Project 2

Shubham Miglani 200284655

Question 1:

Padding results:

5 by 5 filter and 3 by 3 input matrix are considered as an example:

Input Matrix:

[[0 1 2]

[3 4 5]

[6 7 8]]

Filter :

[[0. 0.04 0.08 0.12 0.16]

[0.2 0.24 0.28 0.32 0.36]

[0.4 0.44 0.48 0.52 0.56]

[0.6 0.64 0.68 0.72 0.76]

[0.8 0.84 0.88 0.92 0.96]]

1) Zero Padding:

[[0 0 0 0 0 0 0]

[0 0 0 0 0 0 0]

[0 0 0 1 2 0 0]

[0 0 3 4 5 0 0]

[0 0 6 7 8 0 0]

[0 0 0 0 0 0 0]

[0 0 0 0 0 0 0]]

2) Wrap around/circular:

[[4 5 3 4 5 3 4]

[7 8 6 7 8 6 7]

[1 2 0 1 2 0 1]

[4 5 3 4 5 3 4]

[7 8 6 7 8 6 7]

[1 2 0 1 2 0 1]

[4 5 3 4 5 3 4]]

3) Copy edge/replicate:

[[0 0 0 1 2 2 2]

[0 0 0 1 2 2 2]

[0 0 0 1 2 2 2]

[3 3 3 4 5 5 5]

[6 6 6 7 8 8 8]

[6 6 6 7 8 8 8]

[6 6 6 7 8 8 8]]

4) Reflect across edge/symmetric:

[[4 3 3 4 5 5 4]

[1 0 0 1 2 2 1]

[1 0 0 1 2 2 1]

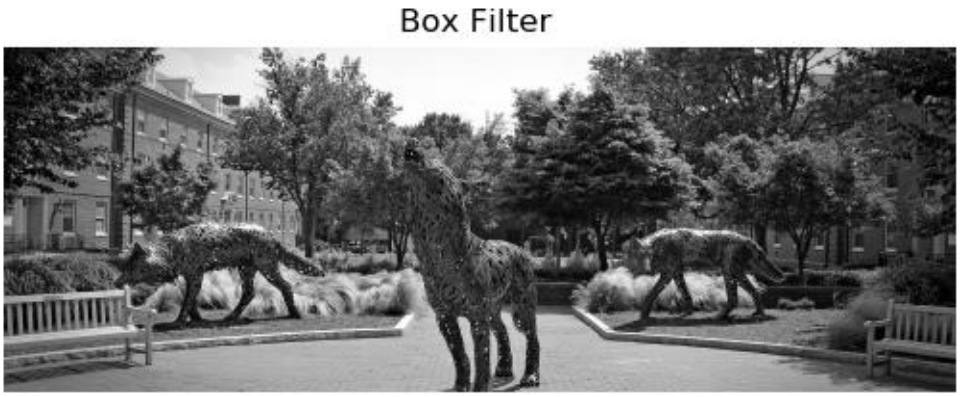
[4 3 3 4 5 5 4]

[7 6 6 7 8 8 7]

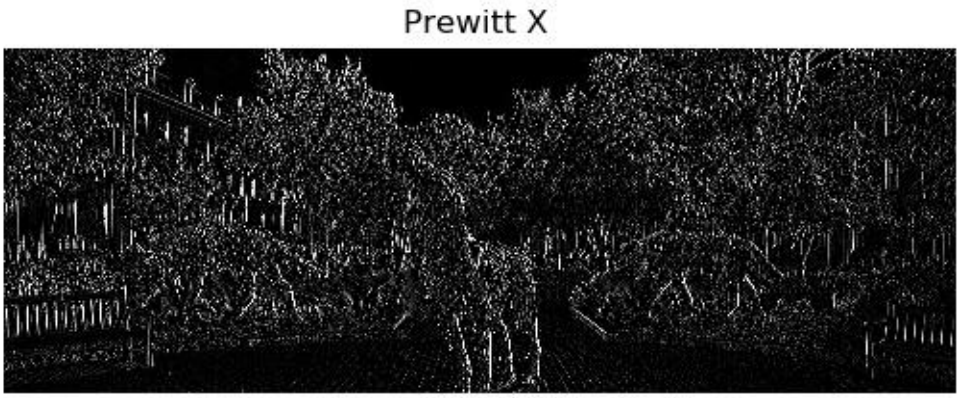
[7 6 6 7 8 8 7]

