DIP LAB ASSIGNMENTS

DAY 1 14/08/2019

1. WRITE A PROGRAM TO READ, WRITE, AND STORE A GRAY LEVEL IMAGE, A COLOR IMAGE AND A BINARY IMAGE IF POSSIBLE. IF POSSIBLE, DISPLAY STORAGE REQUIREMEENT OF IMAGE.

```
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

img=cv2.imread('abc.jpg')

for i in range(0, len(img)):
  for j in range(0, len(img[0])):
    print(img[i][j])
  print("\n")

cv2.imshow('image',img)
cv2.imwrite("out.png", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. WRITE A PROGRAM TO ADD TWO GRAY LEVEL IMAGES OF SAME SIZE AND DISPLAY THE OUTPUT.

```
import numpy as np
import cv2 as cv
import math
import array

image1 = cv.imread('cat.jpg',0)
image2 = cv.imread('building.jpg',0)
image3 = np.zeroes([256,256], dtype='float32')

height,width =image1.shape

for i in range(0,height-1):
        for j in range(0,width-1):
```

```
                image3[i,j]=(image1[i,j]/2 + image2[i,j]/2)

    cv.imshow('first image',image1)
    cv.imshow('second image', image2)
    cv.imshow('addition image', image3)

    cv.waitKey()
    cv.destroyAllWindows()
```

3. WRITE A PROGRAM TO TRANSFORM 256 GRAY LEVELS OF A GRAY LEVEL IMAGE INTO 8 DIFFERENT GRAY LEVELS AND THEN MULTIPLY 1,2,3,…,8 WITH 8 DIFFERENT GRAY LEVELS IN DECREASING ORDER. DISPLAY THE OUTPUT IMAGE.

```
    import cv2
    import numpy as np

    img1 = cv2.imread('lena_color_512.tif', 0)
    img2 = cv2.imread('lena_color_512.tif', 0)

    h,w=img1.shape

    for i in range(0,h-1):
        for j in range(0,w-1):
            k=img1[i,j]
            if(k>=0 and k<=31):
                img2[i,j]=15
                img2[i,j] = (img2[i,j]*8)%256
            elif(k>=32 and k<=63):
                img2[i,j]=47
                img2[i,j] = (img2[i,j]*7)%256
            elif(k>=64 and k<=95):
                img2[i,j]=79
                img2[i,j] = (img2[i,j]*6)%256
            elif(k>=96 and k<=127):
                img2[i,j]=111
                img2[i,j] = (img2[i,j]*5)%256
            elif(k>=128 and k<=159):
                img2[i,j]=143
                img2[i,j] = (img2[i,j]*4)%256
            elif(k>=160 and k<=191):
                img2[i,j]=175
                img2[i,j] = (img2[i,j]*3)%256
            elif(k>=192 and k<=223):
```

```
            img2[i,j]=212
            img2[i,j] = (img2[i,j]*2)%256
        elif(k>=224 and k<=255):
            img2[i,j]=239
            img2[i,j] = (img2[i,j]*1)%256

    cv2.imshow('normal',img1)
    cv2.imshow('modified image',img2)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

4. WRITE A PROGRAM TO REDUCE THE GRAY LEVEL FROM 256 TO 128, 64, 32, 16, 8, 4 AND 2 OF A MONOCHROMATIC IMAGE.

```
    import numpy as np
    import cv2 as cv
    import matplotlib.pyplot as plt
    image1 = cv.imread('model.jpg',0)

    x = int(input("enter the gray level you want to convert to: "))
    image1 = image1//(256/x)

    cv.imshow("Converted image",image1)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

5. WRITE A PROGRAM TO ZOOM AND SHRINK A GRAY LEVEL IMAGE AT A DESIRED LEVEL. APPLY OVERSAMPLING AND UNDERSAMPLING OF THE IMAGE.

