Muhammad Aasharib Nawshad

MSCS 2k20

Registration # 326842

CV CS867 Assign 1

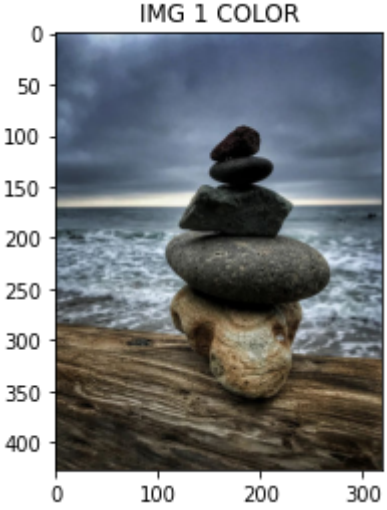
In [ ]:
```
# import required libraries
#import os
#import cv2
import matplotlib.pyplot as plt
%matplotlib inline
#import numpy as np
from skimage.util import random_noise
from matplotlib import cm
from helperUtil import *
```

TASK 1

Load the set of images and display them as Grayscale and rgb images. You are required to show these images ”inline” rather than creating a new window for every other image.

In [2]:
```
def displayImage(image, title, mode):
    if (mode == 0):
        grayImg = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        plt.imshow(grayImg, cmap="gray")
    if (mode == 1):
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.show()

imgFiles = readImagesFromFolder('images/')
index = 0
for imgFile in imgFiles:
    index += 1
    displayImage(imgFile, 'IMG ' + str(index) + ' GREY', 0)
    displayImage(imgFile, 'IMG ' + str(index) + ' COLOR', 1)
```