[4 3 3 4 5 5 4]]

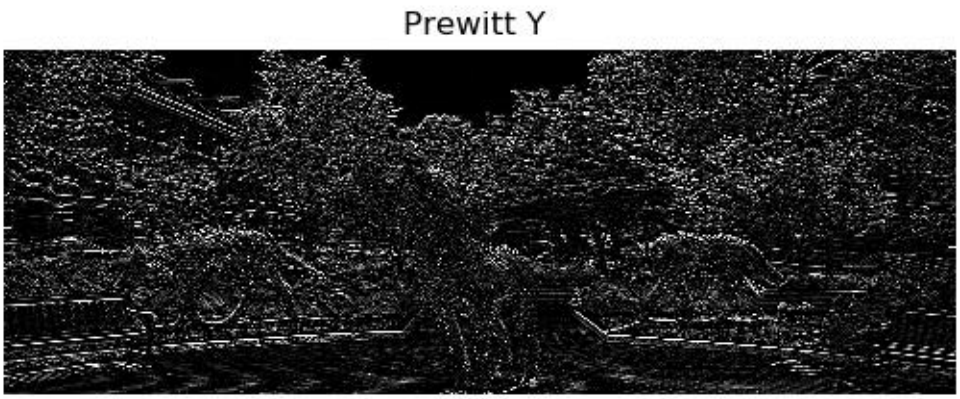
Box Filter:



PrewittMx:



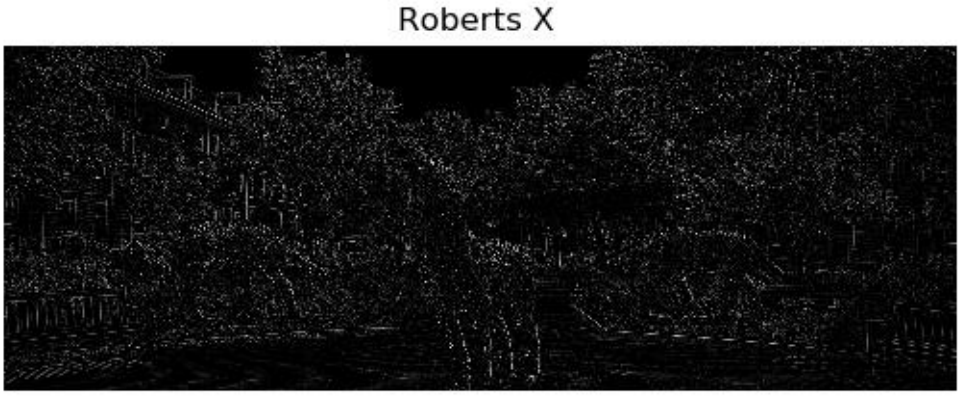
PrewittMy:



Sobel x:

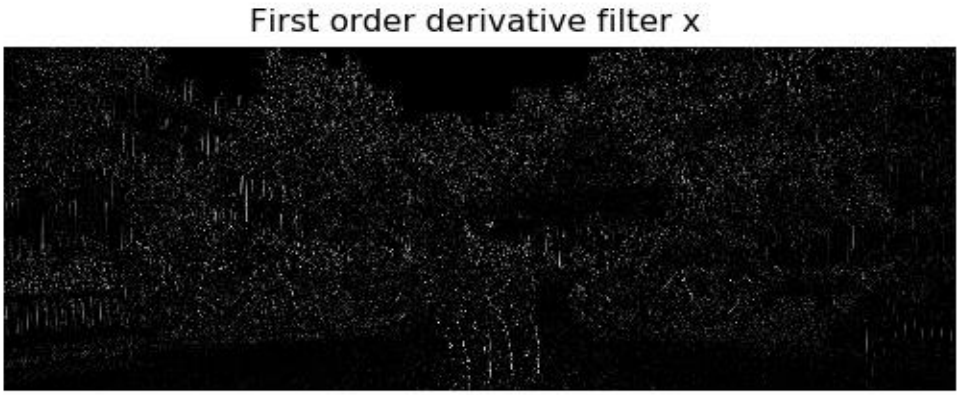
Sobel y:

Roberts x:

