# ECE 558 – Digital Imaging Systems

## Project 2

Submitted by :

Surya Dutta (NCSU ID: 200481187)

Problem 1: 2-Dimensional Convolution

Part (a) –

Screenshot of the code used –

project2_q1_a.py

```python
import cv2
import numpy as np

# Read the input image and convert it to grayscale
img = cv2.imread('Lena.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imwrite('grayimg.png', img)

# Computing 2D convolution of the image with the kernel
def conv2d(image,kernel,padding):

    # Flip the kernel
    kernel = np.flipud(np.fliplr(kernel))
    # Initialize the output and padding arrays
    output = np.zeros(image.shape)
    rows = image.shape[0] + (kernel.shape[0]-1)
    cols = image.shape[1] + (kernel.shape[1]-1)
    pad = np.zeros((rows,cols))

    #Padding
    if padding == 'zero':
        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                pad[i+ int((kernel.shape[0]-1)/2),j+ int((kernel.shape[1]-1)/2)] = image[i,j]

    if padding == 'wraparound':
        pad[0,0] = image[0,0]
        pad[pad.shape[0]-1,0] = image[image.shape[0]-1,0]
        pad[0,pad.shape[1]-1] = image[0,image.shape[1]-1]
        pad[pad.shape[0]-1,pad.shape[1]-1] = image[image.shape[0]-1,image.shape[1]-1]
        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                pad[i+ int((kernel.shape[0]-1)/2),j+ int((kernel.shape[1]-1)/2)] = image[i,j]
        for k in range(1,pad.shape[0]-1):
            pad[k,0] = image[k-1,image.shape[1]-1]
            pad[k,pad.shape[1]-1] = image[k-1,0]
        for l in range(1,pad.shape[1]-1):
            pad[0,l] = image[image.shape[0]-1,l-1]
            pad[pad.shape[0]-1,l] = image[0,l-1]

    if padding == 'copyedge':
        pad[0,0] = image[0,0]
        pad[pad.shape[0]-1,0] = image[image.shape[0]-1,0]
        pad[0,pad.shape[1]-1] = image[0,image.shape[1]-1]
        pad[pad.shape[0]-1,pad.shape[1]-1] = image[image.shape[0]-1,image.shape[1]-1]
```

project2_q1_a.py

```
            for l in range(1,pad.shape[1]-1):
                pad[0,l] = image[image.shape[0]-1,l-1]
                pad[pad.shape[0]-1,l] = image[0,l-1]

        if padding == 'copyedge':
            pad[0,0] = image[0,0]
            pad[pad.shape[0]-1,0] = image[image.shape[0]-1,0]
            pad[0,pad.shape[1]-1] = image[0,image.shape[1]-1]
            pad[pad.shape[0]-1,pad.shape[1]-1] = image[image.shape[0]-1,image.shape[1]-1]
            for i in range(image.shape[0]):
                for j in range(image.shape[1]):
                    pad[i+ int((kernel.shape[0]-1)/2),j+ int((kernel.shape[1]-1)/2)] = image[i,j]
            for k in range(1,pad.shape[0]-1):
                pad[k,0] = image[k-1,0]
                pad[k,pad.shape[1]-1] = image[k-1,image.shape[1]-1]
            for l in range(1,pad.shape[1]-1):
                pad[0,l] = image[0,l-1]
                pad[pad.shape[0]-1,l] = image[image.shape[0]-1,l-1]

        if padding == 'reflectacrossedge':
            pad[0,0] = image[0,0]
            pad[pad.shape[0]-1,0] = image[image.shape[0]-1,0]
            pad[0,pad.shape[1]-1] = image[0,image.shape[1]-1]
            pad[pad.shape[0]-1,pad.shape[1]-1] = image[image.shape[0]-1,image.shape[1]-1]
            for i in range(image.shape[0]):
                for j in range(image.shape[1]):
                    pad[i+ int((kernel.shape[0]-1)/2),j+ int((kernel.shape[1]-1)/2)] = image[i,j]
            for k in range(1,pad.shape[0]-1):
                pad[k,0] = image[k-1,1]
                pad[k,pad.shape[1]-1] = image[k-1,image.shape[1]-2]
            for l in range(1,pad.shape[1]-1):
                pad[0,l] = image[1,l-1]
                pad[pad.shape[0]-1,l] = image[image.shape[0]-2,l-1]

        cv2.imwrite("paddedimg.png",pad)
        # Compute element-wise multiplication and add the products
        for x in range(image.shape[1]):
            for y in range(image.shape[0]):
                output[y, x]=(kernel * pad[y: y+kernel.shape[0], x: x+kernel.shape[1]]).sum()
        return output,pad

kernel = np.array([[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]])
output,pad = conv2d(img,kernel,'zero')
cv2.imwrite("outputimg.png",output)
```

