**Image Processing Lab**

**ASSIGNMENT - 1**

1. Write a code to create an athletic track of size given d.

**Solution**:

import cv2

import numpy as np

import matplotlib.pyplot as plt

height = 250

length = 300

mat = np.zeros((height, length))

x = 50

d = 30

for i in range(x, x+d):

for j in range(x, length-x):

mat[i][j] = 1

for i in range(x+d, height-d-x):

for j in range(x, x+d):

mat[i][j] = 1

for j in range(length-x-d, length-x):

mat[i][j] = 1

for i in range(height-x-d, height-x):

for j in range(x, length-x):

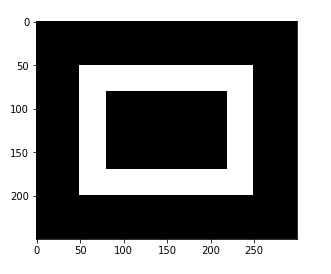
mat[i][j] = 1

img = mat

plt.imshow(img, cmap='gray')

plt.show()

**Output:**



1. Write code to create an athletic track of size given radius r and track width d.

**Solution:**

import cv2

import numpy as np

import math

import matplotlib.pyplot as plt

height = 250

length = 300

mat = np.zeros((height, length))

r = 75

d = 20

mid\_x = height/2

mid\_y = length/2

for i in range(r, r+d):

for angle in np.arange(0, 360, 0.01):

x = mid\_x+i\*math.cos(angle)

y = mid\_y+i\*math.sin(angle)

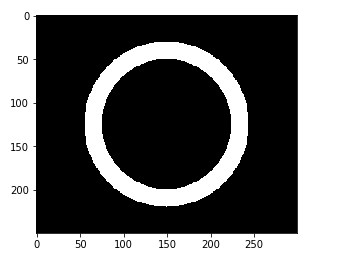
mat[int(x)][int(y)] = 1

img = mat

plt.imshow(img, cmap='gray')

plt.show()

**Output:**

****

1. Display image for the given 2D function

**Solution:**

import math

import matplotlib.pyplot as plt

import numpy as np

height = 512

width = 512

img = np.zeros((height,width),dtype=np.uint8)

for i in range(0, height):

for j in range(0, width):

intensity = 255 \* math.cos(2\*3.1416\*(float(i)/50 + float(j)/25))

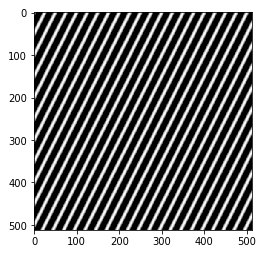
if(int(intensity)>=0):

img[i, j] = int(intensity)

plt.imshow(img, cmap='gray')

plt.show()

**Output:**



1. Perform Zoom-in and Zoom-out operation using nearest neighborhood for `cameraman.tif' test image of size 256 \* 256.

**Solution:**

import math

import matplotlib.pyplot as plt

import numpy as np

def main():

I mg = plt.imread('cameraman.tif')

arr = np.asarray(img, dtype=np.uint8)

row, col = arr.shape

k = int(input('Enter zooming factor : '))

img1 = np.zeros((int(row/k), int(col/k)), dtype=np.uint8)

img2 = np.zeros((row \* k, col \* k), dtype=np.uint8)

for i in range(len(img1)):

for j in range(len(img1[0])):

img1[i,j] = img[i\*k,j\*k];

for i in range(len(img2)):

for j in range(len(img2[0])):

img2[i,j] = img[i/k,j/k];

plt.imshow(img1, cmap="gray")

plt.show()

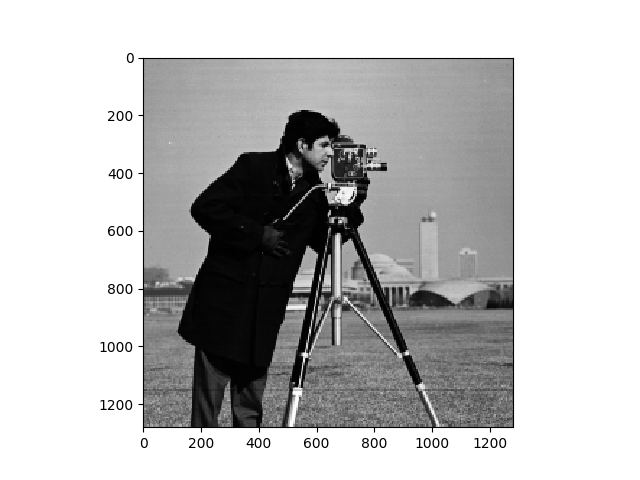
plt.imshow(img2, cmap="gray")

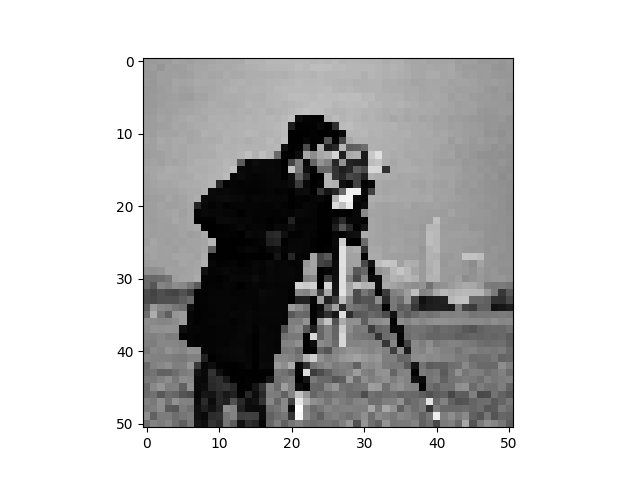
plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

Zoom-in factor 5Zoom-out factor 5

****

**ASSIGNMENT – 2**

1. Bit plane slicing transformation on a given image (`fractal.png').

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def bitSlicing(img,pos):

ht = img.shape[0]

wd = img.shape[1]

img\_bit = np.zeros((ht,wd), np.uint8)

for y in range(ht):

for x in range(wd):

actual\_val = np.binary\_repr(img[y][x],width=8)

if actual\_val[pos] == '1':

img\_bit[y][x] = 255

else:

img\_bit[y][x] = 0

return img\_bit

image = cv2.imread("fractal.png",0)

for i in range(8):

sliced\_img = bitSlicing(image,i)

if(i%2==0):

plt.subplot(121)

plt.imshow(sliced\_img, cmap='gray')

plt.title("Bit-plane"+str(8-i))

else:

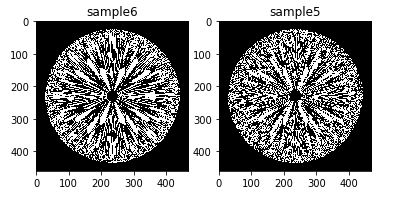
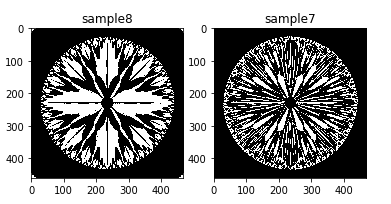
plt.subplot(122)

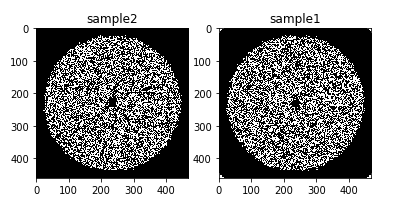
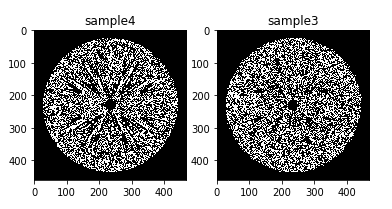
plt.imshow(sliced\_img, cmap='gray')

plt.title("Bit-plane"+str(8-i))

plt.show()

**Output:**





1. Histogram without using any in-built function for a given image (`grain.png').

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread("grain.png",0)

frq = np.zeros((256))

for y in range(image.shape[0]):

for x in range(image.shape[1]):

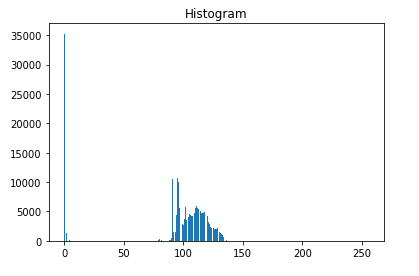
frq[image[y][x]] = frq[image[y][x]] + 1

plt.bar(range(256), frq)

plt.title("Histogram")

plt.show()

**Output:**

****

1. constant stretching to `grain.png' by using the mapping function.