```
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt

    image = cv2.imread('tomatoes.jpg',1)

    half = cv2.resize(image, (0,0), fx = 0.1, fy = 0.1)
    bigger = cv2.resize(image, (1050, 1610))

    stretch_near = cv2.resize(image, (780, 540), interpolation =
cv2.INTER_NEAREST)
```

```
Titles = ["Original", "Half", "Bigger", "Interpolation Nearest"]
images = [image, half, bigger, stretch_near]
count = 4

for i in range(count):
    plt.subplot(2, 2, i + 1)
    plt.title(Titles[i])
    plt.imshow(images[i])

plt.show()
```

6. WRITE A PROGRAM TO DECOMPOSE AN IMAGE INTO EIGHT 1-BIT PLANES RANGING FROM 0TH BIT PLANE TO 7TH BIT PLANE AND SET 0 TO THE MOST SIGNIFICANT BITS(FIRST 4 BITS). THEN SUBTRACT THE RESULT FROM INPUT IMAGE AND ENHANCE THE RESULTANT IMAGE BY HISTOGRAM PROCESSING OPERATION. DISPLAY ALL IMAGES.

```
import cv2
import numpy as np

img1 = cv2.imread('lena_color_512.tif', 0)
img2 = img1 & 15
img3 = img1 - img2
equ = cv2.equalizeHist(img3)

cv2.imshow('original_image', img1)
cv2.imshow('modified_image', img2)
cv2.imshow('subtracted_image', img3)
cv2.imshow('equalized_image', equ)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. REPEAT THE ABOVE QUESTION FOR LEAST SIGNIFICANT BITS(LAST 4 BITS).

```
import cv2
import numpy as np

img1 = cv2.imread('lena_color_512.tif', 0)
img2 = img1 & 240
img3 = img1 - img2
equ = cv2.equalizeHist(img3)
```

```
cv2.imshow('original_image', img1)
cv2.imshow('modified_image', img2)
cv2.imshow('subtracted_image', img3)
cv2.imshow('equalized_image', equ)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

8. WRITE A PROGRAM TO ENHANCE A LOW CONTRAST IMAGE USING DIFFERENT IMAGE ENHANCEMENT TECHNIQUES.

```
import cv2
import numpy as np
def  powerlaw():
    im=cv2.imread('wiki.jpg',0)
    im = im/255.0
    im_power_law_transformation= cv2.pow(im,1.8)
    cv2.imshow('Original Image',im)
    cv2.imshow('Power Law Transformation',im_power_law_transformation)
    cv2.waitKey(0)

def inverse():
    im = cv2.imread('wiki.jpg')
    im_inverse = 255-im
    cv2.imshow('Original Image',im)
    cv2.imshow('Image Inverse',im_inverse)
    cv2.waitKey(0)


a = int(input("ENTER 1 FOR PLT and 2 FOR INVERSE:"))

if a == 1:
    powerlaw()
else:
    inverse()
```

9. WRITE A PROGRAM TO ENHANCE A LOW CONTRAST IMAGE USING HISTOGRAM EQUALIZATION AND HISTOGRAM MATCHING AND ANALYZE THE RESULT.

```
import cv2
import numpy as np
```

```
from matplotlib import pyplot as plt

img = cv2.imread('wiki.jpg',0)

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()
cdf_normalized = cdf * hist.max()/ cdf.max()

plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()


equ = cv2.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side
cv2.imwrite('res.png',res)

cv2.imshow("result",res)
cv2.waitKey(0)
```

10. ADD NOISE TO THE IMAGE

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import random

def rayleigh_noise(image):
   meanvalue=100
   modevalue=np.sqrt(2/np.pi)*meanvalue
   s=np.random.rayleigh(modevalue,(177,284))
   plt.hist(s.ravel(),255,[0,255])
   plt.title('exponential noise histogram')
   plt.show()
   noisy=image+s
   return noisy


img=cv2.imread("messi.jpg",0)
noise_img=rayleigh_noise(img)
plt.subplot(131)
```

```
plt.imshow(img,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Original')
plt.subplot(132)
plt.imshow(noise_img,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('noisy image')
plt.subplot(133)
plt.hist(noise_img[100:200,100:200].ravel(),255,[0,255])
plt.title('histogram')
plt.show()
cv2.waitKey(0)
```