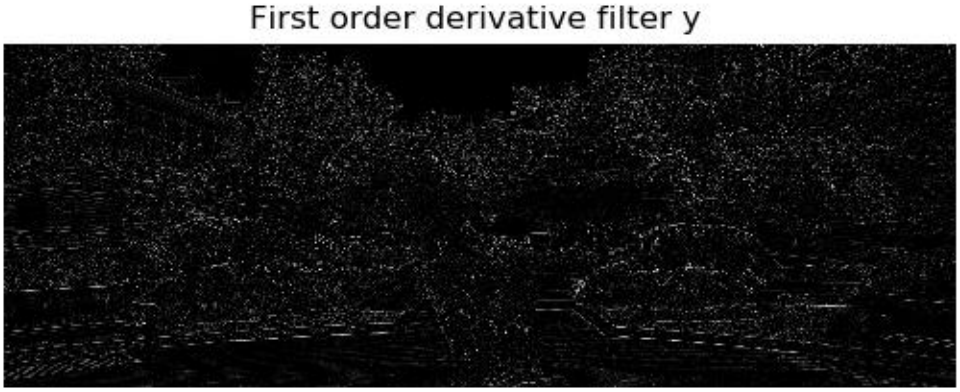


Roberts y:

First order derivative x

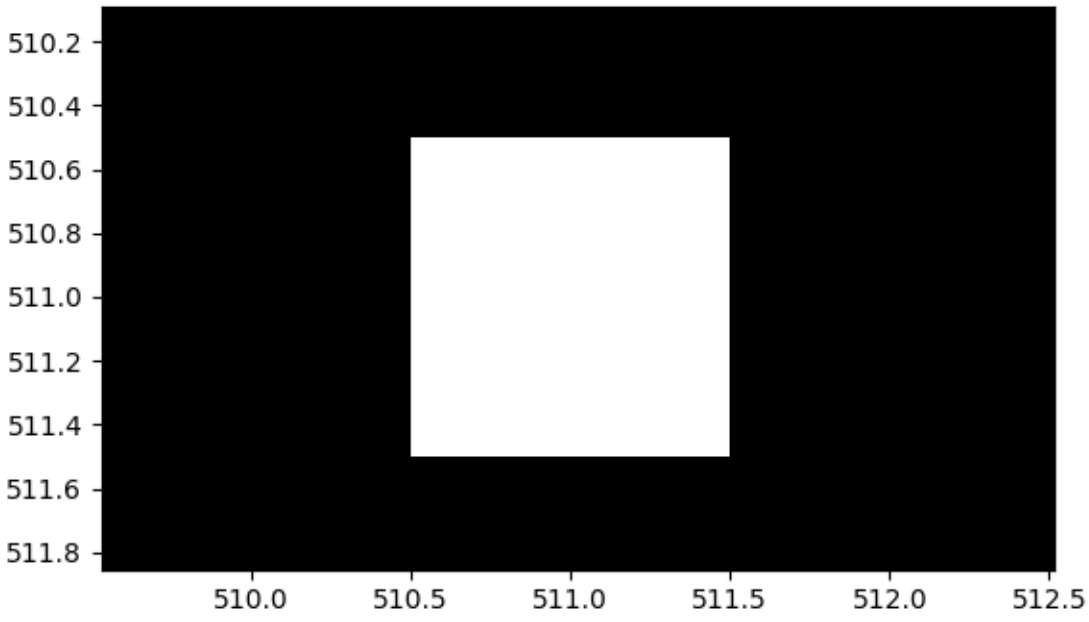


First order derivative y:

