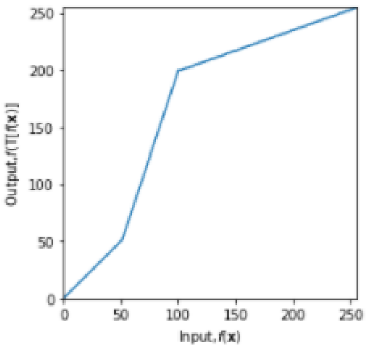




Before equalizing, histogram is confined to some specific range only. After equalizing, it stretches the histogram to either ends and the cdf becomes linear.

(c) Intensity transformations

```
c=np.array([(50,50),(100,200)])
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],255,255-c[1,0]).astype('uint8')
transform=np.concatenate((t1,t2),axis=0).astype('uint8')
transform=np.concatenate((transform,t3),axis=0).astype('uint8')
```




```
img=cv.imread('../a01images/im04.png',cv.IMREAD_COLOR)
image_transformed=cv.LUT(img,transform)
```

(d) Gamma Correction

```
table=np.array([(i/255.0)**(gamma)*255.0 for i in np.arange(0,256)]).astype('uint8')
Image_gamma1=cv.LUT(Image,table)
```

- When 0<gamma<1, dark regions become bright and bright regions become dark.
- When gamma>1, dark regions become more dark and bright regions become more bright.

(e) Gaussian Smoothing

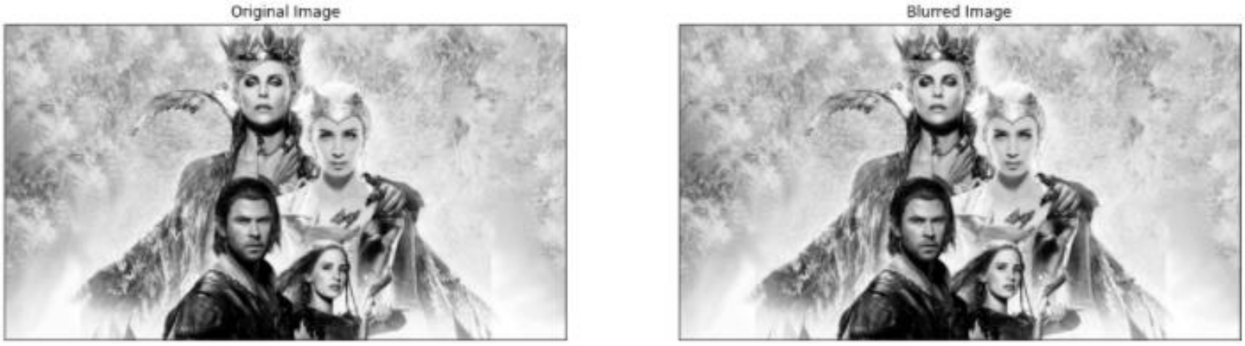
Kernel Size = 13

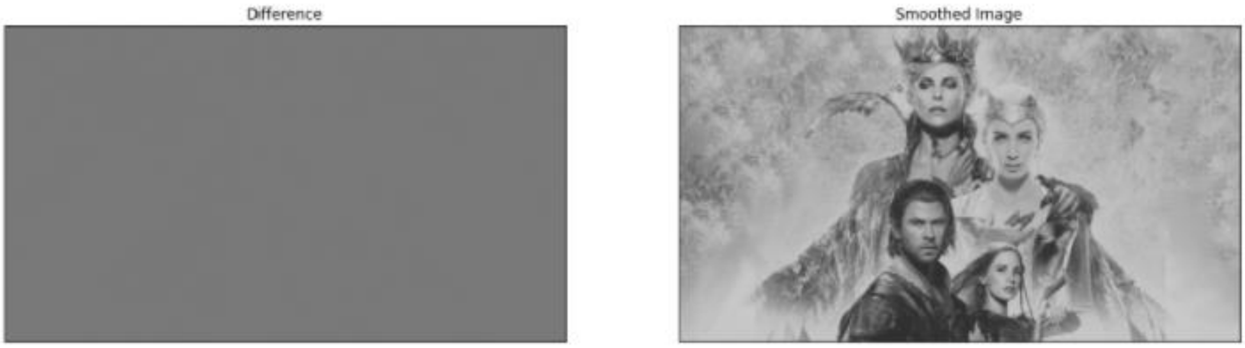
Sigma = 7

```
img=cv.imread('../a01images/im04.png',cv.IMREAD_COLOR)
img_blurred=cv.GaussianBlur(img,(5,5),4)
```

