**Muhammad Awais Suhail BCSF16A507**

**Computer Vision**

**Assignment # 1**

**Question1:**

**Code:**

# import the necessary packages  
import numpy as np  
import cv2  
from PIL import Image  
  
def convolve(image, kernel):  
 # convert into grey scale  
  
 kernel = np.flipud(np.fliplr(kernel)) # Flip the kernel  
  
 iR = image.shape[0]  
 iC = image.shape[1]  
 kR = kernel.shape[0]  
 kC = kernel.shape[1]  
 #no of pad rows  
 pad = (kC - 1) // 2  
  
 image = np.pad(image, pad, mode=**'constant'**, constant\_values=0)  
 output = np.zeros((iR, iC), dtype=**"float32"**)  
 iR1 = image.shape[0]  
 iC1 = image.shape[1]  
  
 for x in np.arange(pad, iR1 - pad):  
 for y in np.arange(pad, iC1 - pad):  
 k = 0  
 for i in range(0, kR):  
 for j in range(0, kC):  
 k = k + (kernel[i][j] \* image[x - pad + i, y - pad + j])  
  
 output[x-pad, y-pad] = k  
  
 return output  
  
gaussian = np.array((  
 [1/16, 2/16, 1/16],  
 [2/16, 4/16, 2/16],  
 [1/16, 2/16, 1/16]), dtype=**"float"**)  
  
# load the image in rgb  
image = cv2.imread(**'dollar.tif'**)  
image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
  
print(**"Applying the Gausian Kernel"**)  
#Getting the Convolution Result  
Output = convolve(image, gaussian)  
  
#converting the image and output arrays to PIL images  
Output = Image.fromarray(Output)  
image = Image.fromarray(image)  
  
#image.show("Original Image")  
Output.show(**"Convolved Image"**)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Results:**

Original:



Convolved with Gaussian:



**Question2:**

1. The given operations are linear as multiplication of a matrix with a scaler is a linear operation and we are multiplying the corresponding entries of the matrix and then adding all the values all these operations are linear so applying blurring and finding ***gy*** is also linear.
2. If we apply B first and then find ***gy*** then we will have true edges and also fine edges. If we apply ***gy*** and then blur the image then

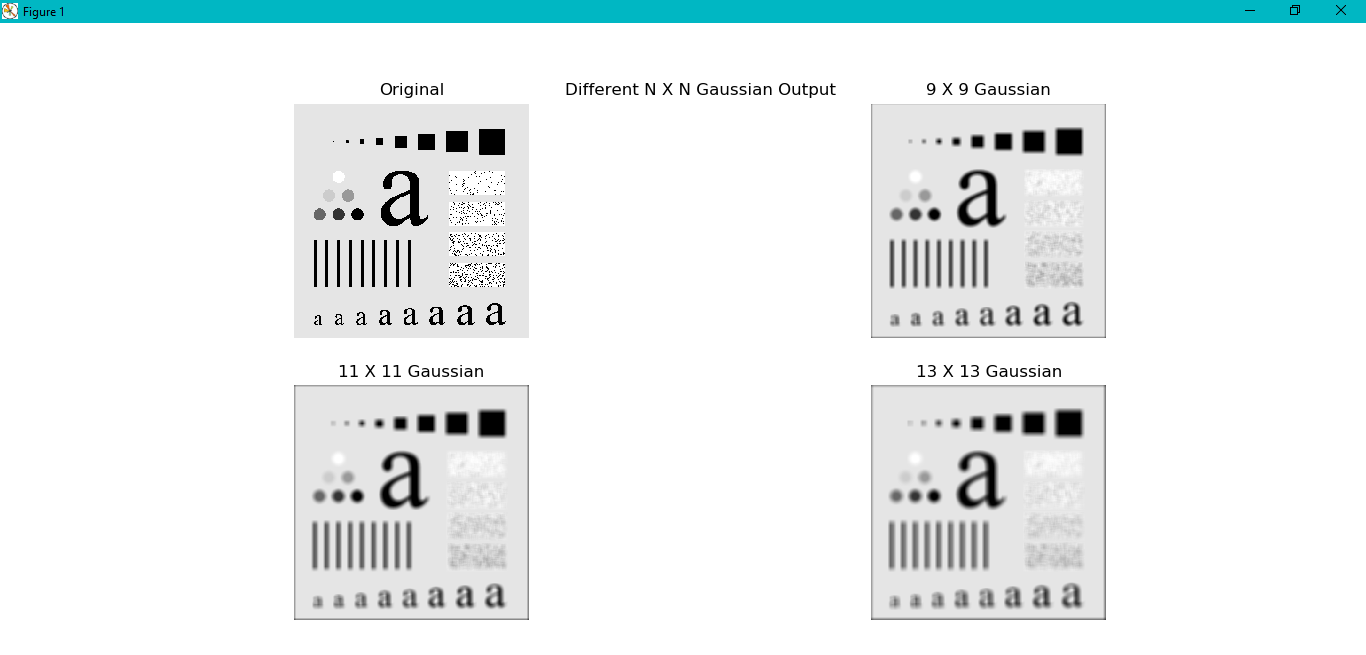
We will have false edges.