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread("grain.png",0)

def bitStretching(img):

ht = img.shape[0]

wd = img.shape[1]

img\_bit = np.zeros((ht,wd), np.float64)

for y in range(ht):

for x in range(wd):

if(img[y][x]>=0 and img[y][x]<80):

img\_bit[y][x] = img[y][x] \* 0.5

elif(img[y][x]>=80 and img[y][x]<160):

img\_bit[y][x] = img[y][x] \* 2-120

elif(img[y][x]>=160 and img[y][x]<200):

img\_bit[y][x] = img[y][x] \* 0.57+107.52

return img\_bit

new\_img = bitStretching(image)

plt.imshow(new\_img,cmap='gray')

plt.title("Image after stretching")

plt.show()

frq = np.zeros((256))

for y in range(new\_img.shape[0]):

for x in range(new\_img.shape[1]):

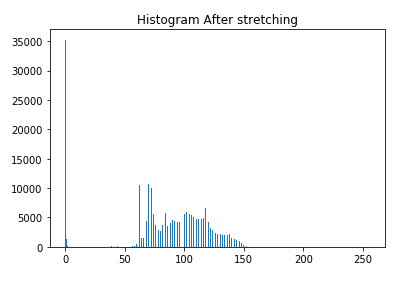
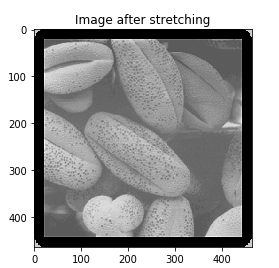
frq[int(new\_img[y][x])] = frq[int(new\_img[y][x])] + 1

plt.bar(range(256), frq)

plt.title("Histogram After stretching")

plt.show()

**Output:**



**ASSIGNMENT - 3**

1. Perform histogram equalization and compared output with the output

obtained using 'histeq' inbuilt function.

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

image\_or = cv2.imread("tire.tif",0)

image\_eq = cv2.imread("tire.tif",0)

image\_ib = cv2.imread("tire.tif",0)

frq\_or = np.zeros((256))

frq\_eq = np.zeros((256))

frq\_ib = np.zeros((256))

mapping = np.zeros((256))

cumulative = np.zeros((256))

for y in range(image\_or.shape[0]):

for x in range(image\_or.shape[1]):

frq\_or[image\_or[y][x]] = frq\_or[image\_or[y][x]] + 1

cumulative[0] = frq\_or[0]

for i in range(1,256):

cumulative[i] = cumulative[i-1]+frq\_or[i]

cum = cumulative[255]

for i in range(256):

cumulative[i] = cumulative[i]/cum

for i in range(256):

mapping[i] = math.floor(cumulative[i]\*255)

for y in range(image\_eq.shape[0]):

for x in range(image\_eq.shape[1]):

image\_eq[y][x] = mapping[image\_or[y][x]]

image\_ib = cv2.equalizeHist(image\_or)

for y in range(image\_ib.shape[0]):

for x in range(image\_ib.shape[1]):

frq\_ib[image\_ib[y][x]] = frq\_ib[image\_ib[y][x]] + 1

for y in range(image\_eq.shape[0]):

for x in range(image\_eq.shape[1]):

frq\_eq[image\_eq[y][x]] = frq\_eq[image\_eq[y][x]] + 1

plt.figure(1)

plt.bar(range(256), frq\_eq)

plt.title("Histogram Equalization")

plt.show()

plt.bar(range(256), frq\_eq)

plt.title("Using inbuilt funciton")

plt.show()

plt.imshow(image\_or, cmap='gray')

plt.title("Original Image")

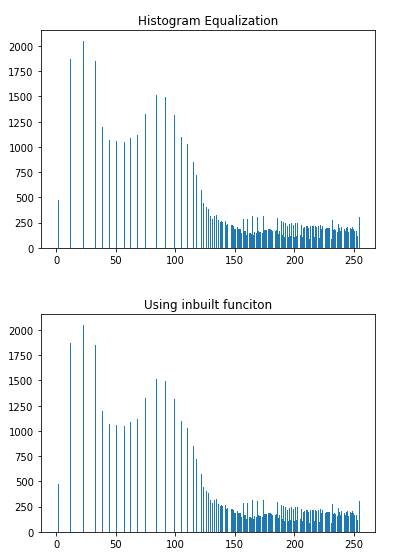
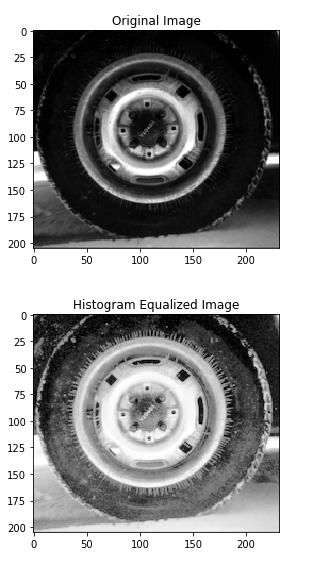
plt.show()

plt.imshow(image\_eq, cmap='gray')

plt.title("Histogram Equalized Image")

plt.show()

**Output:**

****

1. perform histogram specification for a given input image and desired/specified image. (take input as tire.tif and desired as cameraman.tif)

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image\_or = cv2.imread("tire.tif",0)

image\_sp = cv2.imread("tire.tif",0)

image\_tr = cv2.imread("cameraman.tif",0)

frq\_or = np.zeros((256))

frq\_sp = np.zeros((256))

frq\_tr = np.zeros((256))

cumulative\_or = np.zeros((256))

cumulative\_tr = np.zeros((256))

#cumulative for input

for y in range(image\_or.shape[0]):

for x in range(image\_or.shape[1]):

frq\_or[image\_or[y][x]] = frq\_or[image\_or[y][x]] + 1

cumulative\_or[0] = frq\_or[0]

for i in range(1,256):

cumulative\_or[i] = cumulative\_or[i-1]+frq\_or[i]

cum = cumulative\_or[255]

for i in range(256):

cumulative\_or[i] = cumulative\_or[i]/cum

#########################

#cumulative for target

for y in range(image\_tr.shape[0]):

for x in range(image\_tr.shape[1]):

frq\_tr[image\_tr[y][x]] = frq\_tr[image\_tr[y][x]] + 1

cumulative\_tr[0] = frq\_tr[0]

for i in range(1,256):

cumulative\_tr[i] = cumulative\_tr[i-1]+frq\_tr[i]

cum = cumulative\_tr[255]

for i in range(256):

cumulative\_tr[i] = cumulative\_tr[i]/cum

##########################

#matching

j=0

index\_map = np.zeros((256))

for i in range(256):

this\_val = cumulative\_or[i]

min\_diff = abs(cumulative\_or[i]-cumulative\_tr[j])

for k in range(j,256):

new\_diff = abs(cumulative\_or[i]-cumulative\_tr[j])

if new\_diff>min\_diff:

j = j-1

break

elif new\_diff<=min\_diff and j<255:

j = j+1

index\_map[i] = j

#########################

for y in range(image\_sp.shape[0]):

for x in range(image\_sp.shape[1]):

image\_sp[y][x] = index\_map[image\_or[y][x]]

for y in range(image\_sp.shape[0]):

for x in range(image\_sp.shape[1]):

frq\_sp[image\_sp[y][x]] = frq\_sp[image\_sp[y][x]] + 1

plt.figure(1)

plt.subplot(311)

plt.bar(range(256), frq\_or)

plt.subplot(312)

plt.bar(range(256), frq\_tr)

plt.subplot(313)

plt.bar(range(256), frq\_sp)

plt.show()

plt.imshow(image\_or, cmap='gray')

plt.title("Original Image")

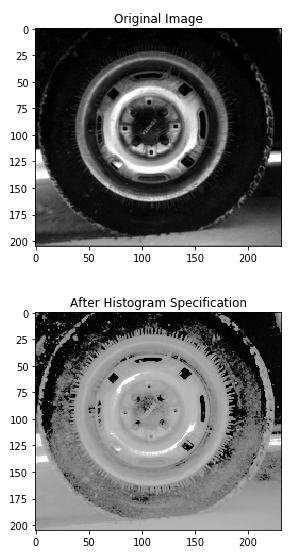
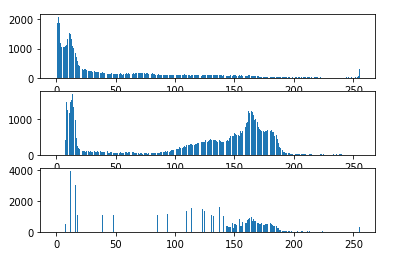
plt.show()

plt.imshow(image\_sp,cmap='gray')

plt.title("After Histogram Specification")

plt.show()

