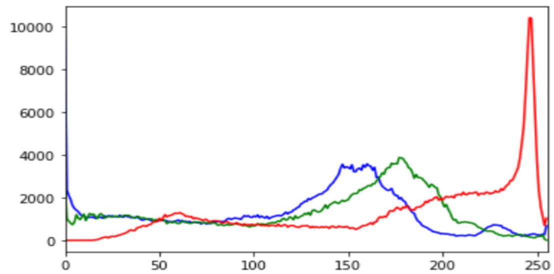
Assignment Report

General Code

```
In [2]:  import cv2 as cv
         import numpy as np
         import matplotlib.pyplot as plt
```

Q1. A)

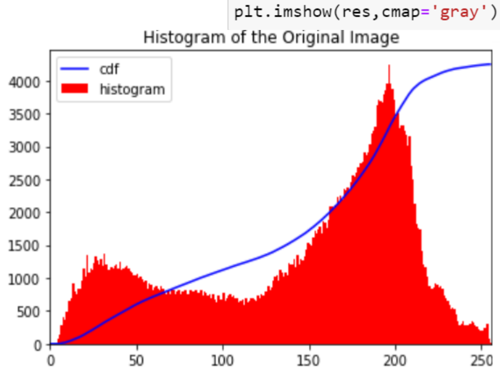
```
In [12]:  #computing the histogram of the given image
          img = cv.imread('Katrina_Kaif.jpg',cv.IMREAD_COLOR )
          color=('b','g','r')
          for i, c in enumerate(color):
              hist = cv.calcHist([img], [i], None, [256], [0,256])
              plt.plot(hist,color=c )
          plt.xlim([0,256])
          plt.show()
```

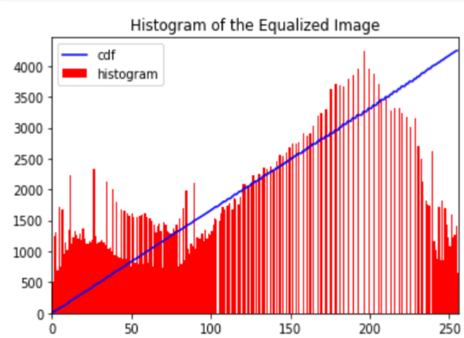


The graph shows the intensity of each pixels.

B)

```
In [4]:  img = cv.imread('Katrina_Kaif.jpg', cv.IMREAD_GRAYSCALE )
         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 the Original Image')
         plt.show()
         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() , 256 , [0,256] , color='r')
         plt.xlim ([0,256] )
         plt.legend(('cdf','histogram') , loc='upper left' )
         plt.title('Histogram of the Equalized Image' )
         plt.show()
         res=np.hstack((img , equ))
         plt.axis ('off' )
         plt.imshow(res,cmap='gray')
```

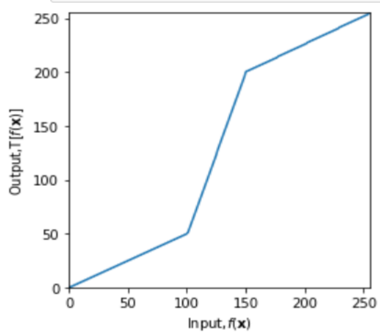




We can observe that the intensity has been distributed all over the image.

C)

```
c = np.array([(100,50),(150,200)])
t1=np.linspace(0,c[0,1],c[0,0]+1-0).astype('uint8')
print(len(t1))
t2 = np.linspace(c[0,1]+1,c[1,1],c[1,0]-c[0,0]).astype('uint8')
print(len(t2))
t3=np.linspace(c[1,1]+1,255,255-c[1,0]).astype('uint8')
print(len(t3))
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3), axis=0).astype('uint8')
print(len(transform))
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.show()
img_orig =cv.imread('Katrina_Kaif.jpg' , cv.IMREAD_GRAYSCALE )
cv.namedWindow ( " Image " , cv.WINDOW_AUTOSIZE)
cv.imshow ( " Image " , img_orig)
cv.waitKey(0)
image_transformed = cv.LUT(img_orig , transform )
cv.imshow ( "Image" , image_transformed )
cv.waitKey(0)
cv.destroyAllWindows()
```



We see that the intensity of the image has changed because of the transformation we have made. (small slope in lower and higher pixels and larger slope in mid-range pixels)

D)

In [7]:

```
gamma=4
f=cv.imread( 'Katrina_Kaif.jpg' , cv.IMREAD_GRAYSCALE)/255.

cv.namedWindow('Image',cv.WINDOW_AUTOSIZE)
cv.imshow('Image',f)
cv.waitKey(0)
g=f**gamma
cv.imshow('Image',g)
cv.waitKey(0)
cv.destroyAllWindows()
```

With gamma, intensity of the pixels change and we can see that through the image as the colour changes with gamma. Gamma is 4 here.