11. WRITE A PROGRAM TO FIND 4, 8, m ADJACENT AMONG THE PIXELS FOR V = {1}.

```
def N4(a, i , j):
  s = set()
  if (i -1) >= 0:
    s.add(a[i-1][j])
  if (i+1) < len(a):
    s.add(a[i+1][j])
  if (j-1) >= 0:
    s.add(a[i][j-1])
  if (j+1) < len(a[0]):
    s.add(a[i][j+1])
  return s

def N8(a, i, j):
  s = set()
  row = len(a)
  col = len(a[0])
  for x in N4(a, i, j):
   s.add(x)
  if (i+1 < row) and (j+1 < col):
    s.add(a[i+1][j+1])
  if (i-1 >= 0) and (j-1 >= 0):
    s.add(a[i-1][j-1])
  if (i-1 >= 0) and (j+1 < col):
    s.add(a[i-1][j+1])
  if (i+1 < row) and (j-1 >= 0):
    s.add(a[i+1][j-1])
  return s
```

```
def Nm(a, i, j, v):
    s = set()
    s1 = N4(a, i, j)
    for x in s1:
      s.add(x)
    if (i+1) < len(a) and (j+1) < len(a[0]):
      s2 = N4(a, i+1, j+1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    if (i-1) >= 0 and (j-1) >= 0:
      s2 = N4(a, i-1, j-1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    if (i+1) < len(a) and (j-1) >= 0:
      s2 = N4(a, i+1, j-1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    if (i-1) >= 0 and (j+1) < len(a[0]):
      s2 = N4(a, i-1, j+1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    return s


arr = [[0, 1, 1 , 0, 1],
[1, 1, 0 , 1, 1],
[1, 0, 1 , 1, 1],
[0, 1, 0 , 1, 1],
[0, 1, 1 , 1, 0]]

v = {1}

for i in range(len(arr)):
  for j in range(len(arr[0])):
     print(N4(arr, i, j), end = " ")

print("\n")
```

```
for i in range(len(arr)):
  for j in range(len(arr[0])):
     print(N8(arr, i, j), end = " ")
print("\n")

for i in range(len(arr)):
  for j in range(len(arr[0])):
     print(Nm(arr, i, j, v), end = " ")
print("\n")
```

12. WRITE A PROGRAM TO FIND 4, 8, m ADJACENT AMONG THE PIXELS FOR V = {5, 10, 15} .

```
def N4(a, i , j):
   s = set()
   if (i -1) >= 0:
     s.add(a[i-1][j])
   if (i+1) < len(a):
     s.add(a[i+1][j])
   if (j-1) >= 0:
     s.add(a[i][j-1])
   if (j+1) < len(a[0]):
     s.add(a[i][j+1])
   return s

#8-adjacency
def N8(a, i, j):
   s = set()
   row = len(a)
   col = len(a[0])
   for x in N4(a, i, j):
    s.add(x)
   if (i+1 < row) and (j+1 < col):
     s.add(a[i+1][j+1])
   if (i-1 >= 0) and (j-1 >= 0):
     s.add(a[i-1][j-1])
   if (i-1 >= 0) and (j+1 < col):
     s.add(a[i-1][j+1])
   if (i+1 < row) and (j-1 >= 0):
     s.add(a[i+1][j-1])
   return s

#m-adjacency
```

```
def Nm(a, i, j, v):
    s = set()
    s1 = N4(a, i, j)
    for x in s1:
      s.add(x)
    if (i+1) < len(a) and (j+1) < len(a[0]):
      s2 = N4(a, i+1, j+1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    if (i-1) >= 0 and (j-1) >= 0:
      s2 = N4(a, i-1, j-1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    if (i+1) < len(a) and (j-1) >= 0:
      s2 = N4(a, i+1, j-1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    if (i-1) >= 0 and (j+1) < len(a[0]):
      s2 = N4(a, i-1, j+1)
      s3 = s1 & s2
      s4 = s3 - v
      for x in s4:
        s.add(x)
    return s

#given binary img
img = [[10, 4, 4, 4, 5, 5, 5, 15, 15],
[15, 15, 15 , 4, 4, 5, 5, 15, 2],
[5, 2, 15, 3, 3, 3, 5, 5, 1],
[10, 3, 5, 5, 4, 10, 10, 4, 1],
[10, 2, 4, 5, 15, 5, 10, 5, 10],
[5, 5, 5, 5, 7, 7, 7, 15, 5],
[15, 4, 10, 10, 10, 10, 7, 7, 5],
[15, 4, 15, 15, 5, 10, 7, 10, 10],
[15, 15, 5, 5, 5, 10, 10, 10, 10]]

v = {5, 10, 15}

for i in range(len(img)):
```

```
  for j in range(len(img[0])):
    print(N4(img, i, j), end = " ")

print("\n")

for i in range(len(img)):
  for j in range(len(img[0])):
    print(N8(img, i, j), end = " ")
print("\n")

for i in range(len(img)):
  for j in range(len(img[0])):
    print(Nm(img, i, j, v), end = " ")
print("\n")
```