**Output:**



**ASSIGNMENT - 4**

1. Write a program to perform negative, logarithm and power law transformation on `pout.png' and `tire.png' test images.

**Solution -**

1. **Negative Transform**:

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

import sys

image\_or = cv2.imread(‘pout.tif’,0) #change to tire.tif for another input

image\_ng = cv2.imread(‘pout.tif’,0) #change to tire.tif for another input

for y in range(image\_or.shape[0]):

for x in range(image\_or.shape[1]):

image\_ng[y][x] = 255-image\_or[y][x]

plt.subplot(121)

plt.imshow(image\_or, cmap='gray')

plt.title("Original Image")

plt.subplot(122)

plt.imshow(image\_ng, cmap='gray')

plt.title("Negative Tranformed Image")

plt.show()

1. **Logarithmic Transform:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

import sys

image\_or = cv2.imread('tire.tif',0)

image\_ng = cv2.imread('tire.tif',0)

c=100

for y in range(image\_or.shape[0]):

for x in range(image\_or.shape[1]):

image\_ng[y][x] = c\*math.log10(1+image\_or[y][x])

plt.subplot(121)

plt.imshow(image\_or, cmap='gray')

plt.title("Original Image")

plt.subplot(122)

plt.imshow(image\_ng, cmap='gray')

plt.title("Log Tranformed Image")

plt.show()

1. **PowerLaw Transform**:

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

import sys

image\_or = cv2.imread('pout.tif',0)

image\_ng = cv2.imread('pout.tif',0)

c=1

gamma=0.9

for y in range(image\_or.shape[0]):

for x in range(image\_or.shape[1]):

image\_ng[y][x] = c\*math.pow(image\_or[y][x],gamma)

plt.subplot(121)

plt.imshow(image\_or, cmap='gray')

plt.title("Original Image")

plt.subplot(122)

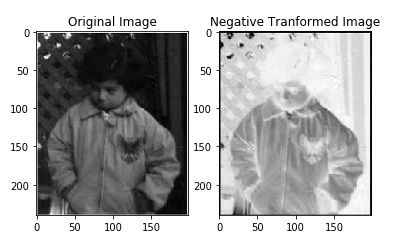
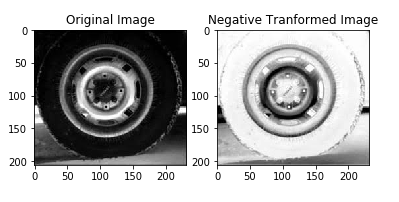
plt.imshow(image\_ng, cmap='gray')

plt.title("Power Tranformed Image")

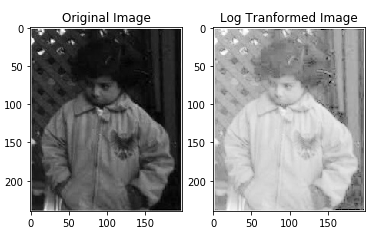
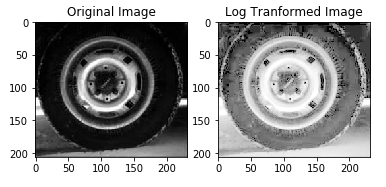
plt.show()

**Output**-

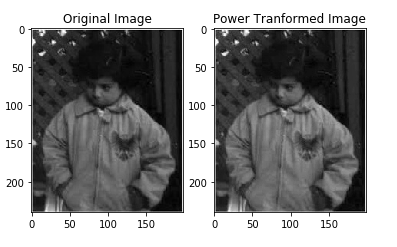
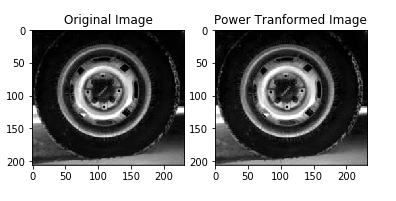
(a)



(b)



(c)



1. Write a program to perform 2D convolution using graphical method and matrix method.

**Solution:**

import numpy as np

import cv2

import matplotlib.pyplot as plt

img = cv2.imread('tire.tif',0)

def convolve2d(image, kernel):

kernel = np.flipud(np.fliplr(kernel))

output = np.zeros\_like(image)

image\_padded = np.zeros((image.shape[0] + 2, image.shape[1] + 2))

image\_padded[1:-1, 1:-1] = image

for x in range(image.shape[1]):

for y in range(image.shape[0]):

output[y,x]=(kernel\*image\_padded[y:y+3,x:x+3]).sum()

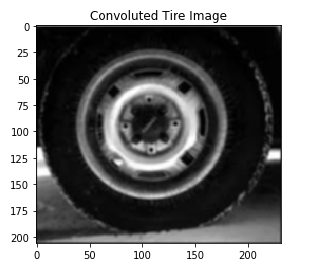
return output

plt.imshow(convolve2d(img,np.array([[1,1,1],[1,1,1],[1,1,1]],np.float64)/9),cmap='gray')

plt.title("Convoluted Tire Image")

plt.show()

**Output:**

****

1. perform image averaging and weighted averaging operation for different masks.

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import sys

image\_or = cv2.imread('eight.tif',0)

def convolution\_2d\_corner(image,kernel): #corner convolution

r,c = kernel.shape

divisor = np.sum(kernel)

y,x = image.shape

y = y-r+1

x = x-c+1

output = np.zeros((y,x))

if(r==c):

for i in range(y):

for j in range(x):

output[i][j] = np.sum(np.multiply(image[i:i+r,j:j+c],kernel))/divisor

else:

print("Kernel should be square matrix")

return output

def convolution\_2d\_centre(image,kernel): #centre convolution

r,c = kernel.shape

divisor = np.sum(kernel)

y,x = image.shape

startrow = int(r/2)

startcol = int(c/2)

rows = y-startrow+1

cols = x-startcol+1

output = np.zeros((y,x))

if(r==c):

for i in range(startrow,rows):

for j in range(startcol,cols):

output[i][j] = np.sum(np.multiply(image[i-startrow:i+startrow,j-startcol:j+startcol],kernel))/divisor

else:

print("Kernel should be square matrix")

return output

kr1 = np.array([[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]])

kr2 = np.array([[1,2,2,1],[1,2,2,1],[1,2,2,1],[1,2,2,1]])

image\_cv = convolution\_2d\_centre(image\_or,kr2)

plt.subplot(121)

plt.imshow(image\_or, cmap='gray')

plt.title("Original Image")

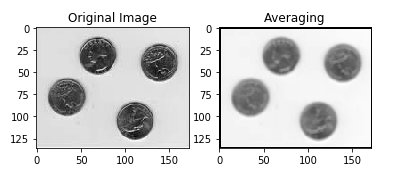
plt.subplot(122)

plt.imshow(image\_cv.astype('uint8'), cmap='gray')

plt.title("Averaging")

plt.show()

**Output:**



**ASSIGNMENT – 5**

1. Plot the real and imaginary component of the basis images separately in case of Discrete Fourier Transform (DFT) for a specified size (e.g., 8 \* 8, 16 \*16)

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

import sys

size = 16

def DFT(k,n,N,arg):

if arg==0:

return np.cos((2\*np.pi\*k\*n)/(float(N)))

else:

return -np.sin((2\*np.pi\*k\*n)/(float(N)))

def getBasisDFT(a,b):

b\_image=[]

for i in range(len(a)):

temp=[]

for j in range(len(b)):

temp.append([a[i][0]\*b[j][0]-a[i][1]\*b[j][1],a[i][0]\*b[j][1]+a[i][1]\*b[j][0]])

b\_image.append(temp)

return b\_image

def basisImage(N,arg):

A\_list=[]

for i in range(N):

Ak=[]

for j in range(N):

Ak.append([DFT(j,i,N,0),DFT(j,i,N,1)])