1. We can convolve two filters with each other and then apply it on the image.

**Question 3:**

**Code:**

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
  
from PIL import Image  
  
def my2dGausian(size):  
 w, h = size,size  
 Mask = [[0] \* w for i in range(h)]  
 sigma = 1  
 sum = 0  
 for i in range(w):  
 for j in range(h):  
 den = 2\*(sigma\*\*2)  
 neu = ((i-1)\*\*2)+((j-1)\*\*2)  
 neu = -neu  
 res = neu/den  
 res = pow(2.71828, res)  
 Mask[i][j] = res  
 sum = sum+Mask[i][j]  
 Mask = np.array(Mask)  
 #Normalizing the Mask  
 for i in range(w):  
 for j in range(h):  
 Mask[i][j] = Mask[i][j] / sum  
 return Mask  
  
def convolve(image, kernel):  
 kernel = np.flipud(np.fliplr(kernel)) # Flip the kernel  
 iR = image.shape[0]  
 iC = image.shape[1]  
 kR = kernel.shape[0]  
 kC = kernel.shape[1]  
 # no of pad rows  
 pad = (kC - 1) // 2  
 image = np.pad(image, pad, mode=**'constant'**, constant\_values=0)  
 output = np.zeros((iR, iC), dtype=**"float32"**)  
  
 iR1 = image.shape[0]  
 iC1 = image.shape[1]  
  
 for x in np.arange(pad, iR1 - pad):  
 for y in np.arange(pad, iC1 - pad):  
 k = 0  
 for i in range(0, kR):  
 for j in range(0, kC):  
 k = k + (kernel[i][j] \* image[x - pad + i, y - pad + j])  
 output[x - pad, y - pad] = k  
 return output  
  
  
image = cv2.imread(**"pattern.tif"**)  
image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
kernel9 = my2dGausian(9)  
kernel11 = my2dGausian(11)  
kernel13 = my2dGausian(13)  
  
  
#Convolution with NXN filters and converting them into PIL grey scale images  
  
Output9 = Image.fromarray(convolve(image, kernel9))  
Output11 = Image.fromarray(convolve(image, kernel11))  
Output13 = Image.fromarray(convolve(image, kernel13))  
image = Image.fromarray(image)  
#ploting code  
fig = plt.figure(figsize=(8, 8))  
columns = 2  
rows = 2  
  
plt.title(**"Different N X N Gaussian Output"**)  
plt.axis(**'off'**)  
images = [image, Output9, Output11, Output13]  
titles = [**"Original"**, **"9 X 9 Gaussian"**, **"11 X 11 Gaussian"**, **"13 X 13 Gaussian"**]  
  
for i in range(1, columns \* rows + 1):  
 ax = fig.add\_subplot(rows, columns, i)  
 ax.title.set\_text(titles[i-1])  
 plt.axis(**'off'**)  
 plt.imshow(images[i-1], cmap=**'gray'**)  
 plt.axis(**'off'**)  
plt.show()  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Results: **

c)