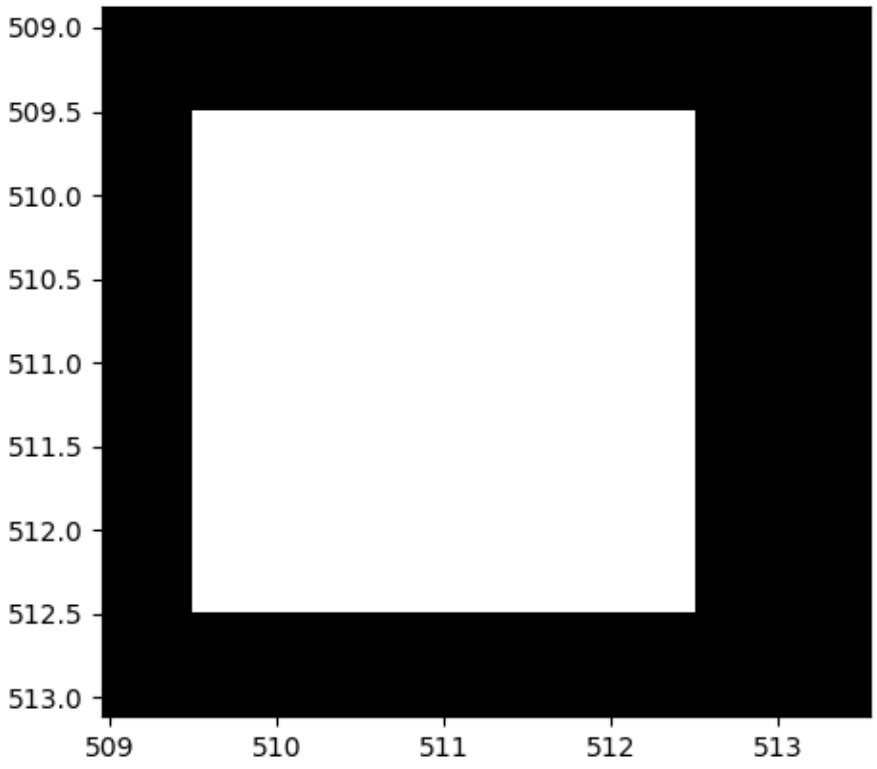


Question1b:
The images have been zoomed in to show convolution.

This is the original image. As can be seen there is one-unit impulse at 512,512

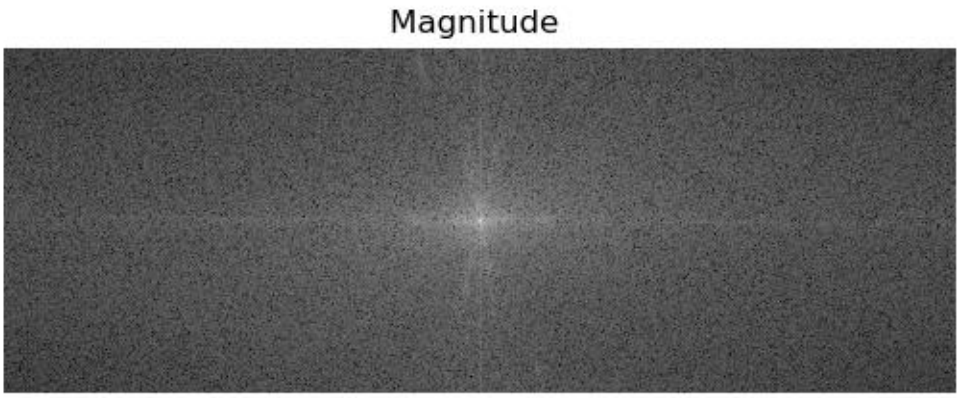


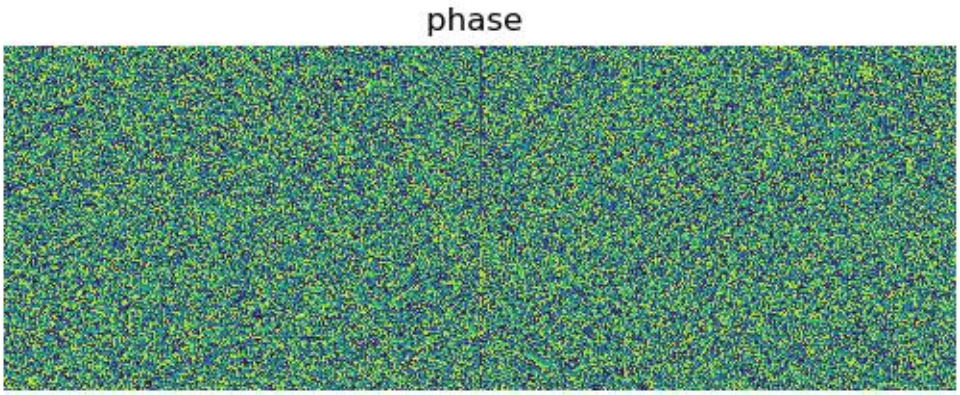
This is the result after convolution after applying box filter:



As can be seen, the unit impulse has spread to 3 by 3 area around it thus our function is indeed performing convolution.