A\_list.append(Ak)

basis=[]

for i in range(len(A\_list)):

temp=[]

for j in range(len(A\_list)):

temp.append(getBasisDFT(A\_list[i],A\_list[j]))

basis.append(temp)

basis=np.asarray(basis)

return basis

result=basisImage(size,0)

print(result.shape)

new\_resr=[]

for i in range(size):

temp1=[]

for j in range(size):

temp2=[]

for k in range(size):

temp3=[]

for l in range(size):

temp3.append(result[i][j][k][l][0]\*255)

temp2.append(temp3)

temp1.append(temp2)

new\_resr.append(temp1)

new\_resi=[]

for i in range(size):

temp1=[]

for j in range(size):

temp2=[]

for k in range(size):

temp3=[]

for l in range(size):

temp3.append(result[i][j][k][l][1]\*255)

temp2.append(temp3)

temp1.append(temp2)

new\_resi.append(temp1)

new\_resr=np.asarray(new\_resr)

new\_resi=np.asarray(new\_resi)

new\_resr[0][0][0][0]=0

fig=plt.figure(figsize=(size,size))

k=1

for i in range(size):

for j in range(size):

fig.add\_subplot(size,size,k)

plt.imshow(new\_resr[i][j],cmap='gray')

k+=1

plt.show()

fig=plt.figure(figsize=(size,size))

k=1

for i in range(size):

for j in range(size):

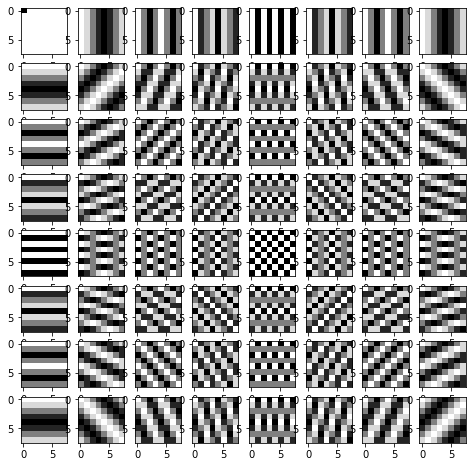
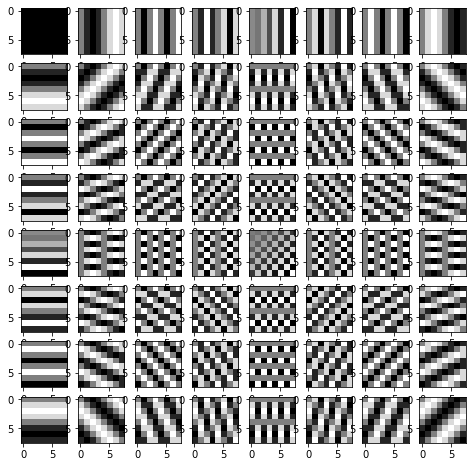
fig.add\_subplot(size,size,k)

plt.imshow(new\_resi[i][j],cmap='gray')

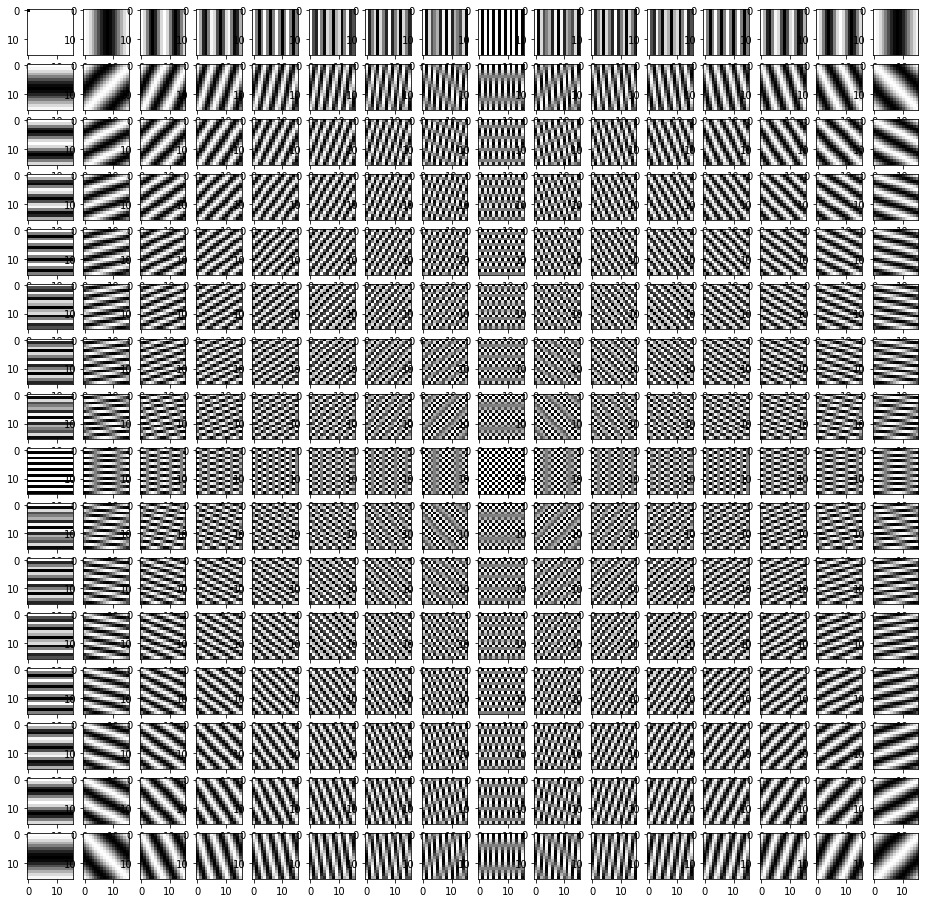
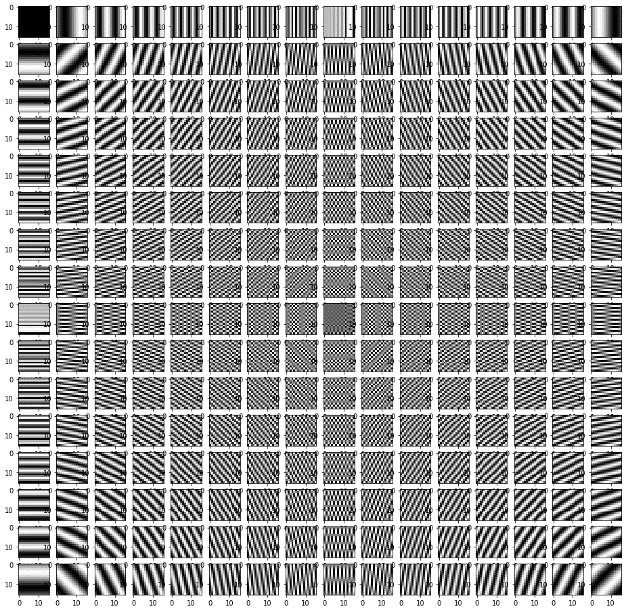
k+=1

plt.show()

**Output:**

** **

**(Real Part) [8x8] (Imaginary part)**

**[16x16]**

1. Plot the real and imaginary component of the basis images separately in case of Discrete cosine Transform (DCT) for a specified size (e.g., 8 \* 8, 16 \*16)

**Solution:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

import sys

size = 8

def DCT(k,n,N):

dct = -1

if n==0:

dct = np.sqrt(1/float(N))\*np.cos((np.pi\*(2\*k+1)\*n)/(2\*float(N)))

else:

dct = np.sqrt(2/float(N))\*np.cos((np.pi\*(2\*k+1)\*n)/(2\*float(N)))

return dct

def getBasisDCT(a,b):

b\_image=[]

for i in range(len(a)):

temp=[]

for j in range(len(b)):

temp.append(a[i]\*b[j])

b\_image.append(temp)

return b\_image

def basisImage(N):

A\_list=[]

for i in range(N):

Ak=[]

for j in range(N):

Ak.append(DCT(j,i,N))

A\_list.append(Ak)

basis=[]

for i in range(len(A\_list)):

temp=[]

for j in range(len(A\_list)):

temp.append(getBasisDCT(A\_list[i],A\_list[j]))

basis.append(temp)

basis=np.asarray(basis)

return basis

result=basisImage(size)

result[0][0][0][0]=0

fig=plt.figure(figsize=(size,size))

k=1

for i in range(size):

for j in range(size):