**Question 4**

a)

The expression for the piecewise linear transformation to stretch the contrast for fig 3.10 is given as follows:

((s2 - s1) / (r2 - r1)) \* (pix - r1) + s1

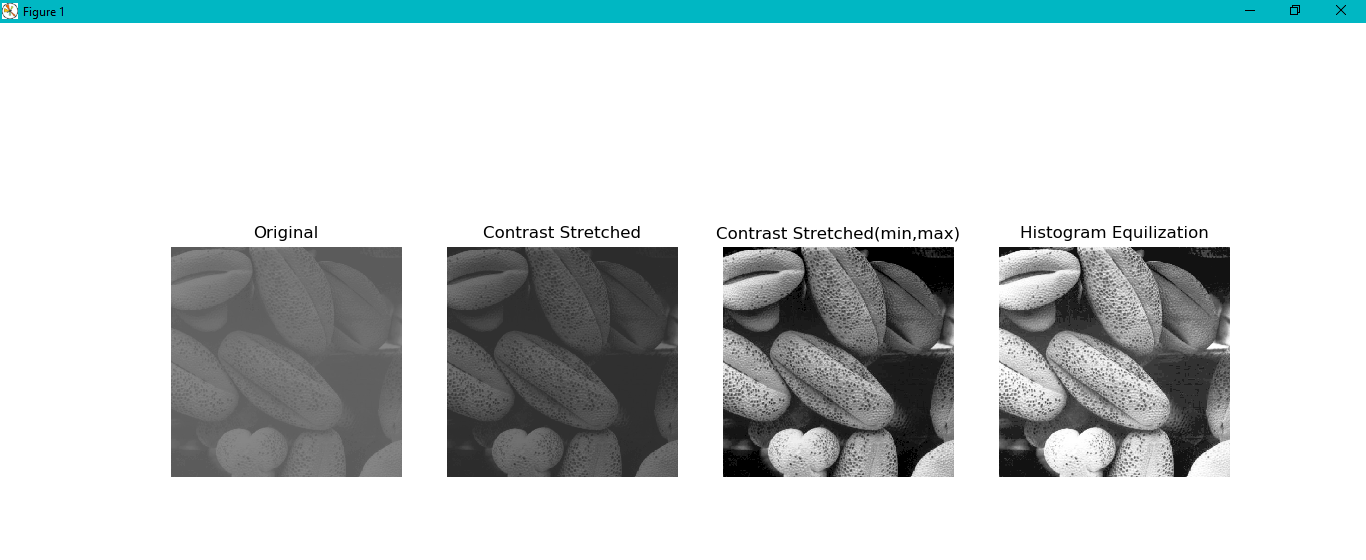
((230 - 50) / (200 - 100)) \* (pix - 100) + 50

b,c,d)

**Code:**

import cv2  
import numpy as np  
from PIL import Image  
import matplotlib.pyplot as plt  
total\_pixels = 0  
grey\_level = 255  
  
def equilization(image, histogram, total\_pixels, Array):  
 #probability  
 new\_arr = [x/total\_pixels for x in histogram]  
 #commulative sum  
 for i in range(1,256):  
 new\_arr[i] = new\_arr[i]+new\_arr[i-1]  
 #multiplication with grey levels  
 new\_arr = [x \* 255 for x in new\_arr]  
 new\_arr = [round(x) for x in new\_arr]  
  
 for i in range (0, 256):  
 for (a, b) in Array[i]:  
 image[a][b] = new\_arr[image[a][b]]  
 return image  
  
def applyHistogramEquilization(image):  
 histogram = [0] \* 256  
 A = np.array([] \* 255)  
 iR = image.shape[0]  
 iC = image.shape[1]  
  