E)

```
In [19]: img = cv.imread('Katrina_Kaif.jpg')
         sigma = 2     #sigma is 2
         k_width = 2*(3*sigma)+1 # size is 13
         G_img = cv.GaussianBlur(img,(k_width,k_width),sigma)
         fig,(ax1,ax2) = plt.subplots(1,2 , figsize=(6,6))
         ax1.imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
         ax1.axis("off")
         ax2.imshow(cv.cvtColor(G_img,  cv . COLOR_BGR2RGB))
         ax2.axis("off")

Out[19]: (-0.5, 481.5, 679.5, -0.5)
```

Image is filtered by calculating the spatial distances and adding it as kernel weights. Sigma is 2.

F)

```
In [22]: img = cv.imread('Katrina_Kaif.jpg')
         sigma =3
         k_width = 2*(3*sigma)+1
         blured_img = cv.GaussianBlur(img,(k_width,k_width),0)
         unsharp_masked_img= cv.addWeighted(img,2.5,blured_img,-1.5,0)
         fig,(ax1,ax2) = plt.subplots(1,2,  figsize=(6,6))
         ax1.imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
         ax1.axis("off")
         ax2.imshow(cv.cvtColor(unsharp_masked_img,  cv . COLOR_BGR2RGB))
         ax2.axis("off")

Out[22]: (-0.5, 481.5, 679.5, -0.5)
```

The image is filtered here by adding the scaled high passed image to the original image.

G)

this

```
In [24]: img = cv.imread('Katrina_Kaif.jpg')
         kernal_size=3
         median_blurred_img = cv.medianBlur(img,kernal_size)
         fig,(ax1,ax2) = plt.subplots(1,2,  figsize=(6,6))
         ax1.imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
         ax1.axis("off")
         ax2.imshow(cv.cvtColor(median_blurred_img,  cv . COLOR_BGR2RGB))
         ax2.axis("off")

Out[24]: (-0.5, 481.5, 679.5, -0.5)
```

The noise in the original image is removed through non-linear digital filter.

H)

In [26]:
```python
img = cv.imread('Katrina_Kaif.jpg')
biletral_img = cv.bilateralFilter(img,14,75,75)
fig,(ax1,ax2) = plt.subplots(1,2,  figsize=(7,7))
ax1.imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
ax1.axis("off")
ax2.imshow(cv.cvtColor(biletral_img,  cv . COLOR_BGR2RGB))
ax2.axis("off")
```

Out[26]: (-0.5, 481.5, 679.5, -0.5)

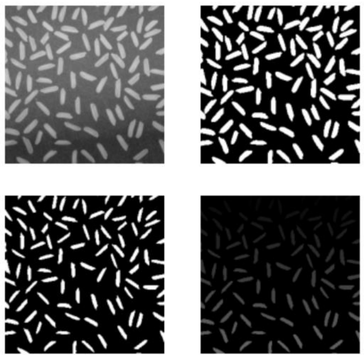
Comparing with the Gaussian filter, Gaussian uses spatial distance only to calculate the weights of the kernel slots. But bilateral filter in addition to spatial distance it considers the photometric distance / radiometric distance (range differences of color intensities and depth distance) too when allocating weights for the kernel. Photometric distance means the distance between pixels in color space. In boundaries the pixels could be closer in spatial distance but very far in color space. So in this filter by considering both type of distances **sharp edges** are preserved.

Q2.

In [28]:
```python
img = cv.imread('../../assignments/a01images/rice.png', cv.IMREAD_GRAYSCALE)
# using adaptive thresholding
adapthresh_img = cv.adaptiveThreshold (img, 255.0,cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 51, -20.0)
# removeing the white dots using erode interatively
kernel = np.ones((3,3),np.uint8)
erosion_img = cv.erode(adapthresh_img, kernel)
# counting the pixel with value =255 and then filling the connected pixels to stop recounting
processing_img = erosion_img.copy()
rice_count = 0
rows, cols = processing_img.shape
for j in range(rows):
    for i in range(cols):
        pixel = processing_img[j, i]
        if 255 == pixel:
            rice_count += 1
            cv.floodFill(processing_img, None, (i, j), rice_count)
print("Number of rice grains", rice_count)
fig,ax = plt.subplots(2,2,  figsize=(6,6))
ax[0,0].imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
ax[0,0].axis("off")
ax[0,1].imshow(cv.cvtColor(adapthresh_img,  cv . COLOR_BGR2RGB))
ax[0,1].axis("off")
ax[1,0].imshow(cv.cvtColor(erosion_img,  cv . COLOR_BGR2RGB))
ax[1,0].axis("off")
ax[1,1].imshow(cv.cvtColor(processing_img,  cv . COLOR_BGR2RGB))
ax[1,1].axis("off")
```

```
Number of rice grains 102
```

```
Out[28]: (-0.5, 255.5, 255.5, -0.5)
```