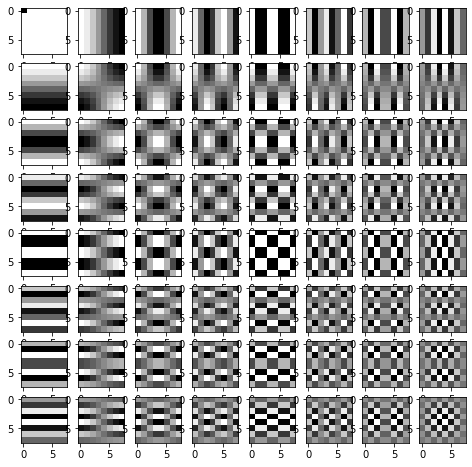
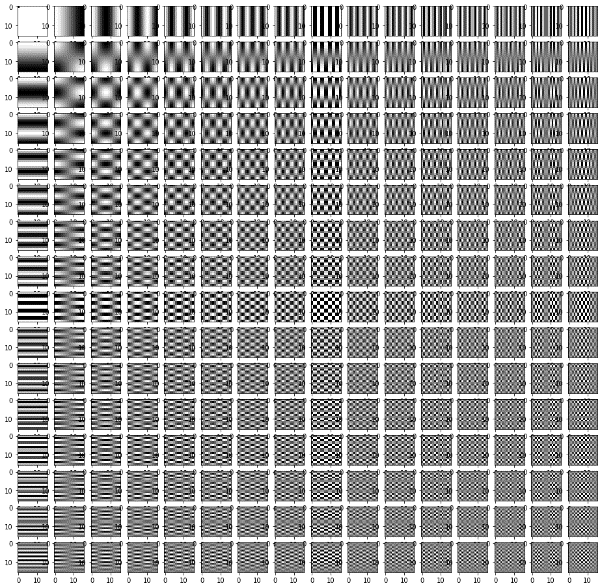
fig.add\_subplot(size,size,k)

plt.imshow(result[i][j],cmap='gray')

k+=1

plt.show()

**Output:**

 ****

(n=8) (n=16)

1. Perform image reconstruction using the high energy coefficients of DCT.

**Solution:**

import numpy as np

import cv2

import math

import matplotlib.pyplot as plt

from skimage.measure import compare\_psnr as psnr

def DCT(k,n,N):

dct = -1

if n==0:

dct = np.sqrt(1/float(N))\*np.cos((np.pi\*(2\*k+1)\*n)/(2\*float(N)))

else:

dct = np.sqrt(2/float(N))\*np.cos((np.pi\*(2\*k+1)\*n)/(2\*float(N)))

return dct

image = cv2.imread('cameraman.tif')

m=image.shape[0]

n=image.shape[1]

greyImage = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

#generate A

complexMatrix = np.zeros((m,n),dtype= complex)

for i in range(0,m):

for j in range(0,n):

if j==0:

complexMatrix[i][j] = np.sqrt(1/float(m))\*np.cos((np.pi\*(2\*i+1)\*j)/(2\*float(m)))

else:

complexMatrix[i][j] = np.sqrt(2/float(n))\*np.cos((np.pi\*(2\*i+1)\*j)/(2\*float(n)))

#get the degraded image

conjMatrix = np.conj(complexMatrix)

dctImage = np.dot(np.dot((complexMatrix), greyImage ) , (np.transpose(conjMatrix)))

new\_dctImage = np.zeros((m,n),dtype= complex)

new\_dctImage1 = np.zeros((m,n),dtype= complex)

k=100 #DCT Coefficients

k1 =20

new\_dctImage[0:k][0:k] = dctImage[0:k][0:k]

new\_dctImage1[0:k1][0:k1] = dctImage[0:k1][0:k1]

idctImage = np.dot(np.dot((np.transpose(conjMatrix)), new\_dctImage) , (complexMatrix))

idctImage1 = np.dot(np.dot((np.transpose(conjMatrix)), new\_dctImage1) , (complexMatrix))

plt.subplot(121)

plt.imshow(np.real(idctImage),cmap='gray')

plt.title("Restored image with K=100")

print(image.shape,idctImage.shape)

print('PSNR of idct image for 100 coefficients: ',psnr(greyImage,idctImage))

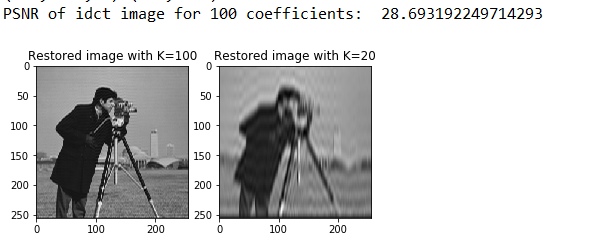
plt.subplot(122)

plt.imshow(np.real(idctImage1),cmap='gray')

plt.title("Restored image with K=20")

plt.show()

**Output:**

****

1. Display the Fourier spectrum for a given image. Perform image reconstruction and find the error.

**Solution:**

import numpy

import cv2

import math

import matplotlib.pyplot as plt

from skimage.measure import compare\_psnr as psnr

image = cv2.imread('cameraman.tif')

m=image.shape[0]

n=image.shape[1]

greyImage = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

#generate A

complexMatrix = numpy.zeros((m,n),dtype= complex)

for i in range(0,m):

for j in range(0,n):

power = i\*j

real = math.cos((2\*math.pi\*power)/m)

img = -math.sin((2\*math.pi\*power)/n)

complexMatrix[i][j] = complex(real,img)

#get the degraded image

conjMatrix = numpy.conj(complexMatrix)

dftImage = numpy.dot(numpy.dot((complexMatrix), greyImage ) , (numpy.transpose(conjMatrix)))

idftImage = numpy.dot(numpy.dot((numpy.transpose(conjMatrix)), dftImage) , (complexMatrix))

plt.imshow(numpy.real(idftImage),cmap='gray')

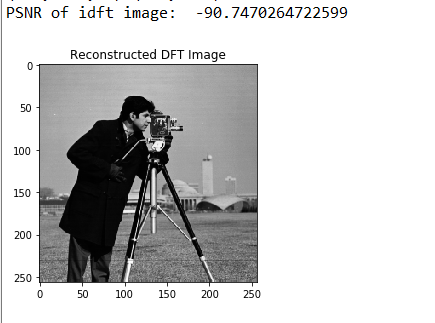
plt.title("Reconstructed DFT Image")

print(image.shape,idftImage.shape)

print('PSNR of idft image: ',psnr(greyImage,idftImage))

plt.show()

**Output:**

****

**ASSIGNMENT- 6**

1. Perform Min, Max, Average and Median Filter after applying salt and pepper noise on an Image.

**Solution:**

import cv2

import math

import numpy as np

from skimage.util import random\_noise

from skimage.measure import compare\_psnr as psnr

import matplotlib.pyplot as plt

img = cv2.imread("cameraman.tif",0)

noise\_img = random\_noise(img, mode='s&p',amount=0.2)

noise\_img = np.array(255\*noise\_img, dtype = 'uint8')

def median\_filter(image,kernel): #centre convolution

r = kernel[0]

c = kernel[1]

y,x = image.shape

startrow = int(r/2)

startcol = int(c/2)

rows = y-startrow+1

cols = x-startcol+1

output = np.zeros((y,x))

if(r==c):

for i in range(startrow,rows):

for j in range(startcol,cols):

#output[i][j] = np.median(image[i-startrow:i+startrow+1,j-startcol:j+startcol+1])

#output[i][j] = np.mean(image[i-startrow:i+startrow+1,j-startcol:j+startcol+1])

#output[i][j] = np.max(image[i-startrow:i+startrow+1,j-startcol:j+startcol+1])

output[i][j] = np.min(image[i-startrow:i+startrow+1,j-startcol:j+startcol+1])

#Change value for the suitable filter

else:

print("Kernel should be square matrix")

output = np.array(output, dtype = 'uint8')

return output

good\_img = median\_filter(noise\_img,[3,3])

print('PSNR of Noisy image: ',psnr(img,noise\_img))

print('PSNR of Filtered image: ',psnr(img,good\_img))

plt.subplot(121)

plt.imshow(noise\_img, cmap='gray')

plt.title('noise image')

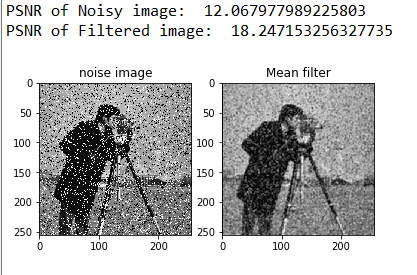
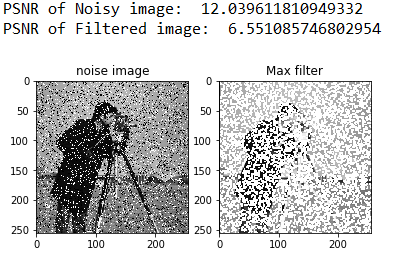
plt.subplot(122)