 # getting number of pixels of intensities  
 total\_pixels = 0  
 Array = []  
 for i in range(0, 256):  
 subarray = []  
 Array.append(subarray)  
  
 for i in range(0, iR):  
 for j in range(0, iC):  
 total\_pixels = total\_pixels + 1  
 histogram[image[i][j]] = histogram[image[i][j]] + 1  
 x = i  
 y = j  
 Array[image[i][j]].append((i, j))  
 image = equilization(image, histogram, total\_pixels, Array)  
 return image  
  
def pixelVal(pix, r1, s1, r2, s2):  
 if (0 <= pix and pix <= r1):  
 return (s1 / r1) \* pix  
 elif (r1 < pix and pix <= r2):  
 return ((s2 - s1) / (r2 - r1)) \* (pix - r1) + s1  
 else:  
 return ((255 - s2) / (255 - r2)) \* (pix - r2) + s2  
  
  
def contrast\_stretched\_image(image, r1, r2, s1, s2):  
 for i in range(0, image.shape[0]):  
 for j in range(0, image.shape[1]):  
 image[i][j] = pixelVal(image[i][j], r1, s1, r2, s2)  
 return image  
  
  
def returnMin(image):  
 minn = image[0][0]  
 for i in range(0, image.shape[0]):  
 for j in range(0, image.shape[1]):  
 if minn > image[i][j]:  
 minn = image[i][j]  
 return minn  
  
def returnMax(image):  
 maxx = image[0][0]  
 for i in range(0, image.shape[0]):  
 for j in range(0, image.shape[1]):  
 if maxx < image[i][j]:  
 maxx = image[i][j]  
 return maxx  
  
  
imge = cv2.imread(**'pollen.tif'**)  
imge = cv2.cvtColor(imge, cv2.COLOR\_BGR2GRAY)  
  
r1 = 100  
s1 = 50  
r2 = 200  
s2 = 230  
  
  
print(**"Contrast Stretching where (r1,s1) = (100,50) and (r2,s2) = (200,230)"**)  
img1 = imge.copy()  
img1 = contrast\_stretched\_image(img1, r1, r2, s1, s2)  
  
s1 = 0  
s2 = 255  
  
print(**"Contrast Stretching where (r1,s1) = (rmin,0) and (r2,s2) = (rmax,255)"**)  
  
r1 = returnMin(imge)  
r2 = returnMax(imge)  
img2 = imge.copy()  
img2 = contrast\_stretched\_image(img2, r1, r2, s1, s2)  
  
print(**"Applying Histogram Equilization "**)  
img3 = imge.copy()  
img3 = applyHistogramEquilization(img3)  
  
images = [imge, img1, img2, img3]  
titles = [**"Original"**, **"Contrast Stretched"**, **"Contrast Stretched(min,max)"**, **"Histogram Equilization"**]  
rows = 1  
columns = 4  
fig = plt.figure(figsize=(20, 20))  
for i in range(1, columns \* rows + 1):  
 ax = fig.add\_subplot(rows, columns, i)  
 ax.title.set\_text(titles[i-1])  
 plt.axis(**'off'**)  
 plt.imshow(images[i-1], cmap=**'gray'**, vmin=0, vmax=255)  
 plt.axis(**'off'**)  
plt.show()  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Results:**



e)