For different kernel sizes we get different number of grains. For example,

- Size -1*1 => 100 grains
- Size -3*3 => 102 grains
- Size -5*5 => 114 grains
- Size -7*7 => 195 grains

Yes it makes sense, with the kernel size when we consider the neighbor pixels, the number of white spots counted changes. So the number of grains computed changes too.

Q3.

```
In [31]:
def zoom_nearest_neighboour(img , z_factor):
    h_o = img.shape[0]
    w_o = img.shape[1]
    h_new = h_o*z_factor
    w_new = w_o*z_factor
    new_img = np.zeros(shape=(h_new,w_new,3), dtype='uint8')
    for i in range(h_new):
        for j in range (w_new):
            x=int(i/z_factor)
            y=int(j/z_factor)
            new_img[i,j]=img[x,y]
    return new_img
```

```
In [34]:
def bilinear_zoom(img,Z_factor):
    R_in,C_in,channels =img.shape
    R_out=R_in*Z_factor
    C_out=C_in*Z_factor
    out_img=np.zeros((R_out,C_out,channels),np.uint8)
    sh=R_out/R_in
    sw=C_out/C_in
    for i in range(R_out):
        for j in range(C_out):
            x = i/sh
            y = j/sw
            p=(i+0.0)/sh-x
            q=(j+0.0)/sw-y
            x=int(x)-1
            y=int(y)-1
            out_img[i, j] =(img[x,y]*(1-p)*(1-q)+img[x,y+1]*q*(1-p)+img[x+1,y]*(1-q)*p+img[x+1,y+1]*p*q)
    return out_img
```

From the calculations mentioned in the above codes, we get to see that both the zooming methods use different steps for zooming an image. As given in the question when we zoom an image by 4 using both the zooming methods, we can observe that the value we get by calculating the sum of squared differences between the original image and the zoomed image, differs for both zooming methods. This proves that the zoomed images through different zooming methods are different.

a) In [47]:

```
img = cv.imread('../../assignments/a01images/im01small.png')
zoomed_img = zoom_nearest_neighbour(img, 4)
#zoomed_img=bilinear_zoom(img,4)
fig,(ax1,ax2) = plt.subplots(1,2 , figsize=(10,10))
ax1.imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
ax1.set_title("original image small")
ax2.imshow(cv.cvtColor(zoomed_img,  cv . COLOR_BGR2RGB))
ax2.set_title("zoomed image")

#testing
def SSD(img1,img2):
    h=img1.shape[0]
    w=img1.shape[1]
    SSD=0
    for i in range(h):
        for j in range(w):
            SSD +=(img1[i][j]-img2[i][j])**2
    return SSD
img_given = cv.imread('../../assignments/a01images/im01.png')
print("SSD of two images = ", SSD(img_given,zoomed_img))
```

```
SSD of two images =  [127  92   9]
```

b) In [48]:

```
img = cv.imread('../../assignments/a01images/im01small.png')
#zoomed_img = zoom_nearest_neighbour(img, 4)
zoomed_img=bilinear_zoom(img,4)
fig,(ax1,ax2) = plt.subplots(1,2 , figsize=(10,10))
ax1.imshow(cv.cvtColor(img,  cv . COLOR_BGR2RGB))
ax1.set_title("original image small")
ax2.imshow(cv.cvtColor(zoomed_img,  cv . COLOR_BGR2RGB))
ax2.set_title("zoomed image")

#testing
def SSD(img1,img2):
    h=img1.shape[0]
    w=img1.shape[1]
    SSD=0
    for i in range(h):
        for j in range(w):
            SSD +=(img1[i][j]-img2[i][j])**2
    return SSD
img_given = cv.imread('../../assignments/a01images/im01.png')
print("SSD of two images = ", SSD(img_given,zoomed_img))
```

```
SSD of two images =  [ 41 108  79]
```