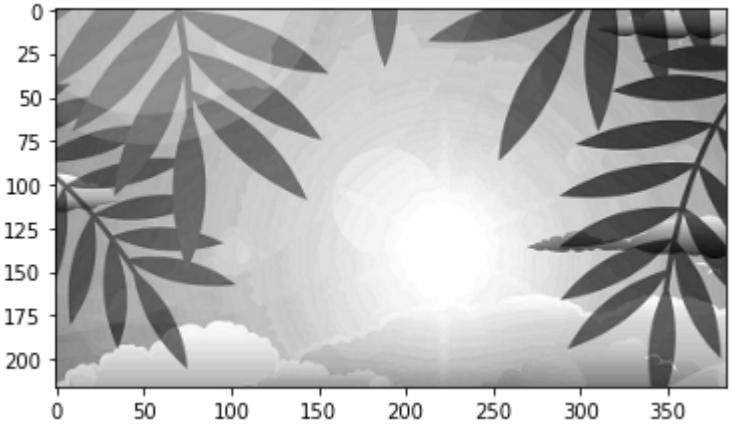
IMG 1 COLOR



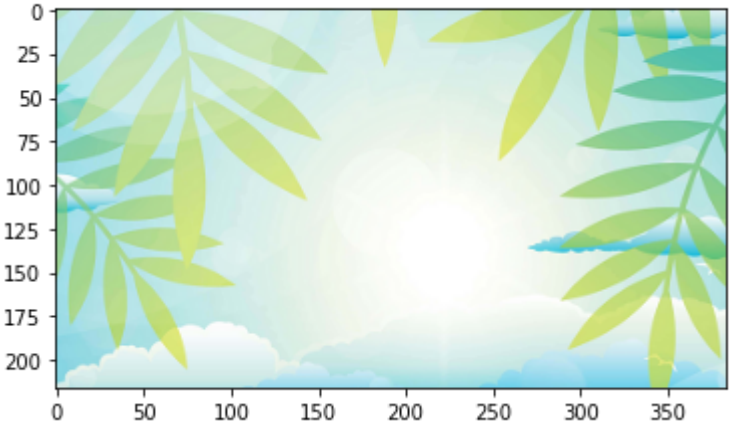
IMG 2 GREY

IMG 2 COLOR

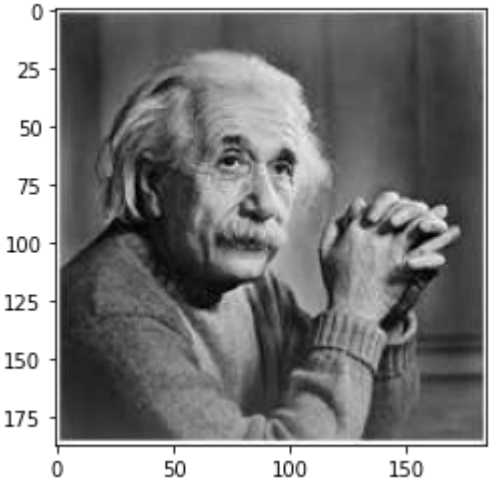
IMG 3 GREY



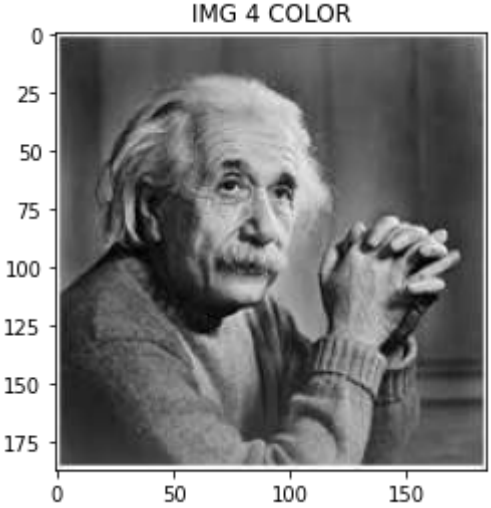
IMG 3 COLOR



IMG 4 GREY



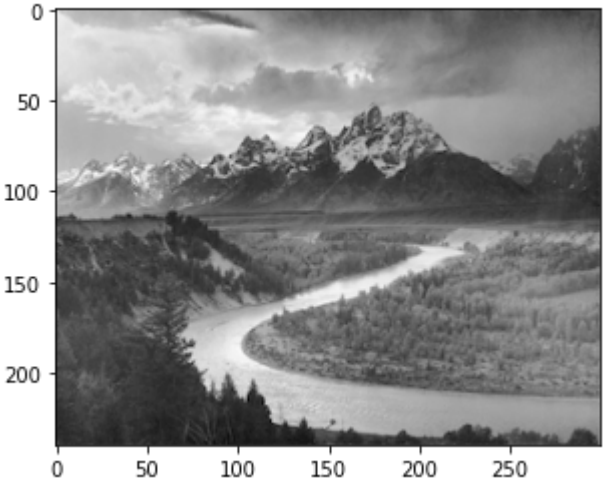
IMG 4 COLOR



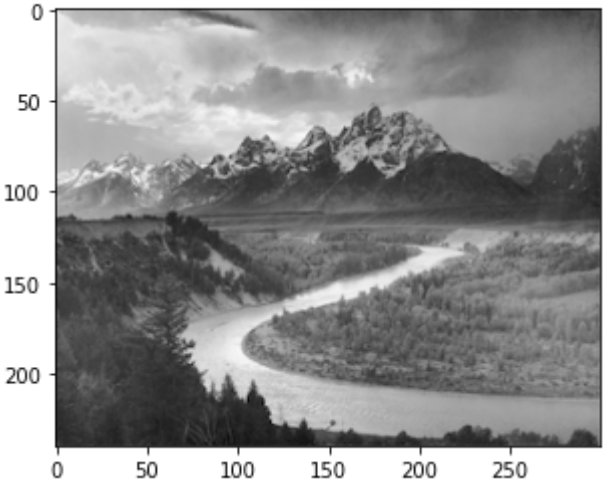
IMG 5 GREY

IMG 5 COLOR

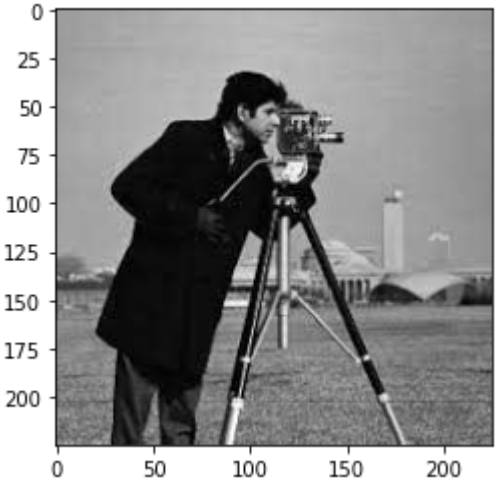
IMG 6 GREY



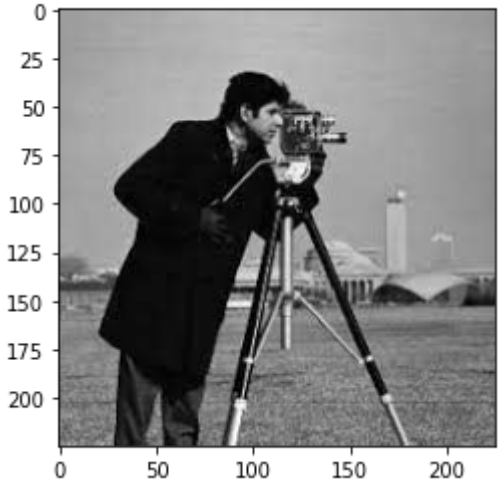
IMG 6 COLOR



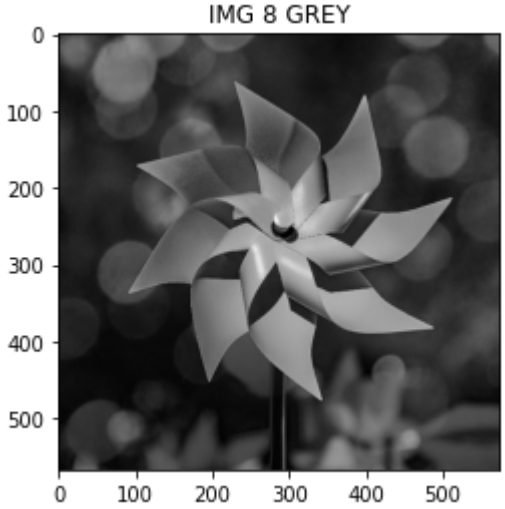
IMG 7 GREY



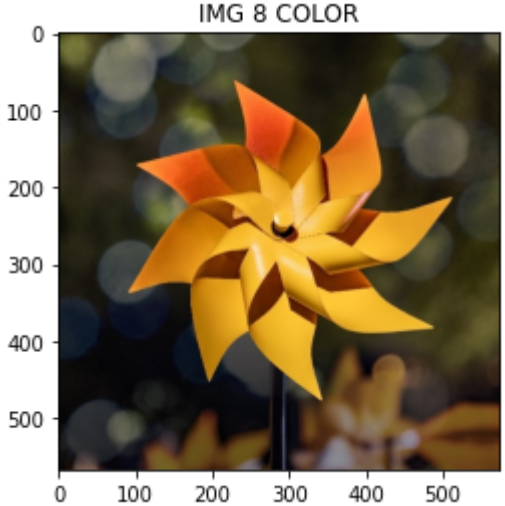
IMG 7 COLOR



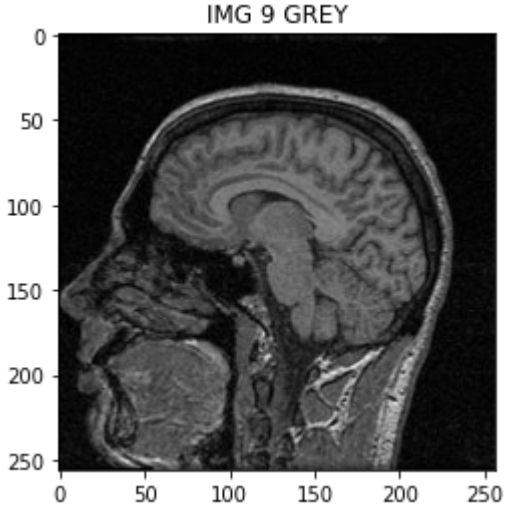
IMG 8 GREY

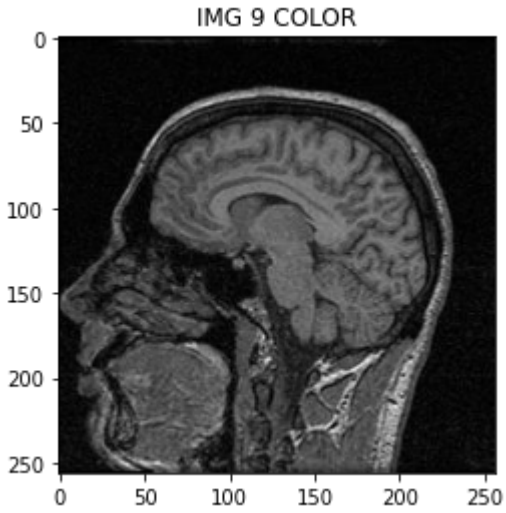


IMG 8 COLOR



IMG 9 GREY

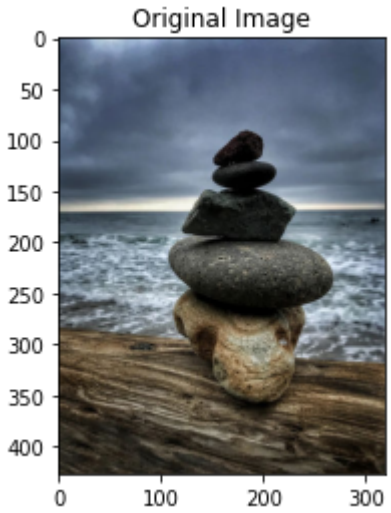


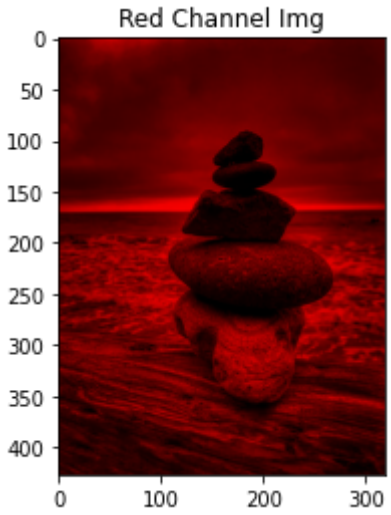


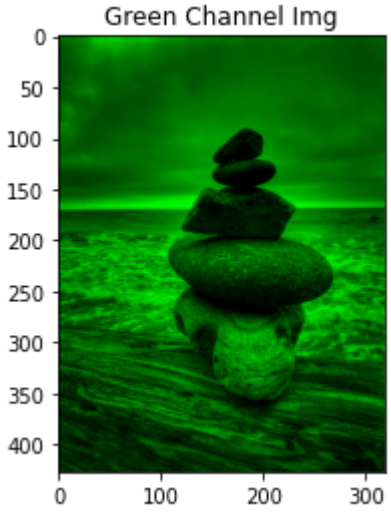
TASK 2

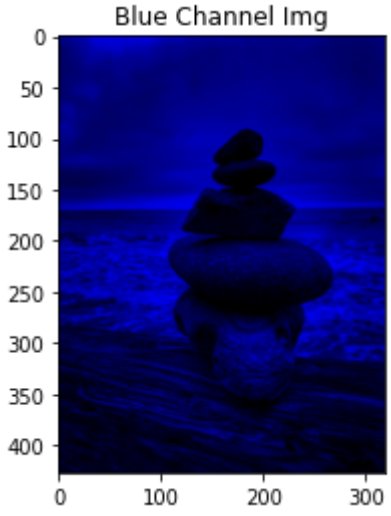
Implement the function rgbExclusion() in the helper script, in which the input image is decomposed into the three channels: R, G and B and return the image excluding the specified channel. Display the results in notebook.

In [3]:
```
from helperUtil import regExclusion
rchannel = regExclusion(imgFiles[0],'R')
gchannel = regExclusion(imgFiles[0],'G')
bchannel = regExclusion(imgFiles[0],'B')
displayImage(imgFiles[0],'Original Image',1)
displayImage(rchannel,'Red Channel Img',1)
displayImage(gchannel,'Green Channel Img',1)
displayImage(bchannel,'Blue Channel Img',1)

rchannel = regExclusion(imgFiles[4],'R')
gchannel = regExclusion(imgFiles[4],'G')
bchannel = regExclusion(imgFiles[4],'B')
displayImage(imgFiles[4],'Original Image',1)
displayImage(rchannel,'Red Channel Img',1)
displayImage(gchannel,'Green Channel Img',1)
displayImage(bchannel,'Blue Channel Img',1)
```

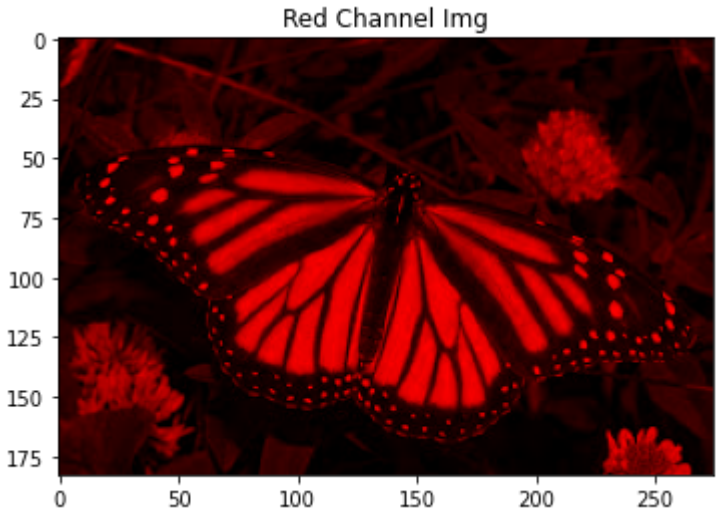


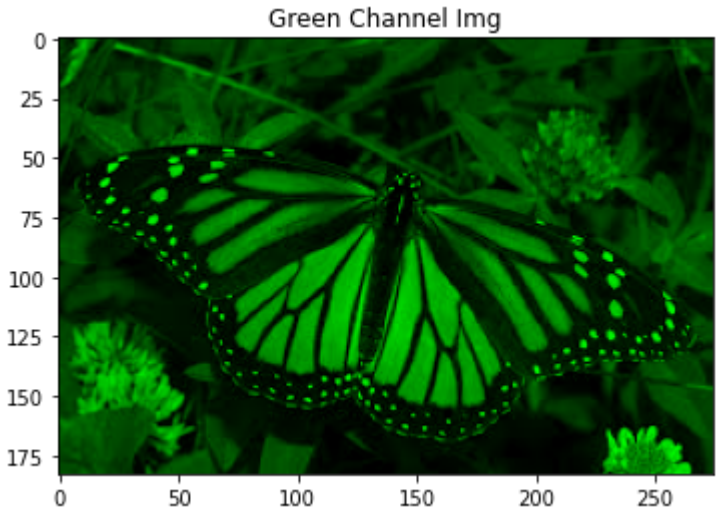

Red Channel Img
0
50
100
150
200
250
300
350
400
0
100
200
300

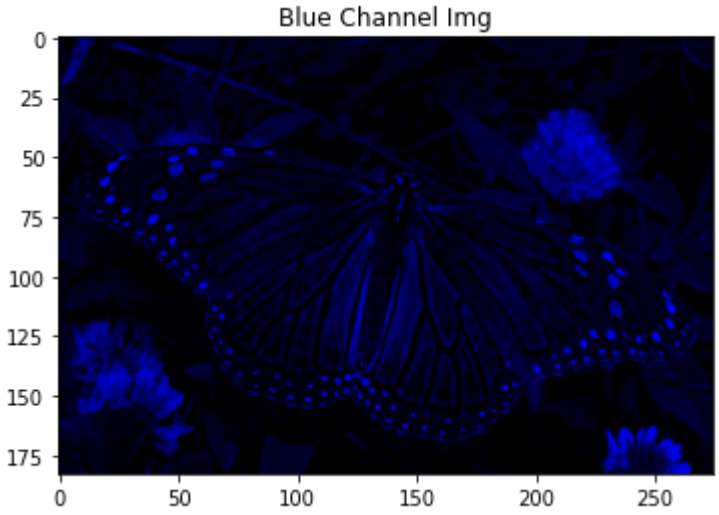

Green Channel Img
0
50
100
150
200
250
300
350
400
0
100
200
300


Blue Channel Img
0
50
100
150
200
250
300
350
400
0
100
200
300


Original Image
0
25
50
75
100
125
150
175
0
50
100
150
200
250







TASK3

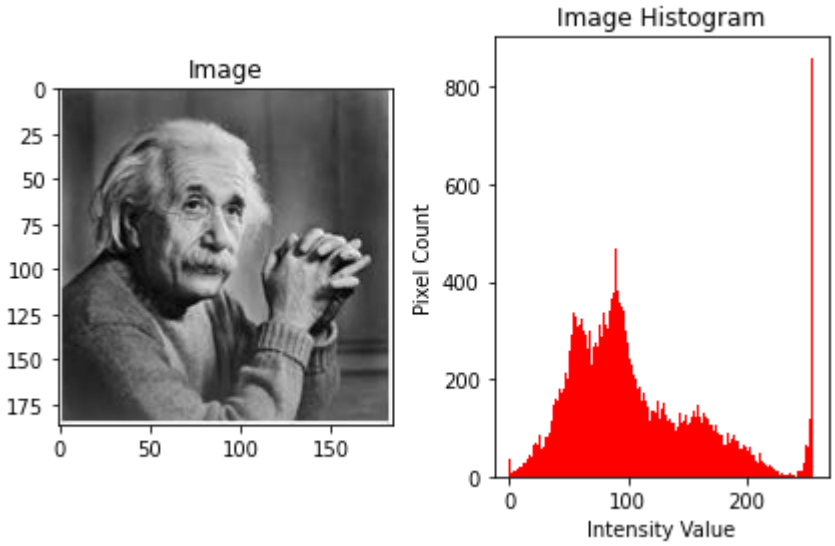
Take at-least 3 images from given set and plot histograms before and after applying histogram equalization. Show these image inline format i.e. grayscale image –> display histogram –> apply histogram equalization–> display the equalized image and its histogram.