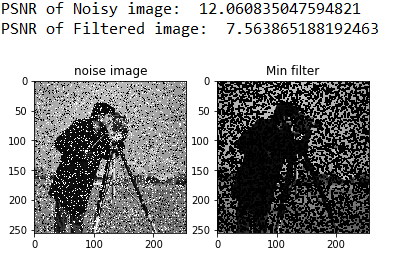
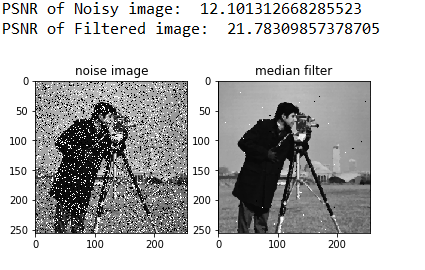
plt.imshow(good\_img,cmap='gray')

plt.title('Min filter') #Change value for the suitable filter

plt.show()

**Output:**





1. Adaptive Median Filtering

**Solution:**

import cv2

import math

import numpy as np

from skimage.util import random\_noise

from skimage.measure import compare\_psnr as psnr

import matplotlib.pyplot as plt

img = cv2.imread("cameraman.tif",0)

noise\_img = random\_noise(img, mode='s&p',amount=0.2)

noise\_img = np.array(255\*noise\_img, dtype = 'uint8')

def median\_filter(image,kernel): #centre convolution

r = kernel[0]

c = kernel[1]

y,x = image.shape

startrow = int(r/2)

startcol = int(c/2)

rows = y-startrow+1

cols = x-startcol+1

output = np.zeros((y,x))

if(r==c):

for i in range(startrow,rows):

for j in range(startcol,cols):

ks=1

while(ks<=int(r/2)):

mat = image[i-ks:i+ks+1,j-ks:j+ks+1]

med = np.median(mat)

A1 = med-np.min(mat)

A2 = med-np.max(mat)

if A1>0 and A2<0:

B1 = image[i][j]-np.min(mat)

B2 = image[i][j]-np.max(mat)

if B1>0 and B2<0:

output[i][j] = image[i][j]

else:

output[i][j] = med

break

ks+=1

if ks==int(r/2)+1:

output[i][j]=image[i][j]

else:

print("Kernel should be square matrix")

output = np.array(output, dtype = 'uint8')

return output

good\_img = median\_filter(noise\_img,[5,5])

# mse\_ni=0

# mse\_gi=0

# for y in range(img.shape[0]):

# for x in range(img.shape[1]):

# mse\_ni += (float(img[y][x])-float(noise\_img[y][x]))\*\*2

# mse\_gi += (float(img[y][x])-float(good\_img[y][x]))\*\*2

# mse\_ni /= (img.shape[0]\*img.shape[1])

# mse\_gi /= (img.shape[0]\*img.shape[1])

# print('PSNR of Noisy image: ',10\*math.log10((255\*\*2)/mse\_ni))

# print('PSNR of Filtered image: ',10\*math.log10((255\*\*2)/mse\_gi))

print('PSNR of Noisy image: ',psnr(img,noise\_img))

print('PSNR of Filtered image(Adaptive-Median): ',psnr(img,good\_img))

plt.subplot(121)

plt.imshow(noise\_img,cmap='gray')

plt.title("noise image")

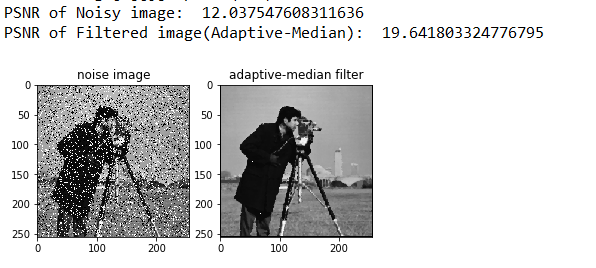
plt.subplot(122)

plt.imshow(good\_img,cmap='gray')

plt.title('adaptive-median filter')

plt.show()

**Output:**

****

**ASSIGNMENT- 7**

1. Atmospheric Blur

**Solution:**

import numpy

import cv2

import math

import matplotlib.pyplot as plt

image = cv2.imread('cameraman.tif')

m=image.shape[0]

n=image.shape[1]

greyImage = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

#generate A and H(u,v)

complexMatrix = numpy.zeros((m,n),dtype= complex)

blurMatrix = numpy.zeros((m,n),dtype=complex)

a = 0.001

b = 0.1

for i in range(0,m):

for j in range(0,n):

power = i\*j

real = math.cos((2\*math.pi\*power)/m)

img = -1\*math.sin((2\*math.pi\*power)/n)

complexMatrix[i][j] = complex(real,img)

blurMatrix[i][j] = numpy.exp(-1\*(0.0001)\*(5/6.0)\*(i\*i + j\*j))

#get the degraded image

conjMatrix = numpy.conj(complexMatrix)

dftImage = numpy.matmul(numpy.matmul((complexMatrix),greyImage) , (numpy.transpose(complexMatrix)))

blurredImage = blurMatrix\*dftImage

idftImage = numpy.matmul(numpy.matmul((numpy.transpose(conjMatrix)), blurredImage) , conjMatrix)

plt.imshow(numpy.real(idftImage),cmap='gray')

plt.title("Atmospheric blurred Image")

plt.show()

#restore the image

restoredImage = blurredImage/blurMatrix

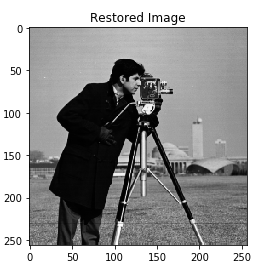
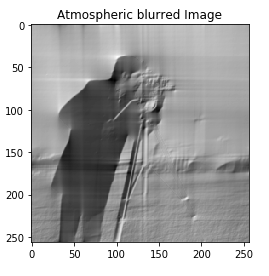
idftImage = numpy.matmul(numpy.matmul((numpy.transpose(conjMatrix)), restoredImage) , conjMatrix)

plt.imshow(numpy.real(idftImage),cmap='gray')

plt.title("Restored Image")

plt.show()

**Output:**

****

1. Motion Blur and Restoration

**Solution:**

import numpy

import cv2

import random

import math

import matplotlib.pyplot as plt

image = cv2.imread('cameraman.tif')

m=image.shape[0]

n=image.shape[1]

greyImage = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

#generate A and H(u,v)

complexMatrix = numpy.zeros((m,n),dtype= complex)

blurMatrix = numpy.zeros((m,n),dtype=complex)

a = 0.001

b = 0.1

for i in range(0,m):

for j in range(0,n):

power = i\*j

real = math.cos((2\*math.pi\*power)/m)

img = -1\*math.sin((2\*math.pi\*power)/n)

complexMatrix[i][j] = complex(real,img)

param = math.pi\*((i+1)\*a + (j+1)\*b)

real = ((math.sin(math.pi\*param)\*math.cos(math.pi\*param))/float(math.pi\*param))

img = -1\*((math.sin(math.pi\*param)\*math.sin(math.pi\*param))/float(math.pi\*param))

blurMatrix[i][j] = complex(real,img)

#get the degraded image

conjMatrix = numpy.conj(complexMatrix)

dftImage = numpy.matmul(numpy.matmul((complexMatrix),greyImage) , (numpy.transpose(complexMatrix)))

blurredImage = blurMatrix\*dftImage

idftImage = numpy.matmul(numpy.matmul((numpy.transpose(conjMatrix)), blurredImage) , conjMatrix)

plt.imshow(numpy.real(idftImage),cmap='gray')

plt.title("Atmospheric blurred Image")

plt.show()

#restore the image

restoredImage = blurredImage/blurMatrix

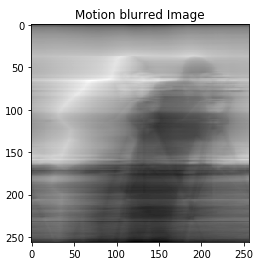
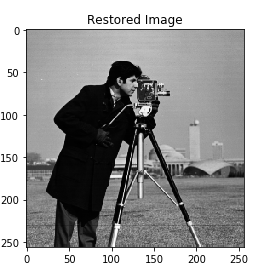
idftImage = numpy.matmul(numpy.matmul((numpy.transpose(conjMatrix)), restoredImage) , conjMatrix)

plt.imshow(numpy.real(idftImage),cmap='gray')

plt.title("Restored Image")

plt.show()