## Results –

1.lena.png

Input Image

Grayscale Image

Box Filter –

Zero Padding

Wrap Around Padding

Reflect Across Edge Padding

Copy Edge Padding

Simple First Order Derivative Filter –

Type 1 – [[-1,1]]

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

Type 2 – [[-1],[1]]

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

Prewitt Filter –

Mx –

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

My –

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

Sobel Filter –

Mx –

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

My –

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

Roberts Filter –

Mx –

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge Padding

My –

Zero Padding

Wrap Around Padding

Copy Edge Padding

Reflect Across Edge

2.wolves.png : Note - The images for wolves.png (both grayscale and RGB) are present in the results folder, but are not added in the report.

Part (b) –

Screenshot of the code used -

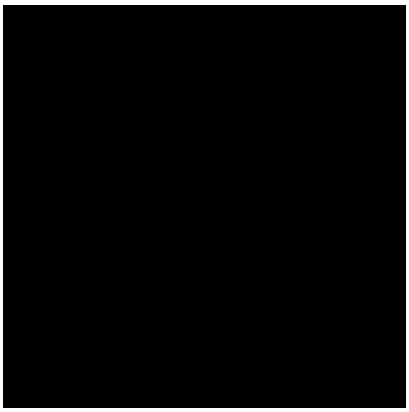
```
# Create gray image which consists of unit impulse at the centre
rows = 1024
cols = 1024
img = signal.unit_impulse((rows,cols),'mid')
cv2.imwrite('inputimg.png', img)
cv2.imshow('Input Image',img)
```

```
kernel = 255*(np.ones([55,55])) # 55*55 matrix with all values equal to 255
output,pad = conv2d(img,kernel,'zero')
cv2.imshow('Output Image', output)
cv2.imwrite("outputimg.png",output)
```

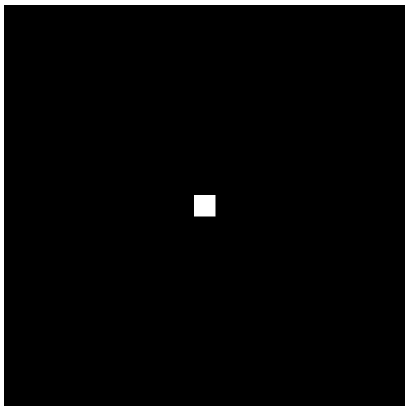
The convolution function is the same function that has been used in part (a).

When any signal is convolved with an impulse response, the resultant is the signal itself. Hence, when the kernel is convolved with the impulse response image, the resultant image will consist of the kernel at the location of the impulse response (i.e. the centre of the image in this case). This is shown in the results below, and hence it is proved that the function is indeed performing convolution.

Results –



Input Image



Output Image

Problem 2 : Implementing and testing the 2-D FFT and its inverse using a built-in 1-D FFT algorithm.