(f) Unsharp Masking

```
img=cv.imread('../a01images/im04.png',cv.IMREAD_GRAYSCALE)
sigma=2
kernel =cv.getGaussianKernel(5,sigma)
blurred_img=cv.sepFilter2D(img,-1,kernel,kernel,anchor=(-1,-1),delta=0,borderType=cv.BORDER_REPLICATE)
diff=img.astype('float32')-blurred_img.astype('float32')
sharpened=cv.addWeighted(img.astype('float32'),0.7,diff,0.3,0)
```

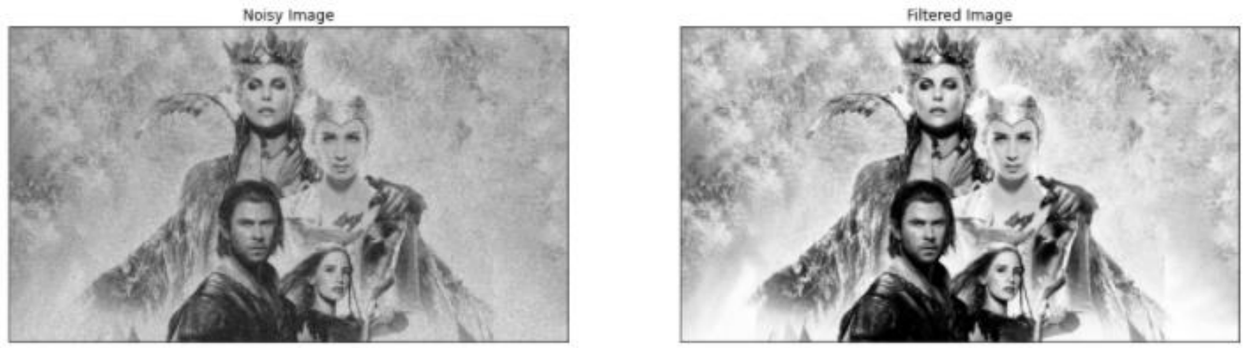




(g) Median Filtering

Kernel Size = 7

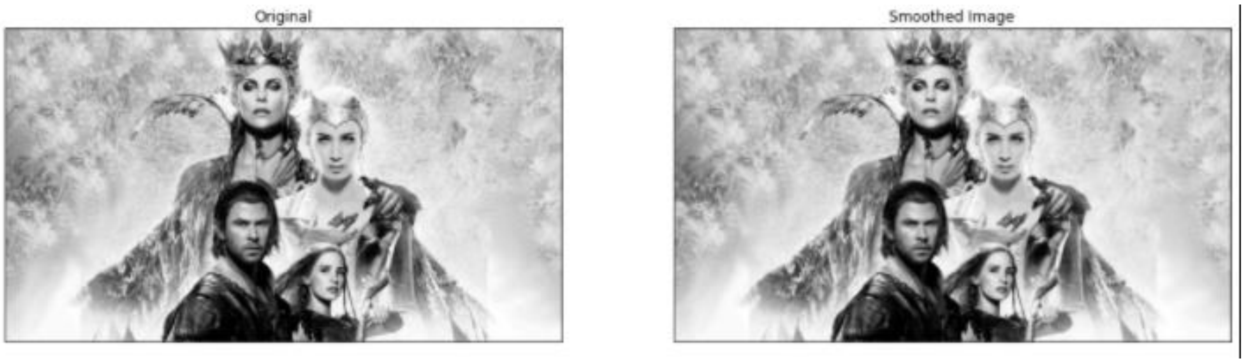
```
Image=cv.imread('../a01images/im04.png',cv.IMREAD_GRAYSCALE)
Image_with_noise=s_p(Image)
Image_Filtered=cv.medianBlur(Image_with_noise,7)
```



First image consists of Salt and pepper noise. After Median filtering noise is removed and the filtered image is obtained.