DAY 2 04/09/2019

1. WRITE A PROGRAM TO OBTAIN ONE DIMENSIONAL DISCRETE FOURIER TRANSFORM OF A GIVEN ONE-DIMENSIONAL VECTOR CONSISTS OF INTEGER NUMBERS GENERATED RANDOMLY.

```
import numpy as np
v = []
n = input("Enter a integer:")
n = int(n)
for i in range(0, n):
   ele = int(input())
   v.append(ele)
print(np.fft.fft(v))
```

2. TWO-DIMENSIONAL DISCRETE FOURIER TRANSFORM AND ITS INVERSE OF A GRAY LEVEL IMAGE OF SIZE 500*500. ALSO COMPUTE MAGNITUDE, PHASE ANGLE AND POWER SPECTRUM OF FOURIER TRANSFORM.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('messi.jpg',0)
img = cv2.resize(img, (500, 500))
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
```

```
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()

rows, cols = img.shape
crow,ccol = rows/2 , cols/2

# create a mask first, center square is 1, remaining all zeros
'''mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1'''

# apply mask and inverse DFT
fshift = dft_shift
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()



#power spectrum
from scipy import fftpack, ndimage
import matplotlib.pyplot as plt

image = plt.imread('messi.jpg', 0)
fft2 = fftpack.fft2(image)

plt.imshow(abs(fft2))
plt.show()
```

4. WRITE A PROGRAM TO DEMONSTRATE BASICS OF FILTERING IN THE FREQUENCY DOMAIN.

```
import cv2
```

```
import numpy as np
from matplotlib import pyplot as plt

# simple averaging filter without scaling parameter
mean_filter = np.ones((3,3))

# creating a guassian filter
x = cv2.getGaussianKernel(5,10)
gaussian = x*x.T

# different edge detecting filters
# scharr in x-direction
scharr = np.array([[-3, 0, 3],
                   [-10,0,10],
                   [-3, 0, 3]])
# sobel in x direction
sobel_x= np.array([[-1, 0, 1],
                   [-2, 0, 2],
                   [-1, 0, 1]])
# sobel in y direction
sobel_y= np.array([[-1,-2,-1],
                   [0, 0, 0],
                   [1, 2, 1]])
# laplacian
laplacian=np.array([[0, 1, 0],
                    [1,-4, 1],
                    [0, 1, 0]])

filters = [mean_filter, gaussian, laplacian, sobel_x, sobel_y, scharr]
filter_name = ['mean_filter', 'gaussian','laplacian', 'sobel_x', \
               'sobel_y', 'scharr_x']
fft_filters = [np.fft.fft2(x) for x in filters]
fft_shift = [np.fft.fftshift(y) for y in fft_filters]
mag_spectrum = [np.log(np.abs(z)+1) for z in fft_shift]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(mag_spectrum[i],cmap = 'gray')
    plt.title(filter_name[i]), plt.xticks([]), plt.yticks([])

plt.show()
```

5. WRITE A PROGRAM TO APPLY LOW PASS AND HIGH PASS FILTERS TO A GRAY LEVEL IMAGE OF SIZE 500*500 AND DISPLAY THE FILTERED IMAGE.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('messi.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

rows, cols = img.shape
crow,ccol = int(rows/2) , int(cols/2)

# create a mask first, center square is 1, remaining all zeros
mask = np.zeros((rows,cols),np.uint8)
for i in range(crow-30, crow +30):
   for j in range(ccol - 30, ccol +30):
               mask[i][j] = 1
#mask[crow-30:crow+30, ccol-30:ccol+30] = 1

# apply mask and inverse DFT
fshift = fshift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)

plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.hist(img_back.ravel(),256,[0,256])
plt.title('Histogram'), plt.xticks([]), plt.yticks([])
plt.show()
```

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('messi.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
#print(f)
'''magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
```