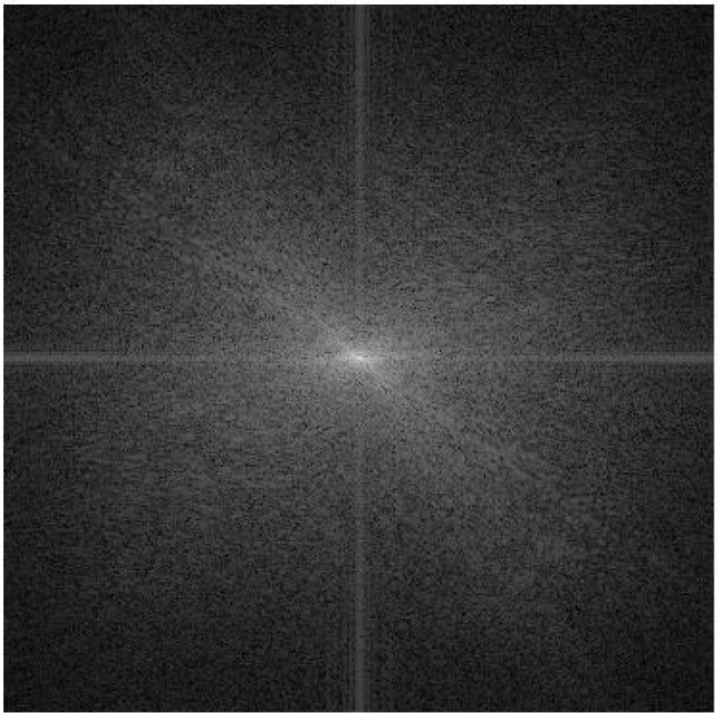
Q2
For wolves:



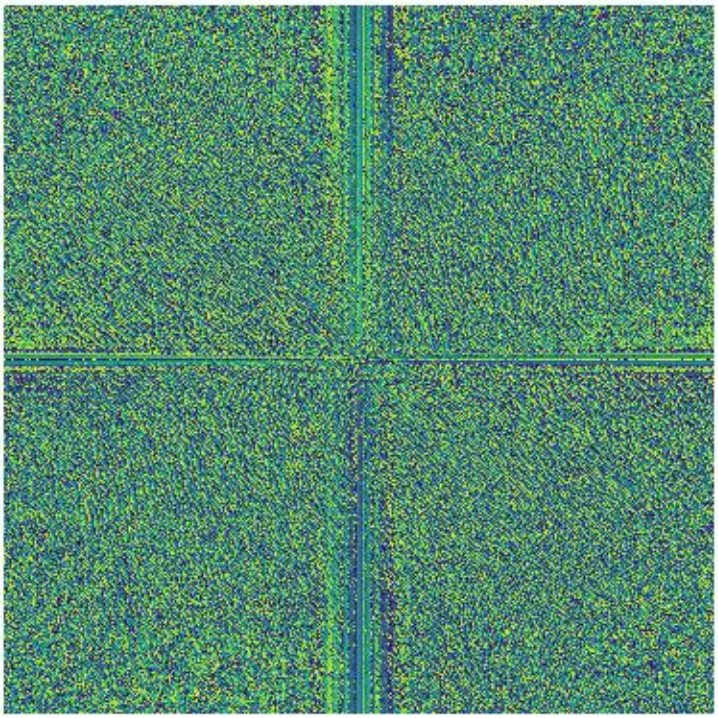


For lenna:

Magnitude



phase



b)

Input Image

Inverse

Question 1 Code:

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


img = cv2.imread('C:/Users/Shubham/Desktop/Fall19/imagin 558/ECE558-
HW01/ECE558-HW01/wolves.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


b,g,r = cv2.split(img)
def loop(kernel,result):
    size0 = 1
    size1 = 1
    resultnew = np.zeros(result.shape)

    if kernel.shape[0] == 1 or kernel.shape[1] == 1 :
        if kernel.shape[0] ==1:
            size0 = int((kernel.shape[0]-1)/2)
            for c1 in range(1,result.shape[0]-1):
                for c2 in range(1,result.shape[1]-1):