**Output:**

** **

1. Median Blur and Filtration by Weiner Filter

**Solution:**

import numpy

import cv2

import math

import matplotlib.pyplot as plt

image = cv2.imread('cameraman.tif')

m=image.shape[0]

n=image.shape[1]

greyImage = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

#generate A and H(u,v)

cMatrix = numpy.zeros((m,n),dtype= complex)

deg\_fun=numpy.zeros((m,n),dtype=complex)

a=0.001

b=0.1

T=1.0000

for i in range(0,m):

for j in range(0,n):

power = i\*j

real = math.cos((2\*math.pi\*power)/m)

img = -math.sin((2\*math.pi\*power)/n)

cMatrix[i][j] = complex(real,img)

realv=(float(T)/(math.pi\*((i+1)\*a+(j+1)\*b)))\*math.sin(math.pi\*((i+1)\*a+(j+1)\*b))\*math.cos(-math.pi\*((i+1)\*a+(j+1)\*b))

imgv=(float(T)/(math.pi\*((i+1)\*a+(j+1)\*b)))\*math.sin(math.pi\*((i+1)\*a+(j+1)\*b))\*-math.sin(math.pi\*((i+1)\*a+(j+1)\*b))

deg\_fun[i][j]=complex(realv,imgv)

matConj = numpy.conj(cMatrix)

dftImage = numpy.dot(numpy.dot((cMatrix), greyImage ) , (numpy.transpose(matConj)))

degraded\_image = deg\_fun\*dftImage

idftImage = numpy.dot(numpy.dot((numpy.transpose(matConj)), degraded\_image) , (cMatrix))

plt.subplot(121)

plt.imshow(numpy.real(idftImage),cmap='gray')

plt.title("Median Blur")

plt.subplot(122)

restored = (numpy.conj(deg\_fun)/((deg\_fun\*numpy.conj(deg\_fun))+0.00000001))\*degraded\_image

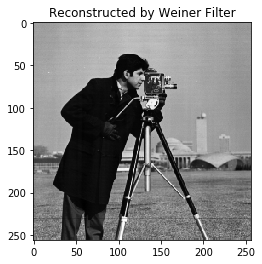
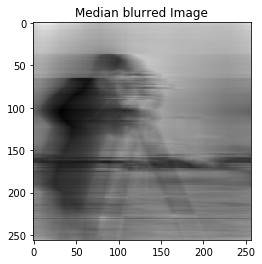
restored\_idftImage = numpy.dot(numpy.dot((numpy.transpose(matConj)), restored) , (cMatrix))

plt.imshow(numpy.real(restored\_idftImage),cmap='gray')

plt.title("Restored by Weiner Filter")

plt.show()

**Output:**

****

**ASSIGNMENT- 8**

1. KL-Transform

**Solution:**

import numpy as np

import cv2

import random

import math

import matplotlib.pyplot as plt

image = cv2.imread('cameraman.tif')

m=image.shape[0]

n=image.shape[1]

greyImage = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

cov\_mat = np.cov(greyImage)

eigan\_values, eigan\_vectors = np.linalg.eig(cov\_mat)

# k=1

# k\_arr=[]

# mse\_arr=[]

# while k<=256:

# k\_arr.append(k)

# kl\_mul = np.asarray(eigan\_vectors[:,0:k])

# compressed = np.matmul(np.transpose(kl\_mul),greyImage)

# restored = np.matmul(kl\_mul,compressed)

# mse\_ri=0

# for y in range(image.shape[0]):

# for x in range(image.shape[1]):

# mse\_ri += (float(greyImage[y][x])-float(restored[y][x]))\*\*2

# mse\_ri /= (greyImage.shape[0]\*greyImage.shape[1])

# mse\_arr.append(mse\_ri)

# k+=1

# plt.plot(k\_arr,mse\_arr)

# plt.show()

k=10

kl\_mul = np.asarray(eigan\_vectors[:,0:k])

compressed = np.matmul(np.transpose(kl\_mul),greyImage)

restored = np.matmul(kl\_mul,compressed)

k1=125

k1l\_mul = np.asarray(eigan\_vectors[:,0:k1])

compressed1 = np.matmul(np.transpose(k1l\_mul),greyImage)

restored1 = np.matmul(k1l\_mul,compressed1)

plt.imshow(restored,cmap='gray')

plt.title("Restoration using KL-Transform (K=10) ")

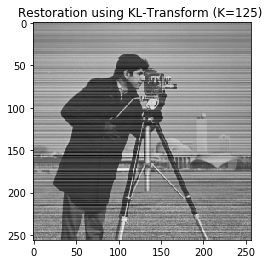
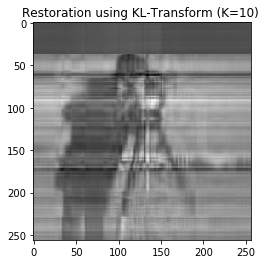
plt.show()

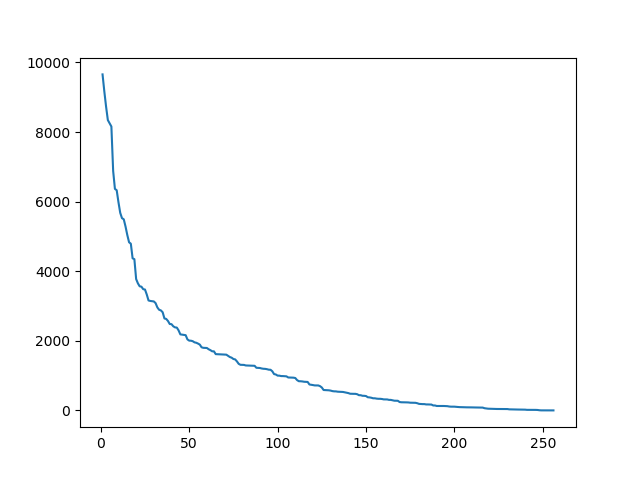
plt.imshow(restored1,cmap='gray')

plt.title("Restoration using KL-Transform (K=125) ")

plt.show()

**Output:**

****

****

MSE vs K

**ASSIGNMENT- 9**

1. Huffman Encoding

**Solution:**

import numpy as np

import cv2

import random

import math

import matplotlib.pyplot as plt

'''

image = cv2.imread('cameraman.tif')

image = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

frq = np.zeros((256))

for y in range(image.shape[0]):

for x in range(image.shape[1]):

frq[image[y][x]] = frq[image[y][x]] + 1

sum\_ = 0

for i in range(256):

sum\_ = sum\_ +frq[i]

for i in range(256):

frq[i] = frq[i]/sum\_

map\_frq = {}

for i in range(256):

map\_frq[frq[i]] = i

hoff\_map = {}

for i in sorted(map\_frq):

hoff\_map[i] = map\_frq[i]

values\_s = []

frq\_s = []

frq\_bk = []

for i in hoff\_map:

frq\_s.append(i)

frq\_bk.append(i)

values\_s.append(hoff\_map[i])

for i in range(len(frq\_s)):

print(frq\_s[i],values\_s[i])

# frq\_s.reverse()

values\_s.reverse()

'''

values\_s = [2,6,1,4,3,5]

frq\_s = [0.04,0.06,0.1,0.1,0.3,0.4]

frq\_bk = [0.04,0.06,0.1,0.1,0.3,0.4]

hoffmann\_code = {}

ind = 0

def hoffmann(arr):

global values\_s,ind,hoffmann\_code

if len(arr)==2:

hoffmann\_code[values\_s[ind]] = '1'

ind = ind+1

return '0'

post\_app = '1'

post\_ret = '0'

if arr[0]==arr[1]:

post\_app = '0'

post\_ret = '1'

temp = round(arr[0]+arr[1],15)

del arr[0]; del arr[0];

arr.append(temp)

arr.sort()

pred = hoffmann(arr)

hoffmann\_code[values\_s[ind]] = pred+post\_app

ind = ind+1

if ind==len(values\_s)-1:

hoffmann\_code[values\_s[ind]] = pred+post\_ret

return pred+post\_ret

pred = hoffmann(frq\_s)

l\_avg = 0

entropy = 0

for i in range(len(frq\_bk)):

l\_avg += frq\_bk[i]\*len(hoffmann\_code[values\_s[i]])

entropy += frq\_bk[i]\*math.log(frq\_bk[i]+0.0000001)

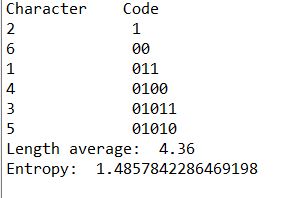
print(values\_s[i],hoffmann\_code[values\_s[i]])

print('Length average: ',l\_avg)

print('Entropy: ',-entropy)