Screenshot of the code used –

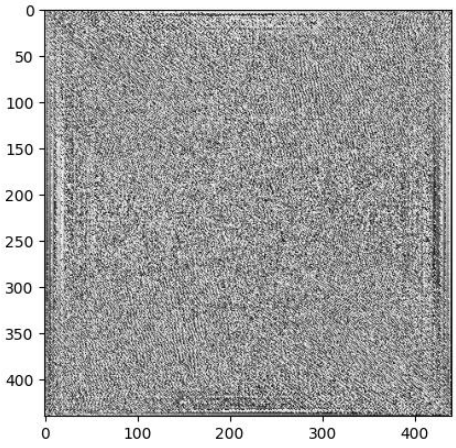
project2_q2.py

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

#Read input image as grayscale
inp = cv2.imread('wolves.png',0)
inp = inp.astype('float32')

#Perform scaling on the input image
rows = inp.shape[0]
cols = inp.shape[1]
rmin = np.amin(inp)
rmax = np.amax(inp)
smin = 0
smax = 1
s = np.zeros((rows,cols))
for i in range(rows):
    for j in range(cols):
        s[i,j] = (((smax-smin)/(rmax-rmin))*(inp[i,j] - rmin)) + smin

#Compute 2D DFT
def DFT2D(img):
    fft = np.fft.fft(img)
    fft = np.transpose(fft)
    fft = np.fft.fft(fft)
    fft = np.transpose(fft)
    return fft

dft_img = DFT2D(s)
angle = np.angle(dft_img)
plt.imshow(np.log(1+abs(dft_img)),cmap='Greys')
plt.savefig('dft_img.png')
plt.imshow(angle,cmap='Greys')
plt.savefig('dft_angle.png')

#Compute 2D IDFT
def IDFT2D(img):
    r = dft_img.shape[0]
    c = dft_img.shape[1]
    d = DFT2D(np.conj(dft_img))
    img = np.real(np.conj(d))
    img = img/(r*c)
    return img

idftimg = IDFT2D(dft_img)
x = np.round(s-idftimg)
plt.imshow(idftimg,cmap='Greys')
plt.savefig('idft_img.png')
cv2.imshow('zero_fig.png',x)
cv2.imwrite('zero_fig.png',x)
```

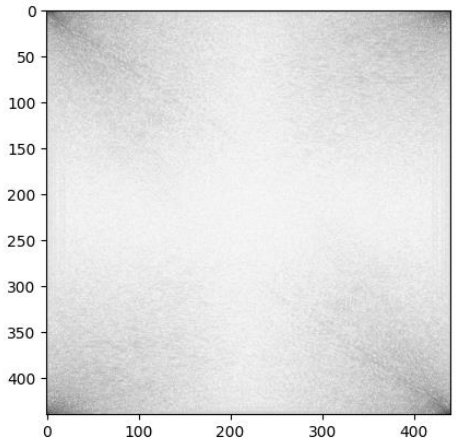
Results –

lena.png –

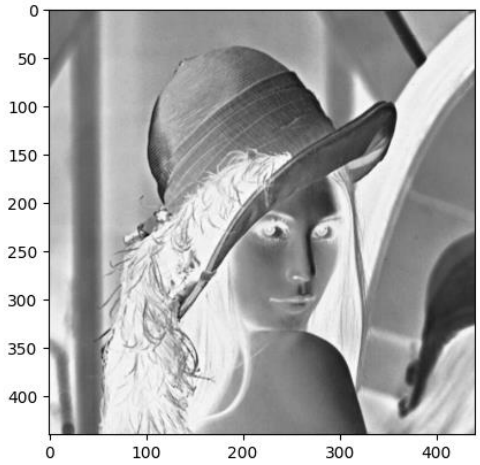
Input Image



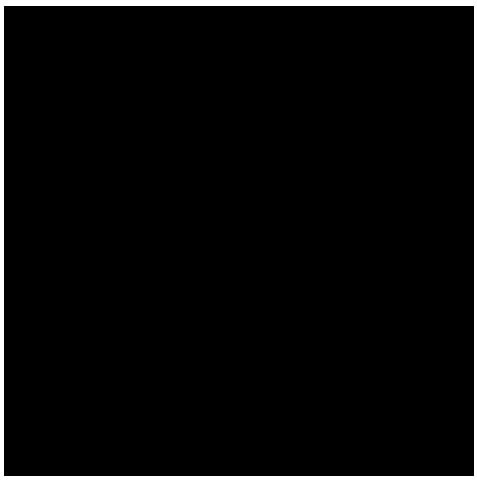

DFT Angle




DFT Image




IDFT Image



Grey Image - IDFT Image = Black Image

*******