                    if np.sum(np.multiply(kernel,result[c1,c2:c2+2])) <0:
                        resultnew[c1,c2] = 0
                    elif np.sum(np.multiply(kernel,result[c1,c2:c2+2])) >255:
                        resultnew[c1,c2] = 255
                    else:
                        resultnew[c1,c2] =
np.sum(np.multiply(kernel,result[c1,c2:c2+2]))
        if kernel.shape[1] ==1:
            size1 = int((kernel.shape[1]-1)/2)
            for c1 in range(1,result.shape[0]-1):
                for c2 in range(1,result.shape[1]-1):
                    x = np.array([result[c1:c1+2,c2]]).transpose()
                    if np.sum(np.multiply(kernel,x)) <0:
                        resultnew[c1,c2] = 0
                    elif np.sum(np.multiply(kernel,x)) >255:
                        resultnew[c1,c2] = 255
                    else:
                        resultnew[c1,c2] = np.sum(np.multiply(kernel,x))
    elif kernel.shape[0]%2 == 0 and kernel.shape[1]%2 == 0 :
        size0=1
        size1=1
        for c1 in range(1,result.shape[0]-1):
            for c2 in range(1,result.shape[1]-1):
                if np.sum(np.multiply(kernel,result[c1:c1+2,c2:c2+2])) <0:
                    resultnew[c1,c2] = 0
                elif np.sum(np.multiply(kernel,result[c1:c1+2,c2:c2+2]))
>255:
                    resultnew[c1,c2] = 255
                else:
                    resultnew[c1,c2] =
np.sum(np.multiply(kernel,result[c1:c1+2,c2:c2+2]))
    else:
```

```
        size0 = int((kernel.shape[0]-1)/2)
        size1 = int((kernel.shape[1]-1)/2)
        for c1 in range(size0,result.shape[0]-size0):
            for c2 in range(size1,result.shape[1]-size1):

                if np.sum(np.multiply(kernel,result[c1-size0:c1+size0+1,c2-
size1:c2+size1+1])) <0:
                    resultnew[c1,c2] = 0
                elif np.sum(np.multiply(kernel,result[c1-size0:c1+size0+1,c2-
size1:c2+size1+1])) >255:
                    resultnew[c1,c2] = 255
                else:
                    resultnew[c1,c2] = np.sum(np.multiply(kernel,result[c1-
size0:c1+size0+1,c2-size1:c2+size1+1]))


    return resultnew.astype(np.uint8)
def padding(image,kernel,paddingtype):
    size0 = 1
    size1 = 1
    if kernel.shape[0]%2 >=1 and kernel.shape[1]%2>= 1:
        size0 = int((kernel.shape[0]-1)/2)
        size1 = int((kernel.shape[1]-1)/2)
    if paddingtype == 'zero':
        result = np.zeros((image.shape[0]+2*size0,image.shape[1]+2*size1))
        result[size0:image.shape[0]+size0,size1:image.shape[1]+size1] +=
image


    if paddingtype == 'circular': #wraparound
        result = np.zeros((image.shape[0]+2*size0,image.shape[1]+2*size1))
        result[size0:image.shape[0]+size0,size1:image.shape[1]+size1] +=
image

        for i in range(size0):
            result[size0-1-i,size1:image.shape[1]+size1] = image[-(1+i)]
            result[image.shape[0]+size0+i,size1:image.shape[1]+size1] =
image[i]
            result[size0:image.shape[0]+size0,size1-1-i] =
np.transpose(image)[-(1+i)]
            result[size0:image.shape[0]+size0,image.shape[1]+size1+i] =
np.transpose(image)[i]

        result[0:size0,0:size1] = image[image.shape[0]-
size0:image.shape[0],image.shape[1]-size1:image.shape[1]]
        result[0:size0,result.shape[1]-size1:result.shape[1]] =
image[image.shape[0]-size0:image.shape[0],0:size1]
        result[result.shape[0]-size0:result.shape[0],0:size1] =
image[0:size0,image.shape[1]-size1:image.shape[1]]
        result[result.shape[0]-size0:result.shape[0],result.shape[1]-
size1:result.shape[1]] = image[0:size0,0:size1]

    if paddingtype == 'replicate': #copyedge
        result = np.zeros((image.shape[0]+2*size0,image.shape[1]+2*size1))
        result[size0:image.shape[0]+size0,size1:image.shape[1]+size1] +=
image
```