**Output:**

Based on the example given in slide

****

1. Arithmetic Encoding

**Solution:**

import matplotlib.pyplot as plt

import numpy as np

from PIL import Image

from numpy import linalg

import operator

import math

prob = dict()

prob['a'] = 0.4

prob['b'] = 0.2

prob['c'] = 0.1

prob['d'] = 0.3

LA = dict()

HA = dict()

LL = dict()

HV = dict()

LA['a'] = 0

HA['a'] = prob['a']

LL['a'] = 0

HV['a'] = prob['a']

x = prob['a']

for i in prob:

if(i == 'a'):

continue;

LA[i] = x

LL[i] = x

HA[i] = x+prob[i]

HV[i] = x+prob[i]

x = HA[i]

print("Low Range")

print(LL)

print("High Range")

print(HV)

#for encoding

seq = "daada"

low = 0

high = 0

for i in range(len(seq)-1):

low = LL[seq[i]]

high = HV[seq[i]]

rang = high-low

for j in LL:

#print(low, rang, LA[j])

LL[j] = low + rang\*LA[j]

HV[j] = low + rang\*HA[j]

low = LL[seq[len(seq)-1]]

high = HV[seq[len(seq)-1]]

encoded\_ans = (low+high)/2

print("Encoded output: ")

print(encoded\_ans)

#for decoding

length = len(seq)

encoded = encoded\_ans

ans = ""

tag = encoded

tag\_new = 0

key = 'a'

for i in prob:

if(tag>=LA[i] and tag<=HA[i]):

ans = ans+i

key = i

for i in range(length-1):

tag = (tag-LA[key])/(HA[key]-LA[key])

for j in prob:

if(tag>=LA[j] and tag<=HA[j]):

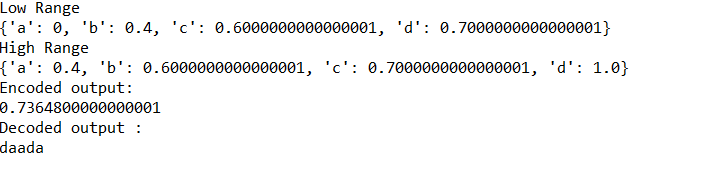
ans = ans+j

key = j

print("Decoded output :")

print(ans)

**Output:**



1. LZW- Encoding

**Solution:**

import numpy

#a,b,c only

inputString = abinput("Enter data: ")

charDict = {"a":1,"b":2,"c":3}

count = 4

check = False

i=0

while i <len(inputString):

for j in range(i+1,len(inputString)+1):

check = False

if inputString[i:j] not in charDict:

charDict[inputString[i:j]] = count

count = count +1

#print(inputString[i:j-1],charDict[inputString[i:j-1]])

check = True

i = j-1

break

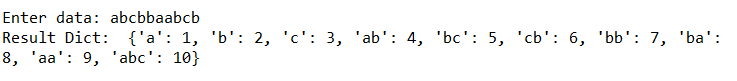
if check == False:

#print(inputString[i:len(inputString)],charDict[inputString[i:len(inputString)]])

break

print('Result Dict: ',charDict)

**Output:**



**ASSIGNMENT- 10**

1. Haar Wavelet Transform

**Solution:**

import numpy as np

import cv2

import random

import math

import matplotlib.pyplot as plt

img = cv2.imread('cameraman.tif')

img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

img = img.astype('float64')

q=0.7071068

hpf = [-q,q]

lpf = [q,q]

tempL = np.copy(img)

tempH = np.copy(img)

for x in range(img.shape[1]):

for y in range(img.shape[0]): #column wise

if y<img.shape[0]-1:

tempL[y][x] = img[y][x]\*lpf[0]+img[y+1][x]\*lpf[1]

tempH[y][x] = img[y][x]\*hpf[0]+img[y+1][x]\*hpf[1]

else:

tempL[y][x] = img[y][x]\*lpf[0]

tempH[y][x] = img[y][x]\*hpf[0]

redL = []

redH = []

for x in range(img.shape[1]): #column reduction

if x%2!=0:

redL.append(tempL[:,[x]].flatten())

redH.append(tempH[:,[x]].flatten())

redL = np.transpose(np.asarray(redL))

redH = np.transpose(np.asarray(redH))

tempLL = np.copy(redL)

tempLH = np.copy(redL)

tempHL = np.copy(redH)

tempHH = np.copy(redH)

for y in range(redL.shape[0]):

for x in range(redL.shape[1]): #row wise

if x<redL.shape[1]-1:

tempLL[y][x] = redL[y][x]\*lpf[0]+redL[y][x+1]\*lpf[1]

tempLH[y][x] = redL[y][x]\*hpf[0]+redL[y][x+1]\*hpf[1]

tempHL[y][x] = redH[y][x]\*lpf[0]+redH[y][x+1]\*lpf[1]

tempHH[y][x] = redH[y][x]\*hpf[0]+redH[y][x+1]\*hpf[1]

else:

tempLL[y][x] = redL[y][x]\*lpf[0]

tempLH[y][x] = redL[y][x]\*hpf[0]

tempHL[y][x] = redH[y][x]\*lpf[0]

tempHH[y][x] = redH[y][x]\*hpf[0]

redLL = []

redLH = []

redHL = []

redHH = []

for y in range(tempLL.shape[0]): #row reduction

if y%2==0:

redLL.append(tempLL[y][:])

redLH.append(tempLH[y][:])

redHL.append(tempHL[y][:])

redHH.append(tempHH[y][:])

redLL = np.asarray(redLL)

redLH = np.asarray(redLH)

redHL = np.asarray(redHL)

redHH = np.asarray(redHH)

print("Haar Wavelet Transform")

plt.subplot(1,2,1)

plt.imshow(redLL,cmap='gray')

plt.title("LL Component")

plt.subplot(1,2,2)

plt.imshow(redLH,cmap='gray')

plt.title("LH Component")

plt.show()

plt.subplot(1,2,1)

plt.imshow(redHL,cmap='gray')

plt.title("HL Component")

plt.subplot(1,2,2)

plt.imshow(redHH,cmap='gray')

plt.title("HH Component")

plt.show()

upLL = []

upLH = []

upHL = []

upHH = []

for y in range(redLL.shape[0]): #row enhance

upLL.append(redLL[y])

upLL.append(redLL[y])

upLH.append(redLH[y])

upLH.append(redLH[y])

upHL.append(redHL[y])

upHL.append(redHL[y])

upHH.append(redHH[y])

upHH.append(redHH[y])

upLL = np.asarray(upLL)

upLH = np.asarray(upLH)

upHL = np.asarray(upHL)

upHH = np.asarray(upHH)

tempL = np.copy(upLL)

tempH = np.copy(upHH)

for y in range(upLL.shape[0]):

for x in range(upLL.shape[1]): #row wise

if x<upLL.shape[1]-1:

tempL[y][x] = upLL[y][x]\*lpf[1]+upLL[y][x+1]\*lpf[0]+upLH[y][x]\*hpf[1]+upLH[y][x+1]\*hpf[0]

tempH[y][x] = upHL[y][x]\*lpf[1]+upHL[y][x+1]\*lpf[0]+upHH[y][x]\*hpf[1]+upHH[y][x+1]\*hpf[0]

else:

tempL[y][x] = upLL[y][x]\*lpf[1]+upLH[y][x]\*hpf[1]

tempH[y][x] = upHL[y][x]\*lpf[1]+upHH[y][x]\*hpf[1]

upL = []

upH = []

for x in range(tempL.shape[1]): #column reduction

upL.append(tempL[:,[x]].flatten())

upL.append(tempL[:,[x]].flatten())

upH.append(tempH[:,[x]].flatten())

upH.append(tempH[:,[x]].flatten())

upL = np.transpose(np.asarray(upL))

upH = np.transpose(np.asarray(upH))

upL = upL/2

upH = upH/2

restoredImg = np.copy(upL)

for x in range(upL.shape[1]):

for y in range(upL.shape[0]): #column wise

if y<upL.shape[0]-1:

restoredImg[y][x] = upL[y][x]\*lpf[1]+upL[y+1][x]\*lpf[0]+upH[y][x]\*hpf[1]+upH[y+1][x]\*hpf[0]

else:

restoredImg[y][x] = upL[y][x]\*lpf[1]+upH[y][x]\*hpf[1]

restoredImg = restoredImg/2

m,n=img.shape

sum\_=0

for i in range(m):

for j in range(n):

sum\_=sum\_+(img[i][j]-restoredImg[i][j])\*\*2

print('Mean Square Error is: ',str(math.sqrt(sum\_/(m\*n))))

plt.imshow(restoredImg,cmap='gray')

plt.show()

**Output:**