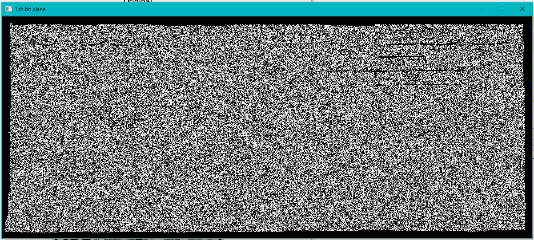
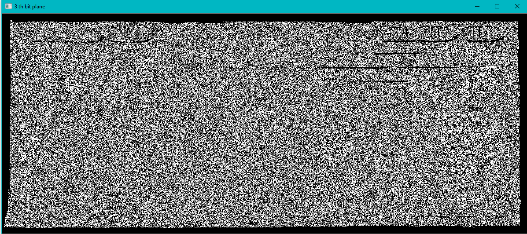
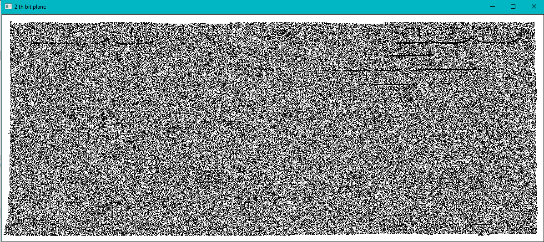
**Question 5:**

**Code:**

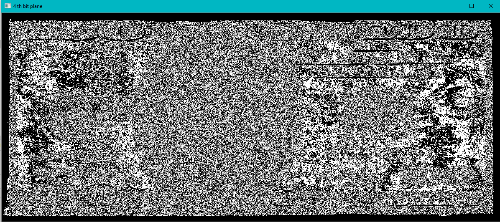
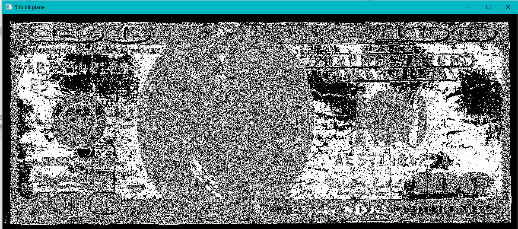
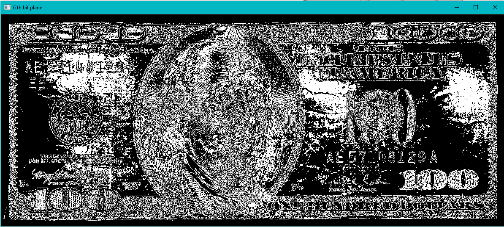
import cv2  
from skimage.exposure import rescale\_intensity  
import matplotlib.pyplot as plt  
import math  
  
  
def showBitPlanes(img, num):  
 if num != **""**:  
 num = int(num)  
  
 plane1 = img % 2  
 plane2 = (img // 2) % 2  
 plane3 = (img // 4) % 2  
 plane4 = (img // 8) % 2  
 plane5 = (img // 16) % 2  
 plane6 = (img // 32) % 2  
 plane7 = (img // 64) % 2  
 plane8 = (img // 128) % 2  
  
 #Resaling between 0 and 1 to convert the image into monocrome  
  
 plane1 = rescale\_intensity(plane1, in\_range=(0, 255))  
 plane1 = (plane1 \* 255).astype(**"uint8"**)  
 plane2 = rescale\_intensity(plane2, in\_range=(0, 255))  
 plane2 = (plane2 \* 255).astype(**"uint8"**)  
 plane3 = rescale\_intensity(plane3, in\_range=(0, 255))  
 plane3 = (plane3 \* 255).astype(**"uint8"**)  
 plane4 = rescale\_intensity(plane4, in\_range=(0, 255))  
 plane4 = (plane4 \* 255).astype(**"uint8"**)  
 plane5 = rescale\_intensity(plane5, in\_range=(0, 255))  
 plane5 = (plane5 \* 255).astype(**"uint8"**)  
 plane6 = rescale\_intensity(plane6, in\_range=(0, 255))  
 plane6 = (plane6 \* 255).astype(**"uint8"**)  
 plane7 = rescale\_intensity(plane7, in\_range=(0, 255))  
 plane7 = (plane7 \* 255).astype(**"uint8"**)  
 plane8 = rescale\_intensity(plane8, in\_range=(0, 255))  
 plane8 = (plane8 \* 255).astype(**"uint8"**)  
  