```
        for i in range(size0,image.shape[0]+size0):
            result[i,0:size1] = image[i-size0][0]
            result[i,result.shape[1]-size1:result.shape[1]] = image[i-
size0][image.shape[1]-1]
        for i in range(size1,image.shape[1]+size1):
            result[0:size1,i] = image[0][i-size0]
            result[result.shape[0]-size1:result.shape[1],i] =
image[image.shape[0]-1][i-size0]

        result[0:size0,0:size1] = image[0][0]
        result[0:size0,result.shape[1]-size0:result.shape[1]] =
image[0,image.shape[1]-1]
        result[result.shape[0]-size0:result.shape[0],0:size1] =
image[image.shape[0]-1,0]
        result[result.shape[0]-size0:result.shape[0],result.shape[1]-
size0:result.shape[1]] = image[image.shape[0]-1,image.shape[1]-1]


    if paddingtype == 'symmetric': #reflect across edge

        result = np.zeros((image.shape[0]+2*size0,image.shape[1]+2*size1))
        result[size0:image.shape[0]+size0,size1:image.shape[1]+size1] +=
image

        result[0:size0,size1:result.shape[1]-size1] = np.flip(image[0:size1],
axis=0)
        result[result.shape[0]-size0:result.shape[0],size1:result.shape[1]-
size1] = np.flip(image[image.shape[0]-size1:image.shape[0]], axis=0)
        result[size1:result.shape[0]-size0,0:size0] =
np.flip(image[:,0:size1], axis=1)
        result[size1:result.shape[0]-size0,result.shape[1]-
size1:result.shape[1]] = np.flip(image[:,image.shape[0]-
size1:image.shape[0]], axis=1)

        result[0:size0,0:size1] = np.flip(image[0:size0,0:size1])
        result[0:size0,result.shape[1]-size0:result.shape[1]] =
np.flip(image[0:size0,image.shape[1]-size1:image.shape[1]])
        result[result.shape[0]-size0:result.shape[0],0:size1] =
np.flip(image[image.shape[0]-size0:image.shape[0],0:size1])
        result[result.shape[0]-size0:result.shape[0],result.shape[1]-
size0:result.shape[1]] = np.flip(image[image.shape[0]-
size0:image.shape[0],image.shape[1]-size1:image.shape[1]])

    return result.astype(np.uint8)
def conv2(image,kernel,paddingtype):
    size0 = int((kernel.shape[0]-1)/2)

    size1 = 1
    if len(kernel.shape) != 1:
        size1 = int((kernel.shape[1]-1)/2)

    #print(size)
    if len(image.shape) ==2:
        result = padding(image,kernel,paddingtype)

        #cv2.waitKey(0)
        result = loop(kernel,result)
```

```
        if kernel.shape[0]%2 >=1 and kernel.shape[1]%2>= 1:

            convimage =
result[size0:image.shape[0]+size0,size1:image.shape[1]+size1]
        else:
            convimage = result[1:image.shape[0]+1,1:image.shape[1]+1]
        return convimage
    if len(image.shape) ==3:
        b,g,r = cv2.split(image)
        resultb = padding(b,kernel,paddingtype)
        resultb = loop(kernel,resultb)
        resultg = padding(g,kernel,paddingtype)
        resultg = loop(kernel,resultg)
        resultr = padding(r,kernel,paddingtype)
        resultr = loop(kernel,resultr)
        convimageb =
resultb[size0:image.shape[0]+size0,size1:image.shape[1]+size1]
        convimageg =
resultg[size0:image.shape[0]+size0,size1:image.shape[1]+size1]
        convimager =
resultr[size0:image.shape[0]+size0,size1:image.shape[1]+size1]
        return cv2.merge((convimageb, convimageg, convimager))

box = np.ones(9).reshape((3,3))/9
dx = np.array([[-1, 1]])
dy = np.transpose(dx)

PrewittMx = np.array([[-1,0, 1],[-1, 0, 1],[-1, 0, 1]])
PrewittMy = np.array([[1,1, 1],[0, 0, 0],[-1, -1, -1]])

SobelMx = np.array([[-1,0, 1],[-2, 0, 2],[-1, 0, 1]])
SobelMy = np.array([[1,2, 1],[0, 0, 0],[-1, -2, -1]])