```
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()'"
#High Pass Filter
rows, cols = img.shape
crow,ccol = int(rows/2) , int(cols/2)
for i in range(crow-30, crow +30):
  for j in range(ccol - 30, ccol +30):
            fshift[i][j] = 0 + 0j

f_ishift = np.fft.ifftshift(fshift)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)

plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.hist(img_back.ravel(),256,[0,256])
plt.title('Histogram'), plt.xticks([]), plt.yticks([])
plt.show()
```

6. WRITE A PROGRAM TO APPLY FAST FOURIER TRANSFORM TO A GRAY LEVEL IMAGE AND OBTAIN THE HISTOGRAM OF THE OUTPUT IMAGE.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('messi.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))
plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Fast Fourier Transform'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.hist(magnitude_spectrum.ravel(),256,[0,256])
plt.title('Histogram'), plt.xticks([]), plt.yticks([])
plt.show()
```

DAY 3 **18/09/2019**

1. WRITE A PROGRAM TO ESTIMATE NOISE FUNCTION IN NOISY IMAGE.

```
import numpy as np
import os
import cv2
from matplotlib import pyplot as plt
def noisy(noise_typ,image):
  if noise_typ == "gauss":
    row,col,ch= image.shape
    mean = 0
    var = 0.1
    sigma = var**0.5
    gauss = np.random.normal(mean,sigma,(row,col,ch))
    gauss = gauss.reshape(row,col,ch)
    noisy = image + gauss
    return noisy
  elif noise_typ == "s&p":
    row,col,ch = image.shape
    s_vs_p = 0.5
    amount = 0.004
    out = np.copy(image)
    # Salt mode
    num_salt = np.ceil(amount * image.size * s_vs_p)
    coords = [np.random.randint(0, i - 1, int(num_salt))
          for i in image.shape]
    out[coords] = 1

    # Pepper mode
    num_pepper = np.ceil(amount* image.size * (1. - s_vs_p))
    coords = [np.random.randint(0, i - 1, int(num_pepper))
          for i in image.shape]
    out[coords] = 0
    return out
  elif noise_typ == "poisson":
    vals = len(np.unique(image))
    vals = 2 ** np.ceil(np.log2(vals))
    noisy = np.random.poisson(image * vals) / float(vals)
    return noisy
```

```
    elif noise_typ =="speckle":
      row,col,ch = image.shape
      gauss = np.random.randn(row,col,ch)
      gauss = gauss.reshape(row,col,ch)
      noisy = image + image * gauss
      return noisy

img = cv2.imread("messi.jpg", 0)
ret = noisy("guass", img)
plt.subplot(111),plt.imshow(ret, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])

cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. WRITE A PROGRAM TO ADD NOISE TO IMAGE.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import random

def sp_noise(image,prob):
  output=np.zeros(image.shape,np.uint8)
  thres=1-prob
  for i in range(image.shape[0]):
        for j in range(image.shape[1]):
              rdn=random.random()
              if rdn <prob:
                    output[i][j]=0
              elif rdn>thres:
                    output[i][j]=255
              else:
                    output[i][j]=image[i][j]
  return output


img=cv2.imread("messi.jpg",0)
noise_img=sp_noise(img,0.05)
plt.subplot(131)
plt.imshow(img,cmap='gray')
plt.title('original')
plt.xticks([]),plt.yticks([])
plt.subplot(132)
```

```
plt.imshow(noise_img,cmap='gray')
plt.title('noisy image')
plt.xticks([]),plt.yticks([])
plt.subplot(133)
plt.hist(noise_img[100:200,100:200].ravel(),255,[0,255])
plt.title('histogram')
plt.show()
cv2.waitKey(0)
```

3. WRITE A PROGRAM TO ADD GAUSSIAN NOISE.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import random

def gaussian_noise(image):
  row,col=image.shape
  mean=0
  var=250
  sigma=var**0.5
  gauss=np.random.normal(mean,sigma,(row,col))
  gauss=gauss.reshape(row,col)
  noisy=image+gauss
  return noisy


img=cv2.imread("messi.jpg",0)
noise_img=gaussian_noise(img)
plt.subplot(131)
plt.imshow(img,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Original')
plt.subplot(132)
plt.imshow(noise_img,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('noisy image')
plt.subplot(133)
plt.hist(noise_img[100:200,100:200].ravel(),255,[0,255])
plt.title('histogram')
plt.show()
cv2.waitKey(0)
```

DAY 4 16/10/2019

1. WRITE A PROGRAM TO APPLY-

a) EROSION

b) OPENING OF ERODED IMAGE

c) DILATION OF RESULT BT STEP b)

d) CLOSING OF RESULT OBTAINED BY STEP b)