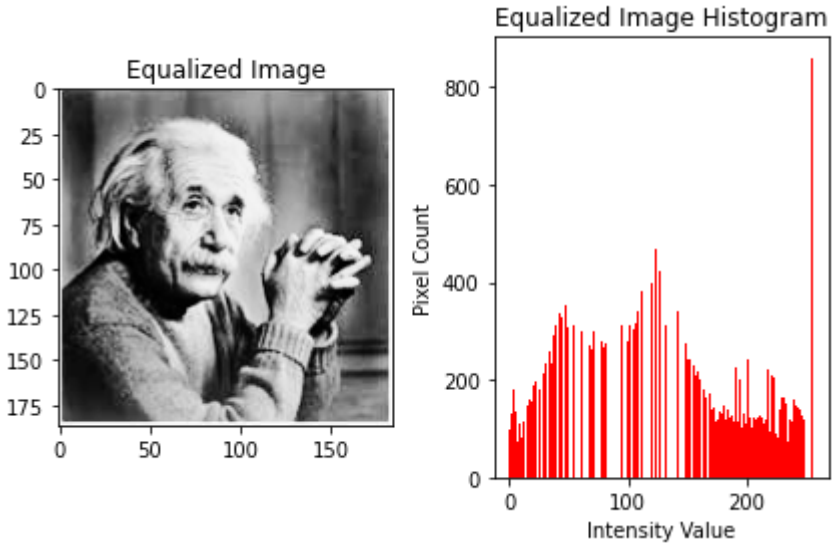
In [4]:

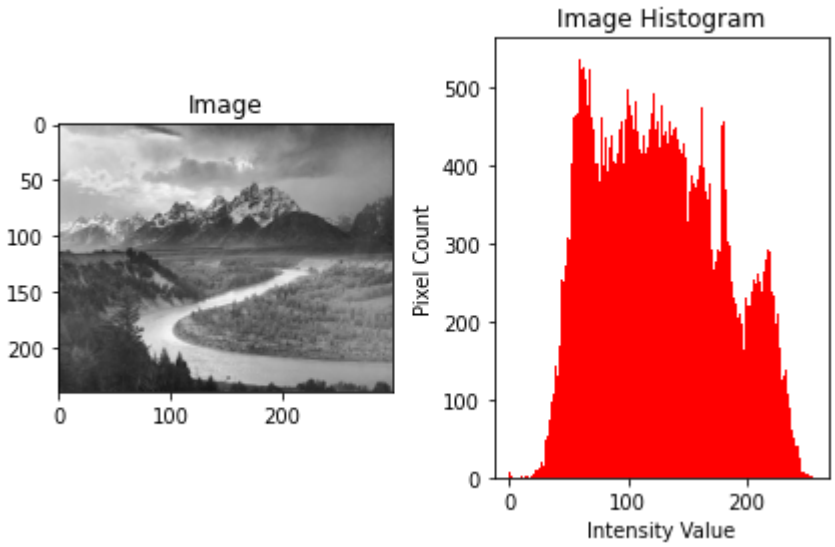
```
import numpy as np
grayImgFiles = readGrayImagesFromFolder('images/')
img1 = grayImgFiles[3]
img2 = grayImgFiles[5]
img3 = grayImgFiles[6]

##first image
fig, ax = plt.subplots(1,2)
```

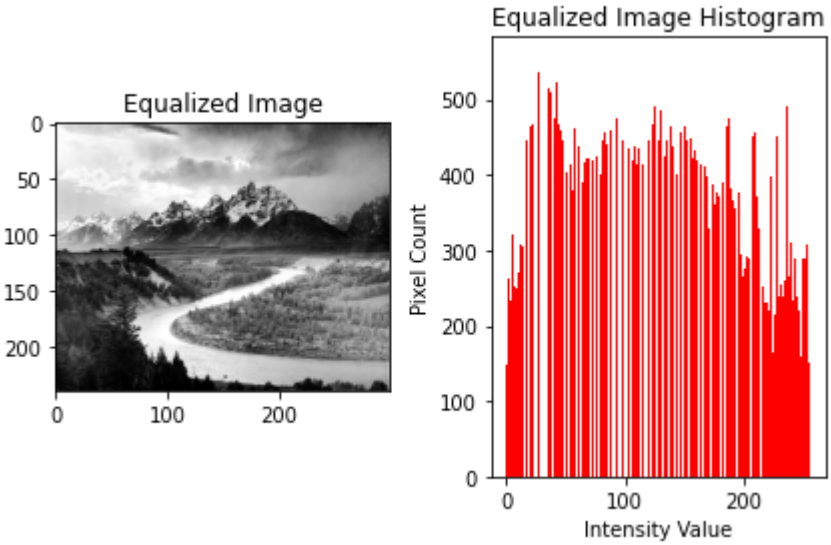
```
fig.tight_layout()
#display image
ax[0].imshow(img1,cmap="gray")
ax[0].set_title('Image')
#plot histogram of unequalized image
n = plt.hist(img1.flatten(),256,[0,256], color = 'r')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.title('Image Histogram')
plt.tight_layout()

fig, ax = plt.subplots(1,2)
fig.tight_layout()
#equalize histogram
eqImg1 = cv2.equalizeHist(img1)
#show equalized image
ax[0].imshow(eqImg1,cmap="gray")
ax[0].set_title('Equalized Image')
#show equalized image histogram
n = plt.hist(eqImg1.flatten(),256,[0,256], color = 'r')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.title('Equalized Image Histogram')
plt.tight_layout()

##second image
fig, ax = plt.subplots(1,2)
fig.tight_layout()
#display image
ax[0].imshow(img2,cmap="gray")
ax[0].set_title('Image')
#plot histogram of unequalized image
n = plt.hist(img2.flatten(),256,[0,256], color = 'r')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.title('Image Histogram')
plt.tight_layout()
fig, ax = plt.subplots(1,2)
fig.tight_layout()
#equalize histogram
eqImg2 = cv2.equalizeHist(img2)
#show equalized image
ax[0].imshow(eqImg2,cmap="gray")
ax[0].set_title('Equalized Image')
#show equalized image histogram
n = plt.hist(eqImg2.flatten(),256,[0,256], color = 'r')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.title('Equalized Image Histogram')
plt.tight_layout()

##third image
fig, ax = plt.subplots(1,2)
fig.tight_layout()
#display image
ax[0].imshow(img3,cmap="gray")
ax[0].set_title('Image')
#plot histogram of unequalized image
n = plt.hist(img3.flatten(),256,[0,256], color = 'r')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.title('Image Histogram')
plt.tight_layout()
fig, ax = plt.subplots(1,2)
fig.tight_layout()
#equalize histogram
eqImg3 = cv2.equalizeHist(img3)
#show equalized image
ax[0].imshow(eqImg3,cmap="gray")
ax[0].set_title('Equalized Image')
```

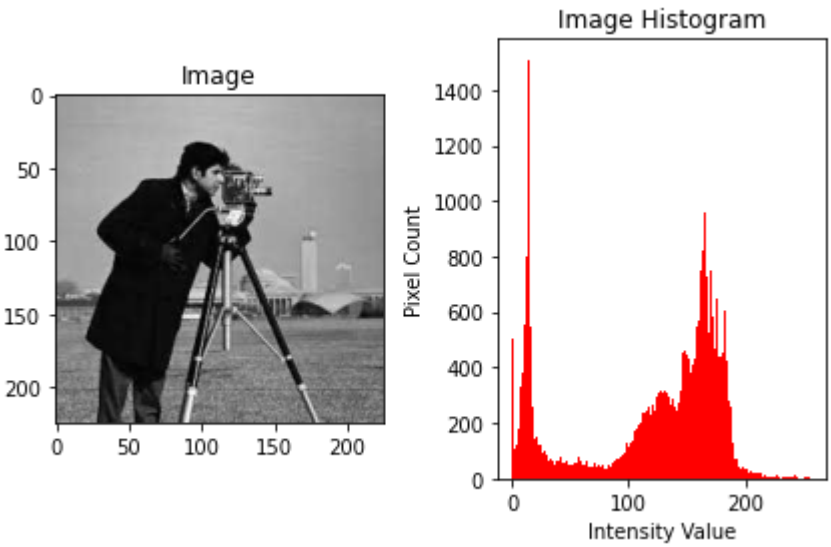
```
#show equalized image histogram
n = plt.hist(eqImg3.flatten(),256,[0,256], color = 'r')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.title('Equalized Image Histogram')
plt.tight_layout()
```

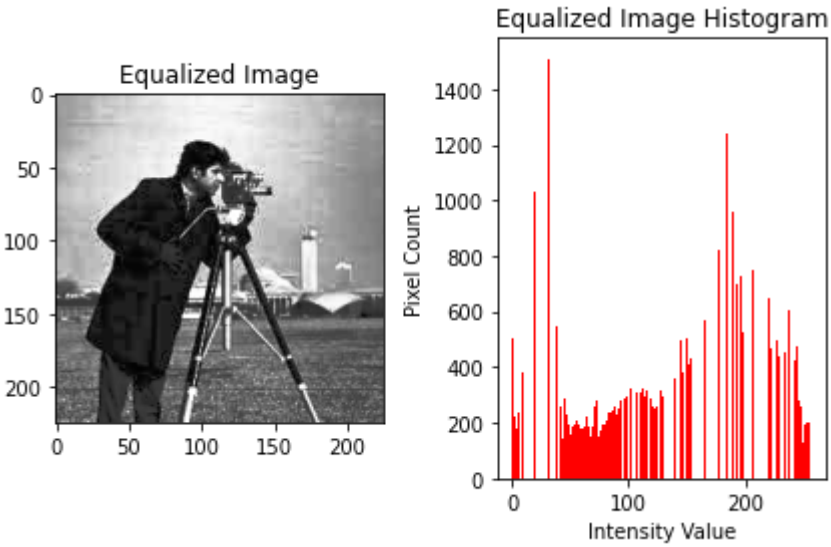












TASK 4

You are required to implement the convolution operation from scratch. This function which takes an image and a kernel and returns the convolution of them. Compare the results of your implemented function with the ones available (built-in) in python packages. You are required to convolve images for sharpening and blurring effects.

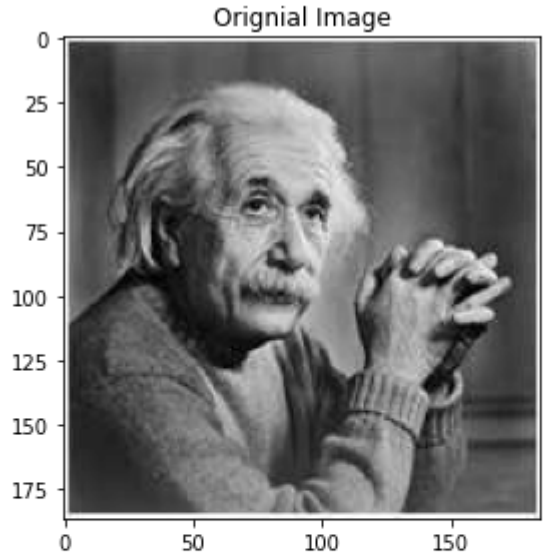
In [5]:
```
##################################     SMOOTHING     ########################

boxFilter3x3 = np.ones((3,3),dtype=float)
fig,ax = plt.subplots(1,3)
```