 if num == 1:  
 cv2.imshow(**"1 th bit plane"**, plane1)  
 elif num == 2:  
 cv2.imshow(**"2 th bit plane"**, plane2)  
 elif num == 3:  
 cv2.imshow(**"3 th bit plane"**, plane3)  
 elif num == 4:  
 cv2.imshow(**"4 th bit plane"**, plane4)  
 elif num == 5:  
 cv2.imshow(**"5 th bit plane"**, plane5)  
 elif num == 6:  
 cv2.imshow(**"6 th bit plane"**, plane6)  
 elif num == 7:  
 cv2.imshow(**"7 th bit plane"**, plane7)  
 elif num == 8:  
 cv2.imshow(**"8 th bit plane"**, plane8)  
 else:  
  
 fig = plt.figure(figsize=(8, 8))  
 columns = 4  
 rows = 2  
  
 plt.title(**"All planes"**)  
 plt.axis(**'off'**)  
 planes = [plane1, plane2, plane3, plane4, plane5, plane6, plane7, plane8]  
 titles = [**"first plane"**, **"second plane"**, **"third plane"**, **"fourth plane"**, **"fifth plane"**, **"sixth plane"**, **"seventh plane"**, **"eight plane"** ]  
  
 for i in range(1, columns \* rows + 1):  
 ax = fig.add\_subplot(rows, columns, i)  
 ax.title.set\_text(titles[i-1])  
 plt.axis(**'off'**)  
 plt.imshow(planes[i-1],cmap=**'gray'**)  
 plt.axis(**'off'**)  
 plt.show()  
  
  
def Resconstruct(img, num=0):  
 plane1 = img % 2  
 plane2 = (img // 2) % 2  
 plane3 = (img // 4) % 2  
 plane4 = (img // 8) % 2  
 plane5 = (img // 16) % 2  
 plane6 = (img // 32) % 2  
 plane7 = (img // 64) % 2  
 plane8 = (img // 128) % 2  
  
 if num == 4:  
 img1 = (2 \* (2 \* (2 \* plane8 + plane7) + plane6) + plane5)  
 return img1  
 elif num == 2:  
 img1 = (2 \* (2 \* (2 \* (2 \* (2 \* (2 \* (2 \* plane8 + plane7) + 0) + 0) + 0) + 0) + 0) +  
 0)  
 return img1  
 else:  
 img1 = (2 \* (2 \* (2 \* (2 \* (2 \* (2 \* (2 \* plane8 + plane7) + plane6) + plane5) + plane4) + plane3) + plane2) +  
 plane1)  
 return img1  
  
  
def MSE(img1, img2):  
 summ = 0  
 for i in range (1, img1.shape[0]):  
 for j in range(1, img1.shape[1]):  
 dif = img1[i, j].astype(**'float'**)-img2[i, j].astype(**'float'**)  
 summ = summ+(dif\*\*2)  
  
 mse = summ/float(img1.shape[0]\*img2.shape[1])  
 return mse  
  
def PSNR(mse):  
 mul = float((255\*\*2)/mse)  
 psnr = float(math.log10(mul))  
 return psnr  
  
img = cv2.imread(**"dollar.tif"**)  
img = cv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)  
x = input(**"Enter the number of a specific bit plane to display or Press Enter to Show all: "**)  
showBitPlanes(img,x)  
img1 = Resconstruct(img, 4)  
img2 = Resconstruct(img, 2)  
cv2.imwrite(**"reconstructedDollar2.tif"**, img2)  
cv2.imwrite(**"reconstructedDollar4.tif"**, img1)  
  
img1 = cv2.imread(**"reconstructedDollar4.tif"**)  
img1 = cv2.cvtColor(img1,cv2.COLOR\_RGB2GRAY)  
  
img2 = cv2.imread(**"reconstructedDollar2.tif"**)  
img2 = cv2.cvtColor(img2,cv2.COLOR\_RGB2GRAY)  
rows = 1  
cols = 3  
  