```
import cv2
import numpy as np

img = cv2.imread('finger.jpg', 0)

kernel = np.ones((3,3), np.uint8)

img_erosion = cv2.erode(img, kernel, iterations=1)

opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

img_dilation = cv2.dilate(img, kernel, iterations=1)

closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

cv2.imshow('Input', img)
cv2.waitKey(0)

cv2.imshow('opening', opening)
cv2.waitKey(0)

cv2.imshow('Erosion', img_erosion)
cv2.waitKey(0)

cv2.imshow('Dilation', img_dilation)
cv2.waitKey(0)

cv2.imshow('closing', closing)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

2. WRITE A PROGRAM TO EXTRACT THE BOUNDARY OF AN OBJECT GIVEN IN AN IMAGE WITH THE HELP OF STRUCTURING ELEMENT.

```
import cv2
import numpy as np

# Let's load a simple image with 3 black squares
image = cv2.imread('man.jpg')
cv2.waitKey(0)

# Grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Find Canny edges
edged = cv2.Canny(gray, 30, 200)
cv2.waitKey(0)

# Finding Contours
# Use a copy of the image e.g. edged.copy()
# since findContours alters the image
contours, hierarchy = cv2.findContours(edged,
  cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

cv2.imshow('Canny Edges After Contouring', edged)
cv2.waitKey(0)

print("Number of Contours found = " + str(len(contours)))

# Draw all contours
# -1 signifies drawing all contours
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. WRITE A PROGRAM TO FILL A REGION OF AN IMAGE WITH THE HELP OF AN APPROPIATE STRUCTURING ELEMENT.

```
import cv2;
import numpy as np;

# Read image
im_in = cv2.imread("fill.jpg", cv2.IMREAD_GRAYSCALE);

# Threshold.
# Set values equal to or above 220 to 0.
# Set values below 220 to 255.

th, im_th = cv2.threshold(im_in, 220, 255, cv2.THRESH_BINARY_INV);

# Copy the thresholded image.
im_floodfill = im_th.copy()

# Mask used to flood filling.
# Notice the size needs to be 2 pixels than the image.
h, w = im_th.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)

# Floodfill from point (0, 0)
cv2.floodFill(im_floodfill, mask, (0,0), 255);

# Invert floodfilled image
im_floodfill_inv = cv2.bitwise_not(im_floodfill)

# Combine the two images to get the foreground.
im_out = im_th | im_floodfill_inv

# Display images.
cv2.imshow("Thresholded Image", im_th)
cv2.imshow("Floodfilled Image", im_floodfill)
cv2.imshow("Inverted Floodfilled Image", im_floodfill_inv)
cv2.imshow("Foreground", im_out)
cv2.waitKey(0)
```

DAY 5 23/10/2019

1. WRITE A PROGRAM TO DETECT POINTS IN THE GIVEN GRAY LEVEL IMAGE WITH SUITABLE MASK.

```
import cv2
import numpy as np
```

```
img = cv2.imread('points.jpg',0)
dim = img.shape

laplacian = [[0,0,0],[1,-8,1],[0,0,0]]


new = np.zeros((dim[0],dim[1]))

for x in range(1,dim[0]-1) :
  for y in range(1,dim[1]-1) :
    pixel_x = ((laplacian[0][0] * img[x-1][y-1]) + (laplacian[0][1] * img[x][y-1]) +
(laplacian[0][2] * img[x+1][y-1])+
            (laplacian[1][0] * img[x-1][y])   + (laplacian[1][1] * img[x][y]) +
(laplacian[1][2] * img[x+1][y]) +
            (laplacian[2][0] * img[x-1][y+1])   + (laplacian[2][1] * img[x][y+1]) +
(laplacian[2][2] * img[x+1][y+1]) )