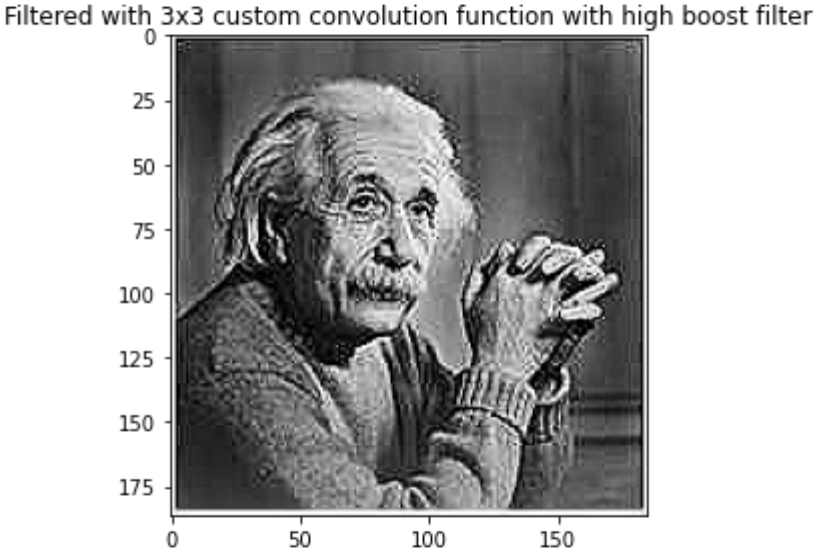
```
imgToBeFiltered = grayImgFiles[3]
ax[0].imshow(imgToBeFiltered,cmap="gray")
filteredImg = convolve(imgToBeFiltered,boxFilter3x3)
ax[1].imshow(filteredImg,cmap="gray")
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1,np.divide(boxFilter3x3,np.sum(boxF
ax[2].imshow(builtInConvolveFilteredImg,cmap="gray")
plt.title("Orignial Image (Left) VS Filtered with custom convolution 3x3 box filter (Center) VS

boxFilter5x5 = np.ones((5,5),dtype=int)
fig,ax = plt.subplots(1,3)
imgToBeFiltered = grayImgFiles[3]
ax[0].imshow(imgToBeFiltered,cmap="gray")
filteredImg = convolve(imgToBeFiltered,boxFilter5x5)
ax[1].imshow(filteredImg,cmap="gray")
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter5x5,np.sum(box
ax[2].imshow(builtInConvolveFilteredImg,cmap="gray")
plt.title("Orignial Image (Left) VS Filtered with custom convolution 5x5 box filter (Center) VS

boxFilter7x7 = np.ones((7,7),dtype=int)
fig,ax = plt.subplots(1,3)
imgToBeFiltered = grayImgFiles[3]
ax[0].imshow(imgToBeFiltered,cmap="gray")
filteredImg = convolve(imgToBeFiltered,boxFilter7x7)
ax[1].imshow(filteredImg,cmap="gray")
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter7x7,np.sum(box
ax[2].imshow(builtInConvolveFilteredImg,cmap="gray")
plt.title("Orignial Image (Left) VS Filtered with custom convolution 7x7 box filter (Center) VS
```

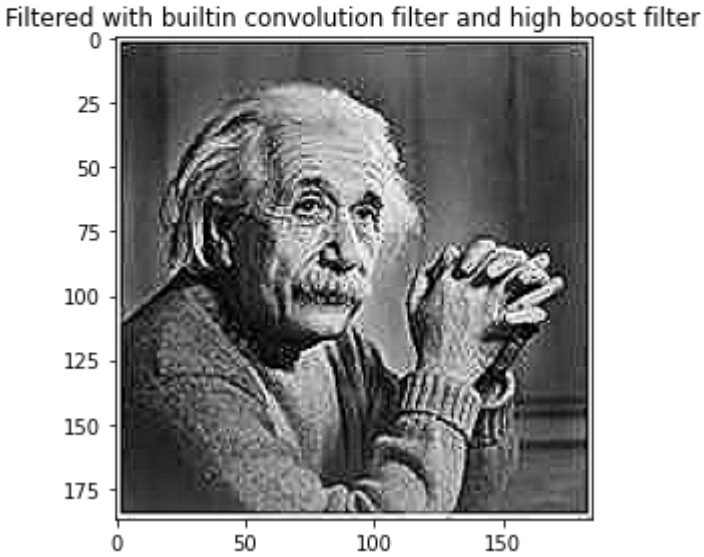
Out[5]:
```
Text(0.5, 1.0, 'Orignial Image (Left) VS Filtered with custom convolution 7x7 box filter (Cente
r) VS Filtered with builtin convolution filter (RIGHT)')
```

In [6]:
```
#### SHARPENING #####
#high boost filtering approach applied.
sharpenFilter3x3 = np.array([[0,-1,0],[-1,5,-1],[-0,-1,0]])
fig,ax = plt.subplots(3,1,figsize=(12,12))
imgToBeFiltered = grayImgFiles[3]
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
filteredImg = convolve(imgToBeFiltered,sharpenFilter3x3)
np.add(imgToBeFiltered ,filteredImg)
ax[1].imshow(filteredImg,cmap="gray")
ax[1].set_title('Filtered with 3x3 custom convolution function with high boost filter')
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1,sharpenFilter3x3)
ax[2].imshow(builtInConvolveFilteredImg,cmap="gray")
```

```
ax[2].set_title('Filtered with builtin convolution filter and high boost filter')
fig.tight_layout()
```







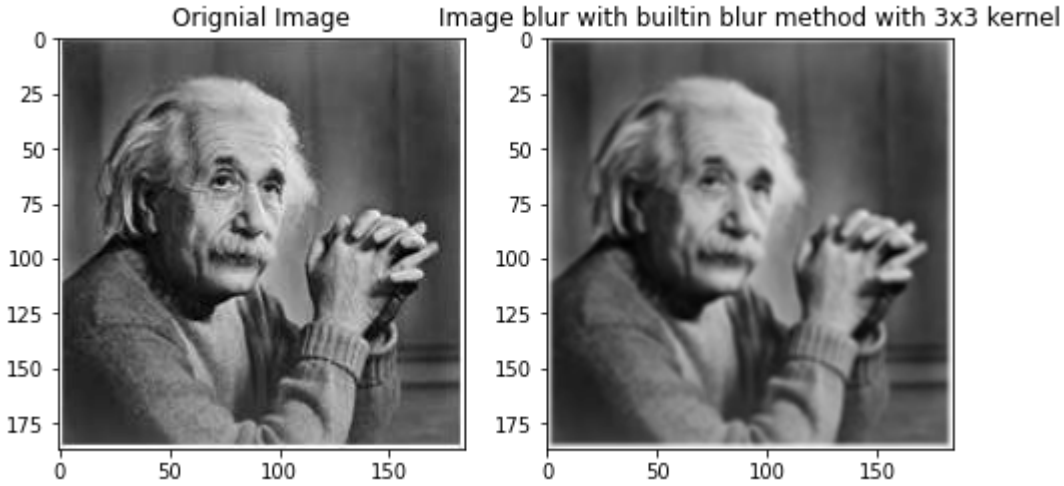
Using builtin blur and sharpening

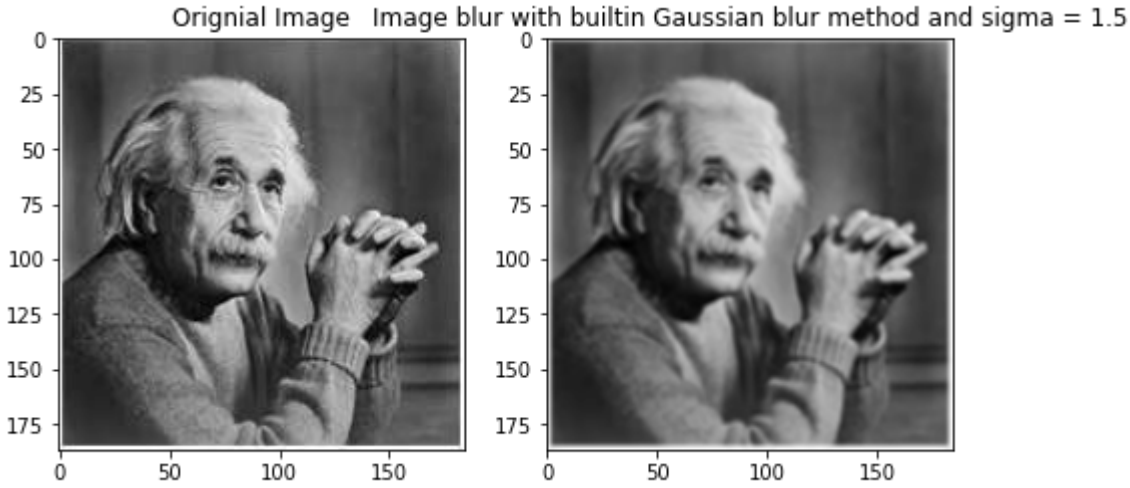
In [7]:

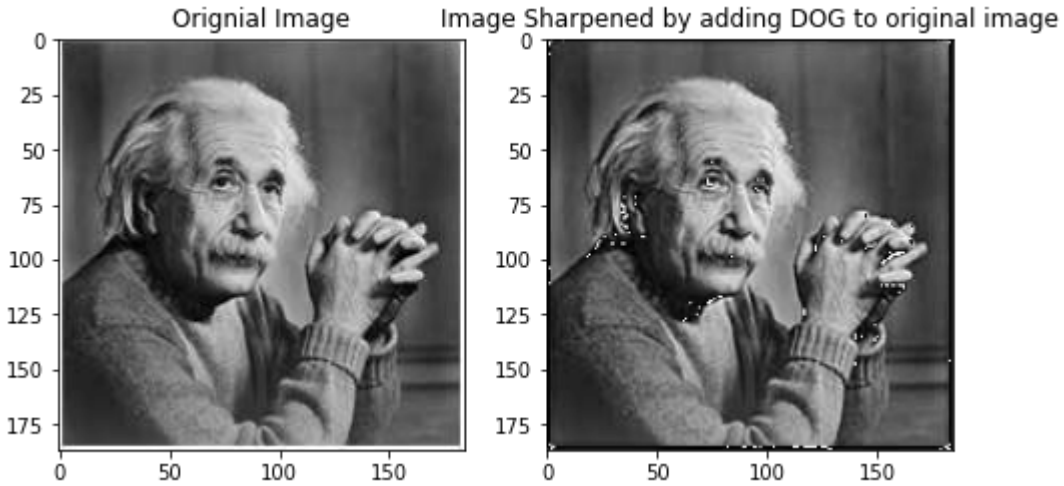
```
imgToBeFiltered = grayImgFiles[3]
fig,ax = plt.subplots(1,2,figsize=(8,8))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# Img blur with builtin blur method with 3x3 kernel
blurredImg = cv2.blur(imgToBeFiltered,(3,3))
ax[1].imshow(blurredImg,cmap='gray')
ax[1].set_title('Image blur with builtin blur method with 3x3 kernel')

fig,ax = plt.subplots(1,2,figsize=(8,8))
```

```
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# Image blur with builtin Gaussian blur method and sigma = 1.5
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(3,3),1.5)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('Image blur with builtin Gaussian blur method and sigma = 1.5')

fig,ax = plt.subplots(1,2,figsize=(8,8))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
blurredImg1 = cv2.GaussianBlur(imgToBeFiltered,(5,5),1)
blurredImg2 = cv2.GaussianBlur(imgToBeFiltered,(5,5),1.5)
# taking DOG
differenceImg = np.subtract(blurredImg1,blurredImg2)
# Image Sharpened by adding DOG to original image
EnhancedImg = np.add(imgToBeFiltered,differenceImg)
ax[1].imshow(EnhancedImg,cmap='gray')
ax[1].set_title('Image Sharpened by adding DOG to original image')
```

Out[7]: Text(0.5, 1.0, 'Image Sharpened by adding DOG to original image')