RobertsMx = np.array([[0,1],[-1, 0]])
RobertsMy = np.array([[1,0],[0,-1]])
impulse = np.zeros((1024,1024))
impulse[511,511] = 255
#cv2.imshow('img', impulse)
#convimpulse = conv2(impulse,box,'zero')
convbox = conv2(gray,box,'zero')
# convboxPrewittMx = conv2(gray,PrewittMx,'zero')
# convboxPrewittMy = conv2(gray,PrewittMy,'zero')
# convboxSobelMx = conv2(gray,SobelMx,'zero')
# convboxSobelMy = conv2(gray,SobelMy,'zero')
# convRobertsMx = conv2(gray,RobertsMx,'zero')
# convRobetsMy = conv2(gray,RobertsMy,'zero')
# convdx = conv2(gray,dx,'zero')
# convdy = conv2(gray,dy,'zero')


plt.figure(1)
plt.imshow(convbox,cmap='gray')
plt.title('Box Filter ')
plt.axis('off')
# plt.figure(2)
# plt.imshow(convboxPrewittMx,cmap='gray')
```

```
# plt.title('Prewitt X ')
# plt.axis('off')
# plt.figure(3)
# plt.imshow(convboxPrewittMy,cmap='gray')
# plt.title('Prewitt Y ')
# plt.axis('off')
# plt.figure(4)
# plt.imshow(convboxSobelMx,cmap='gray')
# plt.title('Sobel x ')
# plt.axis('off')
# plt.figure(5)
# plt.imshow(convboxSobelMy,cmap='gray')
# plt.title('Sobel y ')
# plt.axis('off')
# plt.figure(6)
# plt.imshow(convRobertsMx,cmap='gray')
# plt.title('Roberts X ')
# plt.axis('off')
# plt.figure(7)
# plt.imshow(convRobetsMy,cmap='gray')
# plt.title('Roberts Y ')
# plt.axis('off')
# plt.figure(8)
# plt.imshow(convdx,cmap='gray')
# plt.title('First order derivative filter x ')
# plt.axis('off')
# plt.figure(9)
# plt.imshow(convdy,cmap='gray')
# plt.title('First order derivative filter y ')
# plt.axis('off')
plt.show()

# plt.figure(1)
# plt.imshow(impulse, cmap = 'gray')
# plt.figure(2)
# plt.imshow(convimpulse, cmap = 'gray')
# plt.show()
#cv2.imshow('img1',convimpulse)
#cv2.waitKey(0)
#img1 = conv2(gray,d,'zero')
#cv2.imshow('image', gray)
#cv2.imshow('image1', img1)
#print(img1.shape)
#cv2.waitKey(0)
#cv2.destroyAllWindows()
# k = np.arange(25).reshape((5,5))/25
# #print(k)
# eg = np.arange(9).reshape((3,3))
# print('matrix',eg)
# print('result',conv2(eg,dy,'zero'))
```

Question2 Code:

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

import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack
import cv2
import math
path = 'C:/Users/Shubham/Desktop/Fall19/imagin 558/Project02/'
img = cv2.imread('C:/Users/Shubham/Desktop/Fall19/imagin
558/Project02/wolves.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
def scale(image):
    max = np.max(image)
    min = np.min(image)
    L = 2
    resultnew = np.zeros(image.shape)
    for c1 in range(0,image.shape[0]):
        for c2 in range(0,image.shape[1]):
            resultnew[c1][c2] = ((image[c1][c2]-min)/(max-min))*(L-1)
    return resultnew
grayn = scale(gray)
def DFT2(image):
    oned = np.fft.fft(image)
    return np.transpose(np.fft.fft(oned.transpose()))
fshift = np.fft.fftshift(twod)
magnitude = 50*np.log(1+np.abs(fshift))
twod = DFT2(grayn)
def IDFT2(input):
    real = input.real
    imag = -1*input.imag
    Forwardnew = real+1j*imag
    Inverse = DFT2(Forwardnew)
    Inverse = Inverse/808500
    InverseReal = Inverse.real
    Inverseimag = -1*Inverse.imag
    Inversenew =  InverseReal+1j*Inverseimag
    return Inversenew
Inverses = IDFT2(twod)

plt.figure(1)
plt.imshow(grayn, cmap = 'gray')
plt.title('Input Image')
plt.axis('off')
plt.savefig(path+'input')
plt.figure(2)
plt.imshow(Inverses.real, cmap = 'gray')
plt.axis('off')
plt.title('Inverse')
plt.savefig(path+'Inverse')

plt.figure(3)
plt.imshow(magnitude, cmap = 'gray')
```

```
plt.title('Magnitude ')
plt.axis('off')
plt.savefig(path+'magnitudelena')
plt.figure(4)
phase = np.angle(fshift,deg=True)
plt.title('phase ')
plt.axis('off')
plt.imshow(phase)
plt.savefig(path+'phaselena')
#plt.show()
```