    val = pixel_x
    new[x,y] = np.ceil(val)


cv2.imshow('Image', new)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. WRITE A PROGRAM TO DETECT LINES IN HORIZONTAL AND VERTICAL DIRECTIONS IN GIVEN IMAGE WITH SUITABLE MASKS.

```
#vertical
#from skimage.exposure import rescale_intensity
import numpy as np
import cv2

def convolve(image, kernel):
   (iH, iW) = image.shape[:2]
   (kH, kW) = kernel.shape[:2]
   pad = (kW - 1) // 2
   image = cv2.copyMakeBorder(image, pad, pad, pad, pad,
               cv2.BORDER_REPLICATE)
   output = np.zeros((iH, iW), dtype="float32")
   for y in np.arange(pad, iH + pad):
               for x in np.arange(pad, iW + pad):
                           roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]
                           k = (roi * kernel).sum()
```

```
                output[y - pad, x - pad] = k
    return output

point_dect = np.array((
    [-1,  2, -1],
    [-1,  2, -1],
    [-1,  2, -1]), dtype="int")

image = cv2.imread("house.jpg", 0)
gray = image
cv2.imshow("image", image)
cv2.waitKey(0)

convoleOutput = convolve(gray, point_dect)
cv2.imshow("image", convoleOutput)
cv2.waitKey(0)
cv2.destroyAllWindows()



#Horizontal
#rom skimage.exposure import rescale_intensity
import numpy as np
import cv2

def convolve(image, kernel):
    (iH, iW) = image.shape[:2]
    (kH, kW) = kernel.shape[:2]
    pad = (kW - 1) // 2
    image = cv2.copyMakeBorder(image, pad, pad, pad, pad,
            cv2.BORDER_REPLICATE)
    output = np.zeros((iH, iW), dtype="float32")
    for y in np.arange(pad, iH + pad):
            for x in np.arange(pad, iW + pad):
                    roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]
                    k = (roi * kernel).sum()
                    output[y - pad, x - pad] = k
    return output

point_dect = np.array((
    [-1, -1, -1],
    [2,  2, 2],
    [-1, -1, -1]), dtype="int")

image = cv2.imread("house.jpg", 0)
gray = image
```

```
cv2.imshow("image", image)
cv2.waitKey(0)

convoleOutput = convolve(gray, point_dect)
cv2.imshow("image", convoleOutput)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. WRITE A PROGRAM TO DETECT EDGES IN THE GIVEN IMAGE WITH A SUITABLE EDGE DETECTION OPERATOR AND THEN FURTHER DETECT EDGE POINTS USING SUITABLE MASK IN THE RESULTANT EDGED IMAGE.

```
import cv2
import numpy as np
#from matplotlib import pyplot as plt

img = cv2.imread('messi.jpg',0)
edges = cv2.Canny(img,100,200)
cv2.imshow("image1", edges)
cv2.waitKey(0)

def convolve(image, kernel):
  (iH, iW) = image.shape[:2]
  (kH, kW) = kernel.shape[:2]
  pad = (kW - 1) // 2
  image = cv2.copyMakeBorder(image, pad, pad, pad, pad,
          cv2.BORDER_REPLICATE)
  output = np.zeros((iH, iW), dtype="float32")
  for y in np.arange(pad, iH + pad):
          for x in np.arange(pad, iW + pad):
                  roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]
                  k = (roi * kernel).sum()
                  output[y - pad, x - pad] = k
  return output

point_dect = np.array((
  [-1, -1, -1],
  [-1,  3, -1],
  [-1, -1, -1]), dtype="int")

convoleOutput = convolve(edges, point_dect)
cv2.imshow("image2", convoleOutput)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4. WRITE A PROGRAM TO FIND ALIGNMENT OF A SET OF EDGE POINTS USING HOUGH TRANSFORM.

```
# Read image
img = cv2.imread('lanes.jpg', cv2.IMREAD_COLOR)
# Convert the image to gray-scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Find the edges in the image using canny detector
edges = cv2.Canny(gray, 50, 200)
# Detect points that form a line
lines = cv2.HoughLinesP(edges, 1, np.pi/180, max_slider, minLineLength=10,
maxLineGap=250)
# Draw lines on the image
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)
# Show result
cv2.imshow("Result Image", img)
```