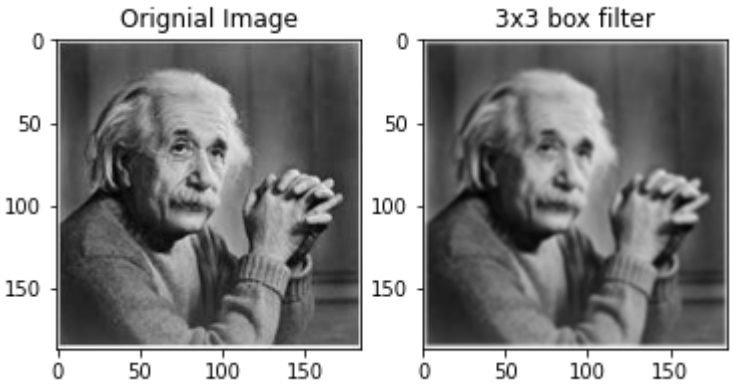


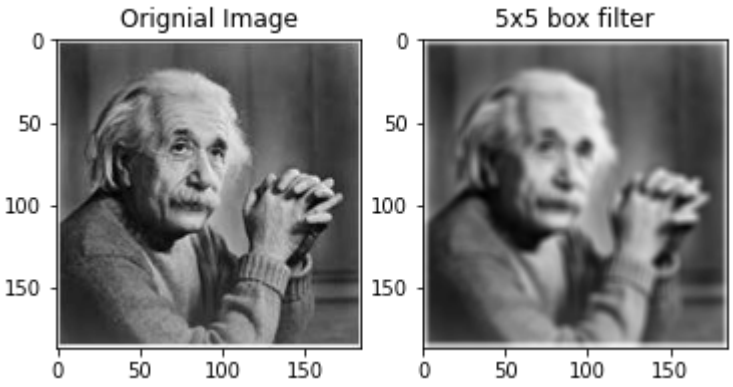
TASK 5

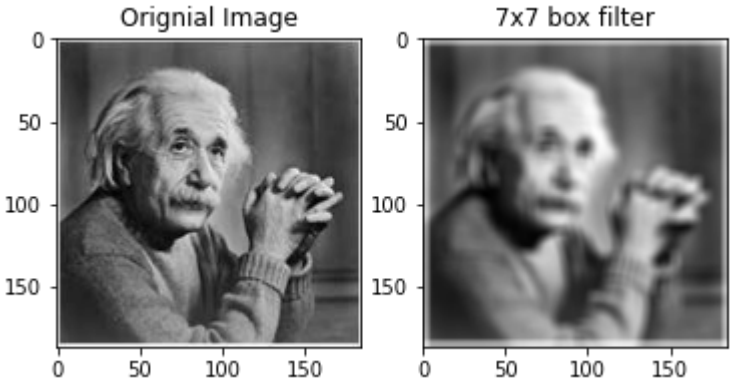
Apply box filter using convolution, and display the resultant image

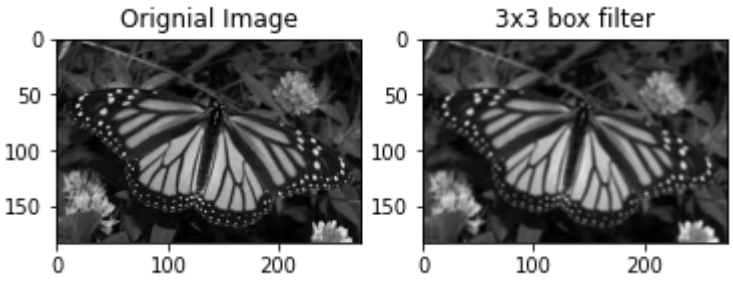
In [8]:
```
boxFilter3x3 = np.ones((3,3),dtype=float)
boxFilter5x5 = np.ones((5,5),dtype=float)
boxFilter7x7 = np.ones((7,7),dtype=float)

#box filter 3x3
imgToBeFiltered = grayImgFiles[3]
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# convolving using builtin filter2D method
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter3x3,np.sum(box
ax[1].imshow(builtInConvolveFilteredImg,cmap="gray")
ax[1].set_title('3x3 box filter')

#box filter 5x5
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# convolving using builtin filter2D method
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter5x5,np.sum(box
ax[1].imshow(builtInConvolveFilteredImg,cmap="gray")
ax[1].set_title('5x5 box filter')

#box filter 7x7
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# convolving using builtin filter2D method
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter7x7,np.sum(box
ax[1].imshow(builtInConvolveFilteredImg,cmap="gray")
ax[1].set_title('7x7 box filter')

#box filter 3x3
imgToBeFiltered = grayImgFiles[4]
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# convolving using builtin filter2D method
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter3x3,np.sum(box
ax[1].imshow(builtInConvolveFilteredImg,cmap="gray")
ax[1].set_title('3x3 box filter')

#box filter 5x5
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# convolving using builtin filter2D method
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter5x5,np.sum(box
ax[1].imshow(builtInConvolveFilteredImg,cmap="gray")
ax[1].set_title('5x5 box filter')

#box filter 7x7
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Orignial Image')
# convolving using builtin filter2D method
builtInConvolveFilteredImg = cv2.filter2D(imgToBeFiltered,-1, np.divide(boxFilter7x7,np.sum(box
ax[1].imshow(builtInConvolveFilteredImg,cmap="gray")
ax[1].set_title('7x7 box filter')
```

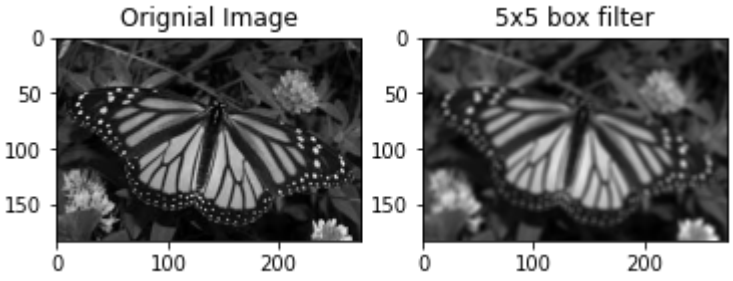
Out[8]:
```
Text(0.5, 1.0, '7x7 box filter')
```

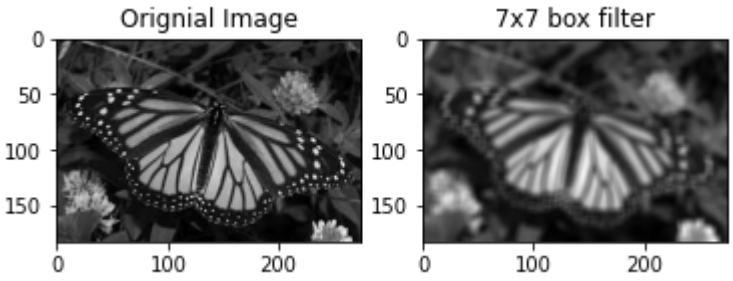








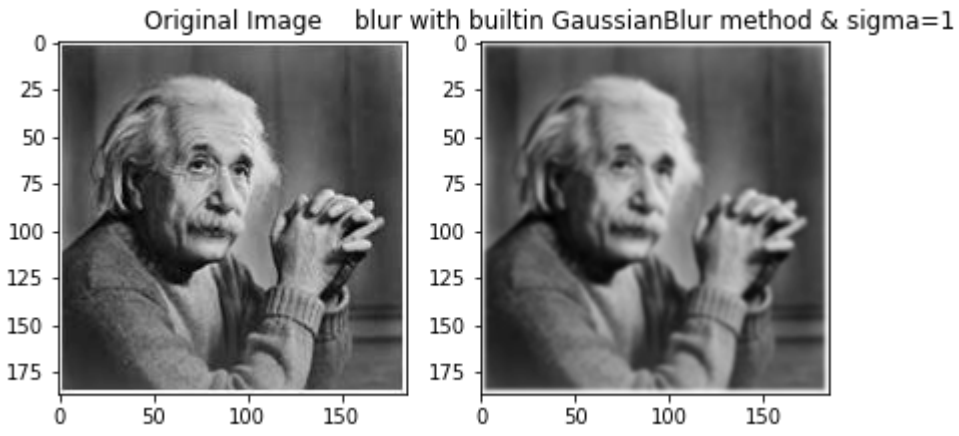


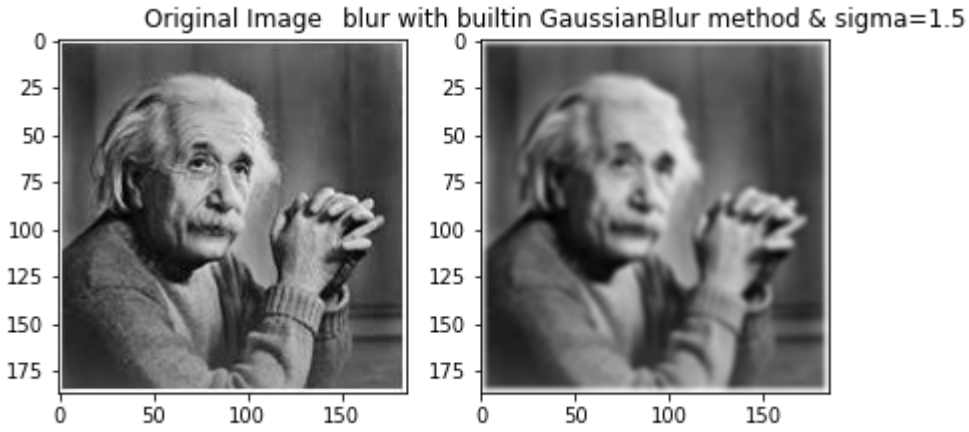


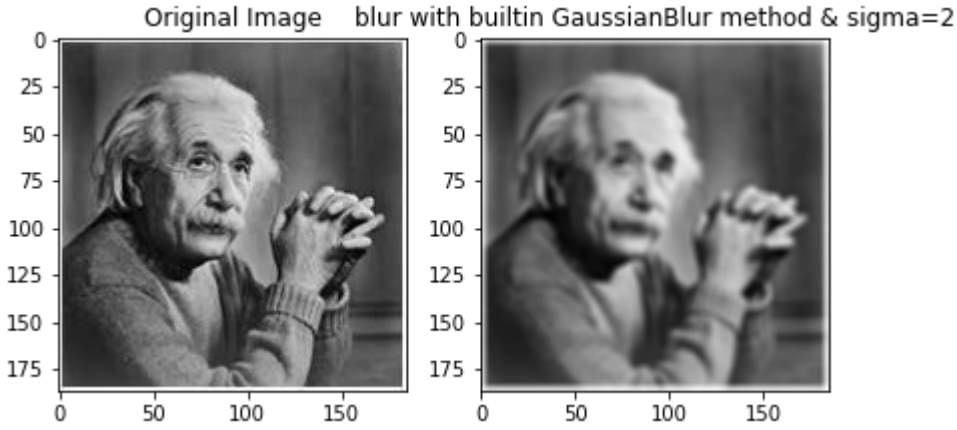
Apply Gaussian filter to the image, with varying sigma values.