images = [img, img1, img2]  
fig = plt.figure(figsize=(8, 12))  
titles = [**"Original"**, **"Reconstructed4Dollar"**, **"Reconstructed2Dollar"**]  
plt.axis(**'off'**)  
rows = 3  
cols = 1  
for i in range(1, rows \* cols + 1):  
  
 ax = fig.add\_subplot(rows, cols, i)  
 ax.title.set\_text(titles[i - 1])  
 plt.axis(**'off'**)  
 plt.imshow(images[i - 1], cmap= **'gray'**)  
  
plt.axis(**'off'**)  
plt.show()  
  
mse1 = MSE(img, img1)  
mse2 = MSE(img, img2)  
psnr1 = PSNR(mse1)  
psnr2 = PSNR(mse2)  
print(**"Mean Square error of Original Image with reconstructed4Dollar "**)  
print(mse1)  
print(**"Signal to Noise Ratio of Original Image with reconstructed4Dollar "**)  
print(psnr1)  
print(**"Mean Square error of Original Image with reconstructed2Dollar"**)  
print(mse2)  
print(**"Mean Square error of Original Image with reconstructed2Dollar"**)  
print(psnr2)  
cv2.waitKey(0)

**Results:**

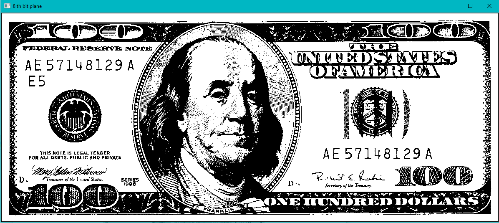
a)

 First Bit Plane Second Bit Plane Third Bit Plane

Fourth Bit Plane Fifth Bit Plane Sixth Bit Plane



Seventh Bit Plane Eight Bit Plane



b,c)



d)

Mean Square error of Original Image with reconstructed4Dollar

24458.501479865772

Signal to Noise Ratio of Original Image with reconstructed4Dollar

0.4246505156476699

Mean Square error of Original Image with reconstructed2Dollar

938.0741476510067

Mean Square error of Original Image with reconstructed2Dollar

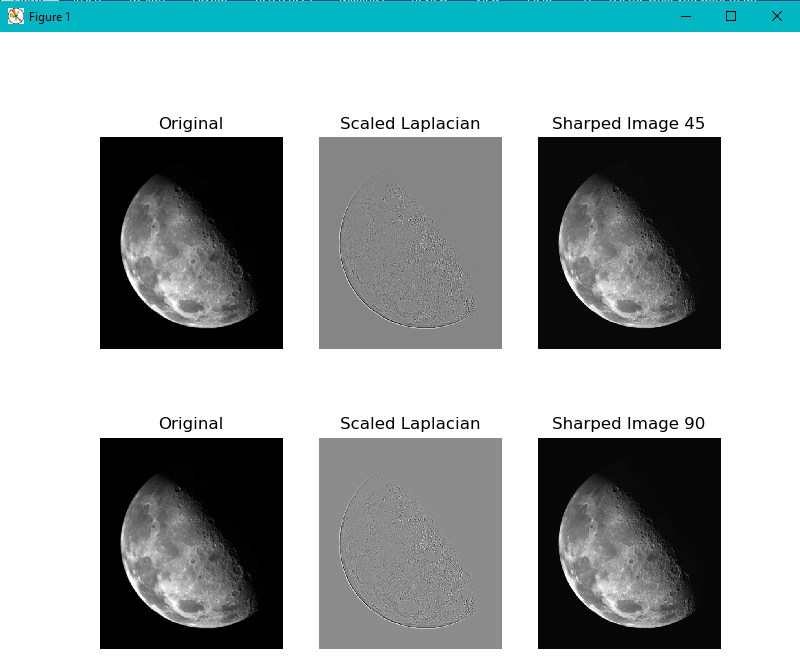
1.8408431934451508

**Question 6**

**Code:**

import numpy as np,array  
import cv2  
from PIL import Image  
from skimage.exposure import rescale\_intensity  
import matplotlib.pyplot as plt  
  