(h) Bilateral Filtering

```
Image=cv.imread('../a01images/im04.png',cv.IMREAD_GRAYSCALE)
Image_Blurred=cv.bilateralFilter(Image,15,75,75)
```



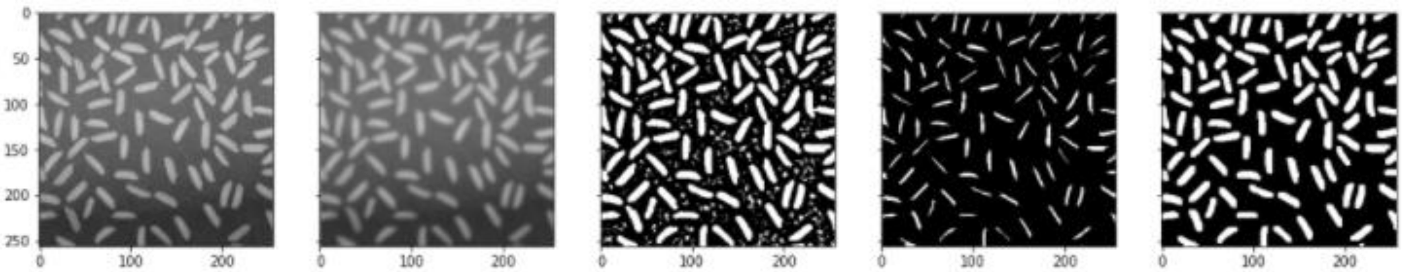
Bilateral Filtering is a technique to smooth images while preserving edges. Similarly to the Gaussian convolution, in bilateral filter each filter is replaced by an average of its neighbors.

The bilateral filter is controlled by two parameters: Range and Spatial parameter

- When range increases, the bilateral filter becomes closer to Gaussian blur because the range Gaussian is flatter i.e., almost a constant over the intensity interval covered by the image.
- Increasing the spatial parameter smooth larger features. An important characteristic of bilateral filtering is that the weights are multiplied, which implies that as soon as one of the weight is close to 0, no smoothing occurs.

(2) No. of rice grains in the rice image

```
Img_rice=cv.imread('../a01images/rice.png',cv.IMREAD_GRAYSCALE)
kernel=np.ones((5,5),np.uint8)
Img_GaussianBlurred=cv.GaussianBlur(Img_rice,(7,7),2)
Img_Threshold=cv.adaptiveThreshold(Img_GaussianBlurred,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINARY,9,0)
Img_Eroded=cv.erode(Img_Threshold,kernel,iterations=1)
Img_Dilate=cv.dilate(Img_Eroded,kernel,iterations=1)
connectedOnes,labels=cv.connectedComponents(Img_Dilate)
ricegrains=connectedOnes-1
```