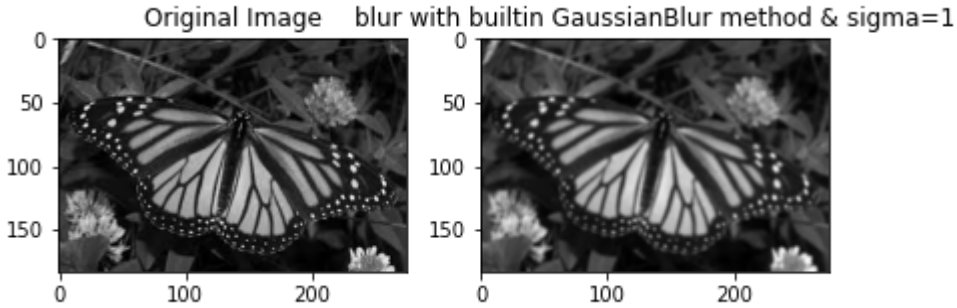
In [9]:

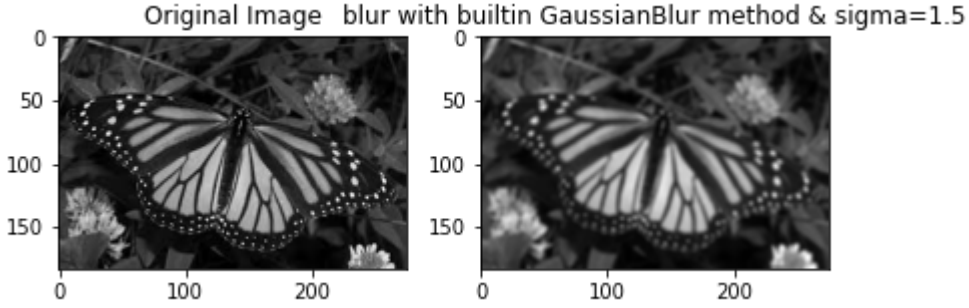
```
imgToBeFiltered = grayImgFiles[3]
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Original Image')
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(5,5),1)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('blur with builtin GaussianBlur method & sigma=1')
fig.tight_layout()

fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Original Image')
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(5,5),1.5)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('blur with builtin GaussianBlur method & sigma=1.5')
fig.tight_layout()

fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Original Image')
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(5,5),2)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('blur with builtin GaussianBlur method & sigma=2')
fig.tight_layout()

imgToBeFiltered = grayImgFiles[4]
fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Original Image')
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(5,5),1)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('blur with builtin GaussianBlur method & sigma=1')
fig.tight_layout()

fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Original Image')
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(5,5),1.5)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('blur with builtin GaussianBlur method & sigma=1.5')
fig.tight_layout()

fig,ax = plt.subplots(1,2,figsize=(6,6))
ax[0].imshow(imgToBeFiltered,cmap="gray")
ax[0].set_title('Original Image')
blurredGaussianImg = cv2.GaussianBlur(imgToBeFiltered,(5,5),2)
ax[1].imshow(blurredGaussianImg,cmap='gray')
ax[1].set_title('blur with builtin GaussianBlur method & sigma=2')
fig.tight_layout()
```

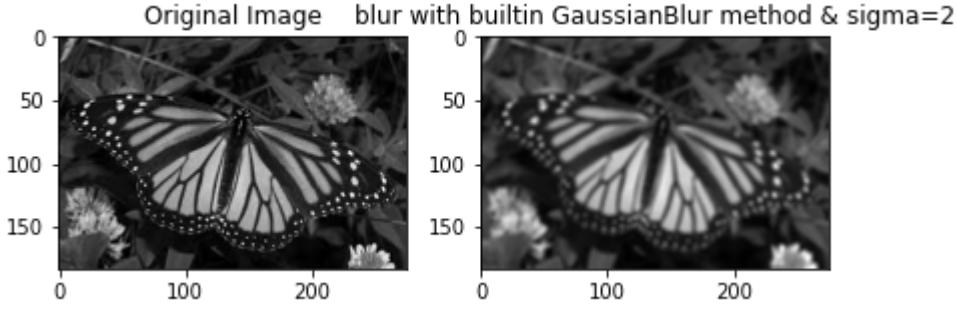












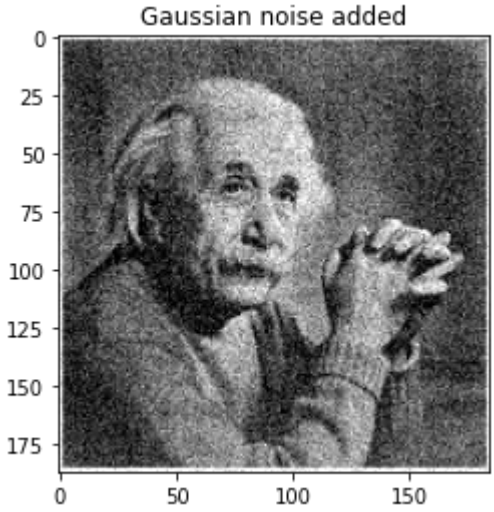
Add Gausian Noise and Salt and Pepper Noise to them.

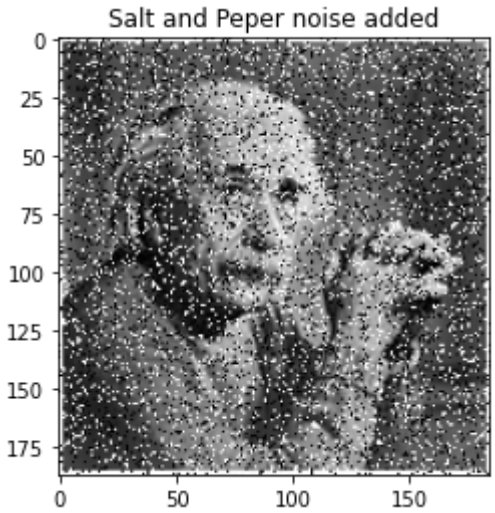
In [10]:

```
from skimage.util import random_noise

imgToWhichNoiseIsAdded = grayImgFiles[3]

# Add gaussian noise to the image
gaussianNoiseImage = random_noise(imgToWhichNoiseIsAdded, mode='gaussian')
```

```
fig, aux = plt.subplots(1,1,figsize=(4,4))
aux.imshow(gaussianNoiseImage,cmap="gray")
plt.title("Gaussian noise added")

# Add salt-and-pepper noise to the image.
saltNPeperNoiseImage= random_noise(imgToWhichNoiseIsAdded, mode='s&p',amount=0.2)
fig, aux = plt.subplots(1,1,figsize=(4,4))
aux.imshow(saltNPeperNoiseImage,cmap="gray")
plt.title("Salt and Peper noise added")
```

Out[10]: Text(0.5, 1.0, 'Salt and Peper noise added')



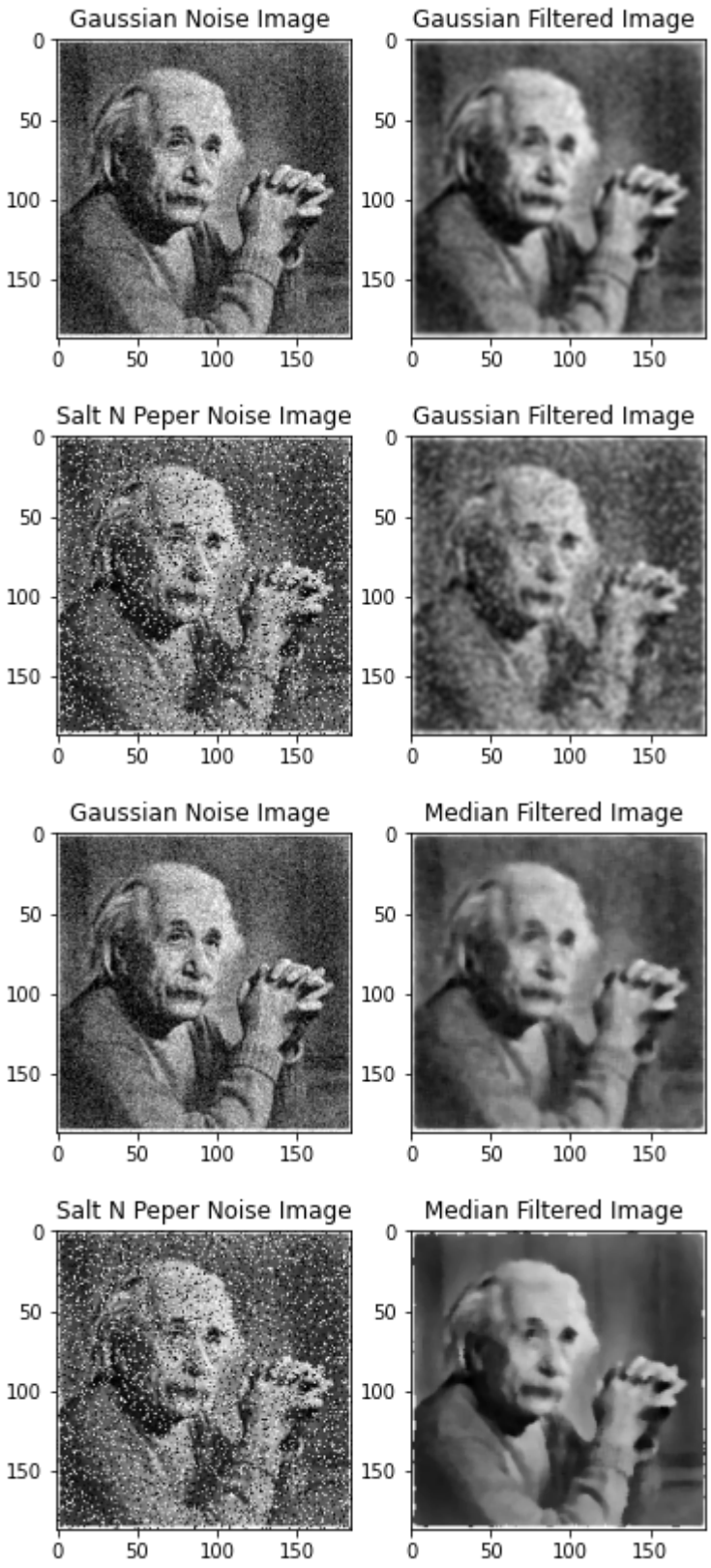


Apply Gaussian Filter and Median Filters

In [11]:
```
#applying gaussian filter to gaussian noise image
gsFilteredGsNoiseImg = cv2.GaussianBlur(gaussianNoiseImage,(5,5),1.5)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(gaussianNoiseImage,cmap="gray")
aux[0].set_title('Gaussian Noise Image ')
aux[1].imshow(gsFilteredGsNoiseImg,cmap="gray")
aux[1].set_title('Gaussian Filtered Image ')

#applying gaussian filter to salt and peper noise image
gsFilteredSnNoiseImg = cv2.GaussianBlur(saltNPeperNoiseImage,(5,5),1.5)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(saltNPeperNoiseImage,cmap="gray")
aux[0].set_title('Salt N Peper Noise Image')
aux[1].imshow(gsFilteredSnNoiseImg,cmap="gray")
aux[1].set_title('Gaussian Filtered Image ')

#applying median filter to gaussian noise image
mdFilteredGsNoiseImg = cv2.medianBlur(gaussianNoiseImage.astype(np.float32),5)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(gaussianNoiseImage,cmap="gray")
aux[0].set_title('Gaussian Noise Image ')
```

```
aux[1].imshow(mdFilteredGsNoiseImg,cmap="gray")
aux[1].set_title('Median Filtered Image ')

#applying median filter to salt and peper noise image
mdFilteredSnNoiseImg = cv2.medianBlur(saltNPeperNoiseImage.astype(np.float32),5)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(saltNPeperNoiseImage,cmap="gray")
aux[0].set_title('Salt N Peper Noise Image')
aux[1].imshow(mdFilteredSnNoiseImg,cmap="gray")
aux[1].set_title('Median Filtered Image ')
```

Out[11]: Text(0.5, 1.0, 'Median Filtered Image ')



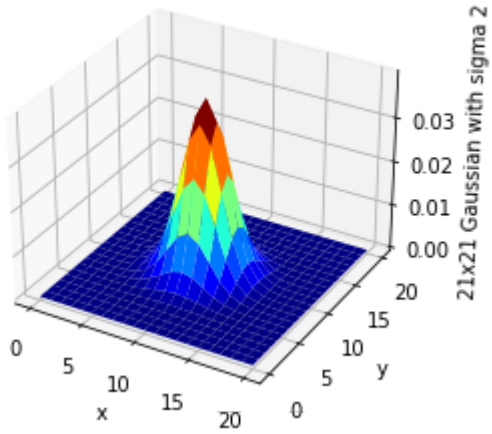
Display mesh plots for different i) Gaussian filters, ii) First Order Derivative of Gaussian, iii) Laplacian of Gaussian; using different sigma values

In [12]:

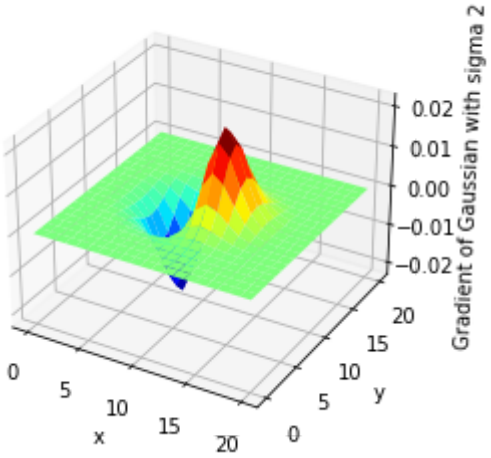
```
############ 21x21 kernel with SIGMA 2 #############

gaussianFilter21x21 = cv2.getGaussianKernel(21,2)
gaussian2Dkernel = gaussianFilter21x21 @ gaussianFilter21x21.transpose()

x, y = np.mgrid[0:gaussian2Dkernel.shape[0], 0:gaussian2Dkernel.shape[1]]
z = gaussian2Dkernel
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('21x21 Gaussian with sigma 2')
ax.set_title('Gaussian Mesh Plot for sigma 2')
plt.show()
#plt.title('21x21 Gaussian filter with sigma 2')

differenceFilter = np.array([1,0,-1])
# calculating gradient of gaussian by convolving 2D gaussian kernel with finite difference filt
firstOrderGaussian = cv2.filter2D(gaussian2Dkernel,-1,differenceFilter)
x, y = np.mgrid[0:firstOrderGaussian.shape[0], 0:firstOrderGaussian.shape[1]]
z = firstOrderGaussian
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('Gradient of Gaussian with sigma 2')
ax.set_title('Gradient of Gaussian Mesh Plot for sigma 2')
plt.show()

laplacianFilter = np.array([1,0,-1])
# calculating LOG by convolving gradient of gaussian with finite difference filter
laplacianOfGaussian = cv2.filter2D(firstOrderGaussian,-1,laplacianFilter)
x, y = np.mgrid[0:laplacianOfGaussian.shape[0], 0:laplacianOfGaussian.shape[1]]
z = laplacianOfGaussian
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('LOG with sigma 2')
ax.set_title('Laplacian of Gaussian Mesh Plot for sigma 2')
plt.show()

############### 21x21 kernel with SIGMA 3 ###################
gaussianFilter21x21 = cv2.getGaussianKernel(21,3)
gaussian2Dkernel = gaussianFilter21x21 @ gaussianFilter21x21.transpose()

x, y = np.mgrid[0:gaussian2Dkernel.shape[0], 0:gaussian2Dkernel.shape[1]]
z = gaussian2Dkernel
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('21x21 Gaussian with sigma 3')
ax.set_title('Gaussian Mesh Plot for sigma 3')
plt.show()
#plt.title('21x21 Gaussian filter with sigma 2')

differenceFilter = np.array([1,0,-1])
# calculating gradient of gaussian by convolving 2D gaussian kernel with finite difference filt
firstOrderGaussian = cv2.filter2D(gaussian2Dkernel,-1,differenceFilter)
x, y = np.mgrid[0:firstOrderGaussian.shape[0], 0:firstOrderGaussian.shape[1]]
z = firstOrderGaussian
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, cmap=cm.jet)
ax.set_xlabel('x')
```

```
ax.set_ylabel('y')
ax.set_zlabel('Gradient of Gaussian with sigma 3')
ax.set_title('Gradient of Gaussian Mesh Plot for sigma 3')
plt.show()

laplacianFilter = np.array([1,0,-1])
# calculating LOG by convolving gradient of gaussian with finite difference filter
laplacianOfGaussian = cv2.filter2D(firstOrderGaussian,-1,laplacianFilter)
x, y = np.mgrid[0:laplacianOfGaussian.shape[0], 0:laplacianOfGaussian.shape[1]]
z = laplacianOfGaussian
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('LOG with sigma 3')
ax.set_title('Laplacian of Gaussian Mesh Plot for sigma 3')
plt.show()
```

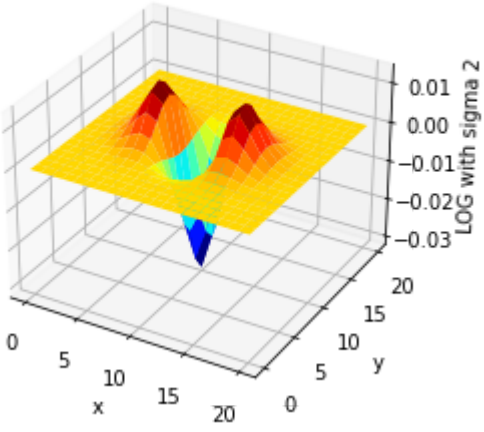
Gaussian Mesh Plot for sigma 2

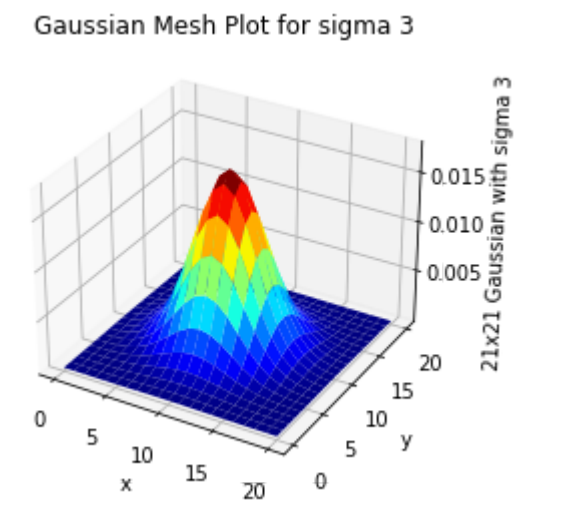


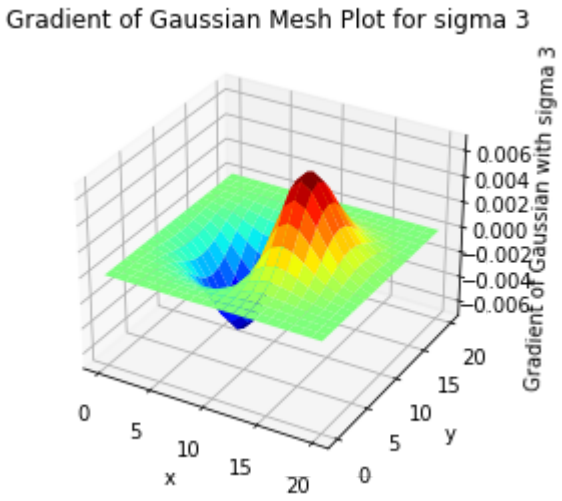
Gradient of Gaussian Mesh Plot for sigma 2

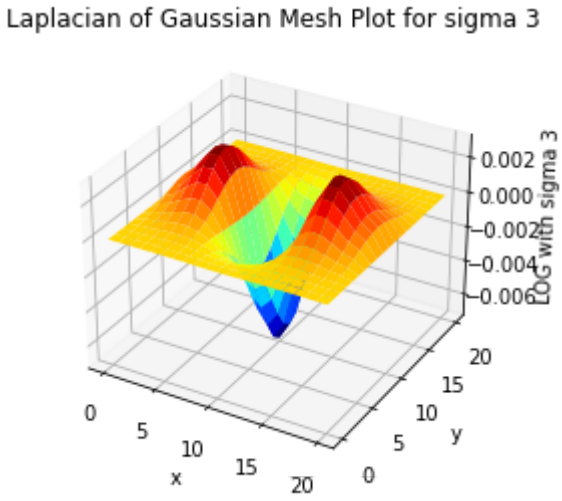


Laplacian of Gaussian Mesh Plot for sigma 2









TASK 6

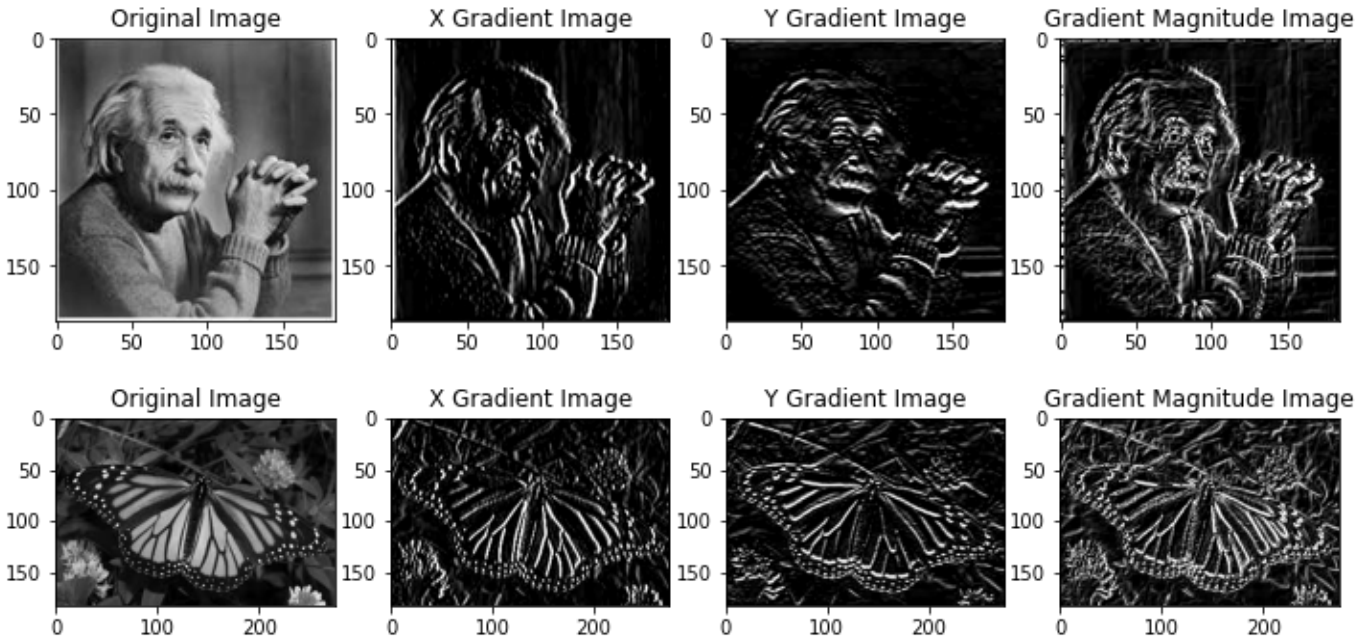
Apply Sobel operator, computer gradient magnitude and display the results (original image, gradient images and gradient magnitude image)