def pixelVal(pix, r1, s1, r2, s2):  
 if (0 <= pix and pix <= r1):  
 return (s1 / r1) \* pix  
 elif (r1 < pix and pix <= r2):  
 return ((s2 - s1) / (r2 - r1)) \* (pix - r1) + s1  
 else:  
 return ((255 - s2) / (255 - r2)) \* (pix - r2) + s2  
  
  
def contrast\_stretched\_image(img, r1, r2, s1, s2):  
 for i in range(0, img.shape[0]):  
 for j in range(0, img.shape[1]):  
 type(img[i][j])  
 img[i][j] = pixelVal(img[i][j], r1, s1, r2, s2)  
 return img  
  
  
def returnMin(img):  
 minn = img[0][0]  
 for i in range(0, img.shape[0]):  
 for j in range(0, img.shape[1]):  
 if minn > img[i][j]:  
 minn = img[i][j]  
 return minn  
  
def returnMax(img):  
 maxx = img[0][0]  
 for i in range(0, img.shape[0]):  
 for j in range(0, img.shape[1]):  
 if maxx < img[i][j]:  
 maxx = img[i][j]  
 return maxx  
  
def scale(image):  
 rmin = returnMin(image)  
 rmax = returnMax(image)  
 image = contrast\_stretched\_image(image, r1=rmin, r2=rmax, s1=0, s2=255)  
 return image  
  
def convolve(image, kernel):  
 kernel = np.flipud(np.fliplr(kernel)) # Flip the kernel  
 iR = image.shape[0]  
 iC = image.shape[1]  
 kR = kernel.shape[0]  
 kC = kernel.shape[1]  
 # no of pad rows  
 pad = (kC - 1) // 2  
  
 image = np.pad(image, pad, mode=**'constant'**, constant\_values=0)  
 output = np.zeros((iR, iC), dtype=**"float32"**)  
 iR1 = image.shape[0]  
 iC1 = image.shape[1]  
  
 for x in np.arange(pad, iR1 - pad):  
 for y in np.arange(pad, iC1 - pad):  
 k = 0  
 for i in range(0, kR):  
 for j in range(0, kC):  
 k = k + (kernel[i][j] \* image[x - pad + i, y - pad + j])  
 output[x - pad][y - pad] = k  
 return output  
  
image = cv2.imread(**"moon.tif"**)  
image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
laplacian90 = np.array((  
 [0, 1, 0],  
 [1, -4, 1],  
 [0, 1, 0]), dtype=**"float"**)  
  
laplacian45 = np.array((  
 [1, 1, 1],  
 [1, -8, 1],  
 [1, 1, 1]), dtype=**"float"**)  
  
  
img\_45 = convolve(image, laplacian45)  
  
img\_90 = convolve(image, laplacian90)  
  
img\_45s = scale(img\_45)  
img\_90s = scale(img\_90)  
  
img\_45 = image-0.2\*img\_45s  
img\_90 = image-0.2\*img\_90s  
  
images = [image, img\_45s, img\_45, image, img\_90s, img\_90]  
titles = [**"Original"**, **"Scaled Laplacian"**, **"Sharped Image 45"**,**"Original"**, **"Scaled Laplacian"**, **"Sharped Image 90"**]  
  
fig = plt.figure(figsize=(8, 8))  
columns = 3  
rows = 2  
  
plt.axis(**'off'**)  
  
for i in range(1, columns \* rows + 1):  
 ax = fig.add\_subplot(rows, columns, i)  
 ax.title.set\_text(titles[i-1])  
 plt.axis(**'off'**)  
 plt.imshow(images[i-1], cmap=**'gray'**)  
 plt.axis(**'off'**)  
plt.show()  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Results:**

****