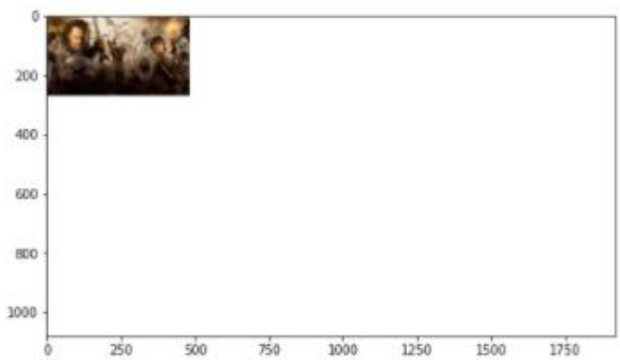


No. of rice grains = 100

(3) Zooming Images

(a) Nearest – neighbor

```
im_original=cv.imread('../a01images/im01.png',cv.IMREAD_COLOR)
im=cv.imread('../a01images/im01small.png',cv.IMREAD_COLOR)
scale=4
rows=int(scale*im.shape[0])
cols=int(scale*im.shape[1])
channels=im.shape[2]
zoomed=np.zeros((rows,cols,channels),dtype=im.dtype)
for i in range(0,rows):
    for j in range(0,cols):
        for k in range(0,3):
            x=round(i/scale)
            y=round(j/scale)
            if x==im.shape[0]:x=x-1
            if y==im.shape[1]:y=y-1
            zoomed[i,j,k]=im[x,y,k]
```



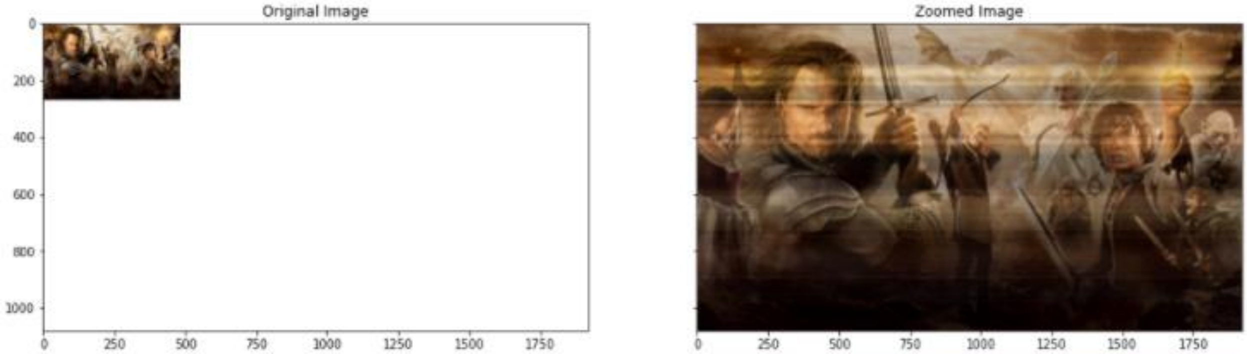
- SSD for Nearest Neighbor = 785292121

```
SSD=0
for i in range(0,im1.shape[0]):
    for j in range(0,im1.shape[1]):
        for k in range(0,3):
            dif=(zoomed[i,j,k]-im_original[i,j,k])**2
            SSD=SSD+dif
print(SSD)
```

```
785292121
```

(b) Bilinear Interpolation

```
im1=cv.imread('../a01images/im01.png',cv.IMREAD_COLOR)
im=cv.imread('../a01images/im01small.png',cv.IMREAD_COLOR)
scale=4
rows=int(scale*im.shape[0])
cols=int(scale*im.shape[1])
channels=im.shape[2]
zoomed=np.zeros((rows,cols,channels),dtype=im.dtype)
for i in range(0,rows):
    for j in range(0,cols):
        for k in range(0,3):
            x=i/scale
            y=j/scale
            x1=int(x)
            y1=int(y)
            p=x-x1
            q=y-y1
            x2=x1+1
            y2=x2+1
            if x2==im.shape[0]:x2=x1
            if y2==im.shape[1]:y2=y1
            value=(1-q)*((1-p)*im[x1,y1][k]+p*im[x2,y1][k])+q*((1-p)*im[x1,y2][k]+p*im[x2,y2][k])
            zoomed[i,j,k]=value
```



- SSD for bilinear Interpolation = 1840641787

```
SSD=0
for i in range(0,im1.shape[0]):
    for j in range(0,im1.shape[1]):
        for k in range(0,3):
            dif=(im1[i,j,k]-zoomed[i,j,k])**2
            SSD=SSD+dif
print(SSD)
1840641787
```