In [13]:

```
sobelOperatorFilterX = np.array([[1,0,-1],[2,0,-2],[1,0,-1]], dtype='float')
sobelOperatorFilterY = np.array([[1,2,1],[0,0,0],[-1,-2,-1]], dtype='float')
inputImage = grayImgFiles[3]
sobelFilteredImage1X = cv2.filter2D(inputImage,-1,sobelOperatorFilterX)
sobelFilteredImage1Y = cv2.filter2D(inputImage,-1,sobelOperatorFilterY)
# For gradient images Gx Gy, Gradient magnitude = abs(Gx) + abs(Gy)
sobelFilteredImage1Mag = np.add(abs(sobelFilteredImage1X),abs(sobelFilteredImage1Y))
fig, aux = plt.subplots(1,4,figsize=(12,12))
aux[0].imshow(inputImage,cmap='gray')
aux[0].set_title('Original Image')
aux[1].imshow(sobelFilteredImage1X,cmap='gray')
aux[1].set_title('X Gradient Image')
```

```
aux[2].imshow(sobelFilteredImage1Y,cmap='gray')
aux[2].set_title('Y Gradient Image')
aux[3].imshow(sobelFilteredImage1Mag,cmap='gray')
aux[3].set_title('Gradient Magnitude Image')

inputImage = grayImgFiles[4]
sobelFilteredImage1X = cv2.filter2D(inputImage,-1,sobelOperatorFilterX)
sobelFilteredImage1Y = cv2.filter2D(inputImage,-1,sobelOperatorFilterY)
# For gradient images Gx Gy, Gradient magnitude = abs(Gx) + abs(Gy)
sobelFilteredImage1Mag = np.add(abs(sobelFilteredImage1X),abs(sobelFilteredImage1Y))
fig, aux = plt.subplots(1,4,figsize=(12,12))
aux[0].imshow(inputImage,cmap='gray')
aux[0].set_title('Original Image')
aux[1].imshow(sobelFilteredImage1X,cmap='gray')
aux[1].set_title('X Gradient Image')
aux[2].imshow(sobelFilteredImage1Y,cmap='gray')
aux[2].set_title('Y Gradient Image')
aux[3].imshow(sobelFilteredImage1Mag,cmap='gray')
aux[3].set_title('Gradient Magnitude Image')
```

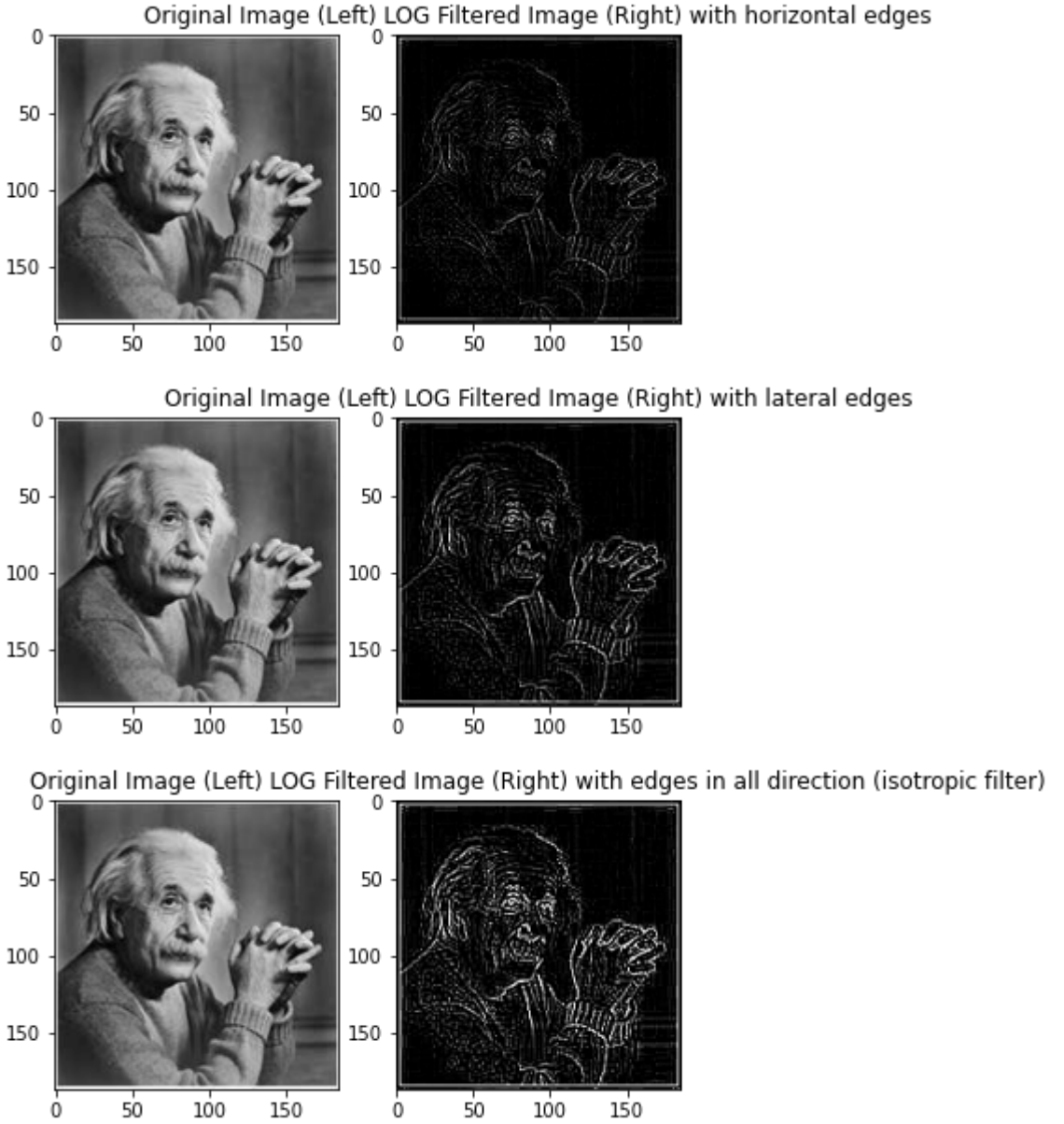
Out[13]: Text(0.5, 1.0, 'Gradient Magnitude Image')



In [14]:

```
inputImageLogFiltering = grayImgFiles[3]
logFilterHorizontal = np.array([[0,1,0],[1,-4,1],[0,1,0]],dtype='float')
# now applying horizontal LOG filter to image to find edges in horizontal and vertical neighbou
horizontalLOGfilteredImage = cv2.filter2D(inputImageLogFiltering,-1, logFilterHorizontal)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(inputImageLogFiltering, cmap='gray')
aux[1].imshow(horizontalLOGfilteredImage, cmap='gray')
plt.title("Original Image (Left) LOG Filtered Image (Right) with horizontal edges")

logFilterLateral = np.array([[1,0,1],[0,-4,0],[1,0,1]],dtype='float')
# now applying lateral LOG filter to image to find edges at 45 degree or lateral
lateralLOGfilteredImage = cv2.filter2D(inputImageLogFiltering,-1, logFilterLateral)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(inputImageLogFiltering, cmap='gray')
aux[1].imshow(lateralLOGfilteredImage, cmap='gray')
plt.title("Original Image (Left) LOG Filtered Image (Right) with lateral edges")

# isotropic filter finds edges on all 8 direction
logFilterIsotropic = np.array([[1,1,1],[1,-8,1],[1,1,1]],dtype='float')
# now applying lateral LOG filter to image to find edges in all 8 directions
isotropicLOGfilteredImage = cv2.filter2D(inputImageLogFiltering,-1, logFilterIsotropic)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(inputImageLogFiltering, cmap='gray')
aux[1].imshow(isotropicLOGfilteredImage, cmap='gray')
plt.title("Original Image (Left) LOG Filtered Image (Right) with edges in all direction (isotrc
```

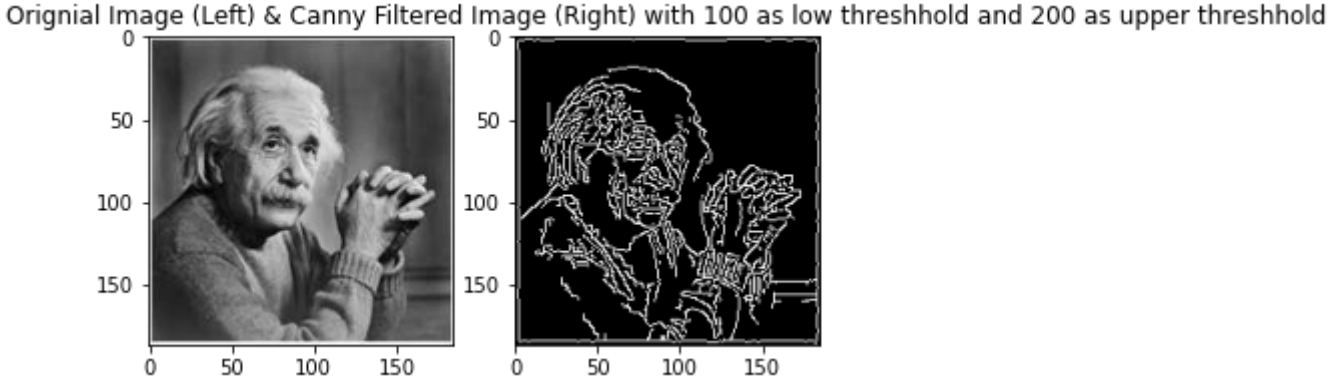
Out[14]: Text(0.5, 1.0, 'Original Image (Left) LOG Filtered Image (Right) with edges in all direction (isotropic filter)')



Apply Canny Edge Detector and display the results.

In [15]:
```
inputImageCannyFiltering = grayImgFiles[3]
# applying builtin canny filter with high threshhold of 200 and low threshhold of 100
cannyFilteredImg = cv2.Canny(inputImageCannyFiltering,100,200)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(inputImageCannyFiltering, cmap='gray')
aux[1].imshow(cannyFilteredImg, cmap='gray')
plt.title('Orignial Image (Left) & Canny Filtered Image (Right) with 100 as low threshhold and
```

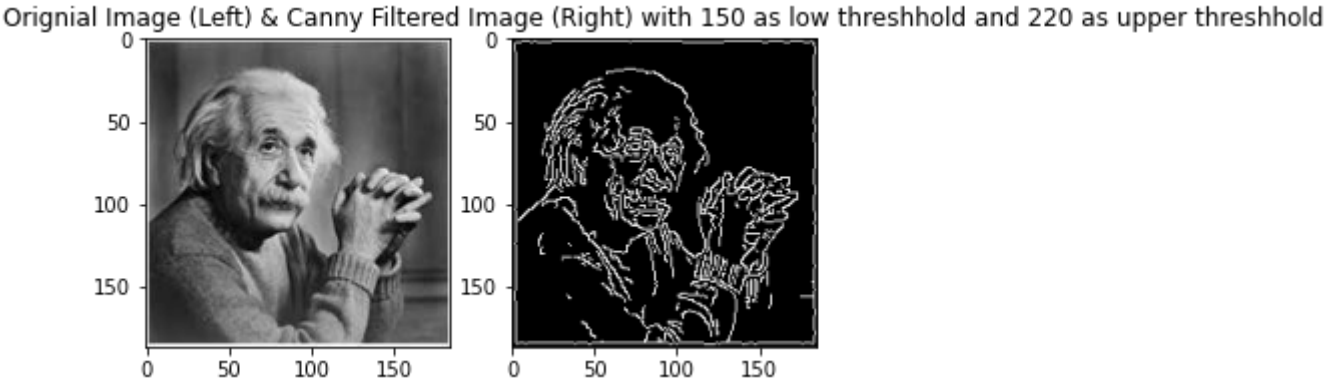
Out[15]: Text(0.5, 1.0, 'Orignial Image (Left) & Canny Filtered Image (Right) with 100 as low threshhold and 200 as upper threshhold')



In [16]:
```
inputImageCannyFiltering = grayImgFiles[3]
# applying builtin canny filter with high threshhold of 220 and low threshhold of 150
```

```
cannyFilteredImg = cv2.Canny(inputImageCannyFiltering,150,220)
fig, aux = plt.subplots(1,2,figsize=(6,6))
aux[0].imshow(inputImageCannyFiltering, cmap='gray')
aux[1].imshow(cannyFilteredImg, cmap='gray')
plt.title('Orignial Image (Left) & Canny Filtered Image (Right) with 150 as low threshhold and
```

Out[16]: Text(0.5, 1.0, 'Orignial Image (Left) & Canny Filtered Image (Right) with 150 as low threshhold and 220 as upper threshhold')



In [ ]: