Name – Vivek Onkarnath Rathi
ID - 200256752
ECE 558 Project 02

P1.

a.
Solution

Code –

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 11 23:50:36 2019

Code works for asked all filters and images.
The outputs in the report are generated using this code
However, few examples with different filters are shown in the code
to reduce the code size for processing
uncommenting the python code can work in various desired ways

It only uses lena img, to use wolves img, change the imread function value.

@author: Vivek Rathi
"""

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

# convert float to uint8
def ftou8(img):
    img = img.astype(np.uint8)
    return img

# convert uint8 to float
def u8tof(img):
    img = img.astype(np.float32)
    return img

# spread intensities
def spread_linear(img,uplimit):
    s = np.copy(img)
    ma = img.max()
    mi = img.min()
    if len(img.shape) == 3:
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                s[i,j,:] = ((uplimit - 1)/(ma - mi)) * (img[i,j,:] - mi)
        return (ftou8(s))
    else:
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
```

```
            s[i,j] = ((uplimit - 1)/(ma - mi)) * (img[i,j] - mi)
        return (ftou8(s))
#padding function
def pad(img,typepad):
    rows = img.shape[0]
    cols = img.shape[1]
    if len(img.shape) == 3:
        imgpad = np.zeros((rows+2,cols+2,3), dtype = 'float')
    elif len(img.shape) == 2:
        imgpad = np.zeros((rows+2,cols+2), dtype = 'float')
    imgpad[1:rows+1,1:cols+1] = img
    rows = imgpad.shape[0]
    cols = imgpad.shape[1]
    if typepad == "zero":
        return imgpad
    elif typepad == "wrap":
        imgpad[:,0] = imgpad[:,cols-2]
        imgpad[:,cols-1] = imgpad[:,1]
        imgpad[0,:] = imgpad[rows-2,:]
        imgpad[rows-1,:] = imgpad[1,:]
        imgpad[0,0] = imgpad[rows-2,cols-2]
        imgpad[rows-1,0] = imgpad[1,cols-2]
        imgpad[0,cols-1] = imgpad[rows-2,1]
        imgpad[rows-1,cols-1] = imgpad[1,1]
        return imgpad
    elif typepad == "copyedge":
        imgpad[0,:] = imgpad[1,:]
        imgpad[rows-1,:] = imgpad[rows-2,:]
        imgpad[:,0] = imgpad[:,1]
        imgpad[:,cols-1] = imgpad[:,cols-2]
        imgpad[0,0] = imgpad[1,1]
        imgpad[rows-1,0] = imgpad[rows-2,1]
        imgpad[0,cols-1] = imgpad[1,cols-2]
        imgpad[rows-1,cols-1] = imgpad[rows-2,cols-2]
        return imgpad
    elif typepad == "reflect":
        imgpad[0,:] = imgpad[2,:]
        imgpad[rows-1,:] = imgpad[rows-3,:]
        imgpad[:,0] = imgpad[:,2]
        imgpad[:,cols-1] = imgpad[:,cols-3]
        imgpad[0,0] = imgpad[2,2]
        imgpad[rows-1,0] = imgpad[rows-3,2]
        imgpad[0,cols-1] = imgpad[2,cols-3]
        imgpad[rows-1,cols-1] = imgpad[rows-3,cols-3]
        return imgpad

# convolution

'''
```

```
p = 0 -> zero padding
p = 1 -> wrap padding
p = 2 -> copyedge padding
p = 3 -> reflect padding
'''
def conv(img,k,p):
    imgcopy = img.astype(np.float32)
    if p == 0:
        padimg = pad(img,"zero")
    elif p == 1:
        padimg = pad(img,"wrap")
    elif p == 2:
        padimg = pad(img,"copyedge")
    elif p == 3:
        padimg = pad(img,"reflect")

    if len(img.shape) == 3:
        B = np.zeros((k.shape[0],k.shape[1],3))
        r = padimg.shape[0]
        c = padimg.shape[1]
        if k.shape[0] == k.shape[1] == 3:
            for i in range(r-2):
                for j in range(c-2):
                    B[:,:,0] = np.multiply(k,padimg[i:i+3,j:j+3,0])
                    B[:,:,1] = np.multiply(k,padimg[i:i+3,j:j+3,1])
                    B[:,:,2] = np.multiply(k,padimg[i:i+3,j:j+3,2])
                    imgcopy[i,j,0] = np.sum(B[:,:,0])
                    imgcopy[i,j,1] = np.sum(B[:,:,1])
                    imgcopy[i,j,2] = np.sum(B[:,:,2])
            return imgcopy
        elif k.shape[0] == k.shape[1] == 2:
            if k.all == k_rx.all():
                for i in range(r-2):
                    for j in range(c-2):
                        B[:,:,0] = np.multiply(k,padimg[i+1:i+3,j:j+2,0])
                        B[:,:,1] = np.multiply(k,padimg[i+1:i+3,j:j+2,1])
                        B[:,:,2] = np.multiply(k,padimg[i+1:i+3,j:j+2,2])
                        imgcopy[i,j,0] = np.sum(B[:,:,0])
                        imgcopy[i,j,1] = np.sum(B[:,:,1])
                        imgcopy[i,j,2] = np.sum(B[:,:,2])
                return imgcopy
            else:
                for i in range(r-2):
                    for j in range(c-2):
                        B[:,:,0] = np.multiply(k,padimg[i+1:i+3,j+1:j+3,0])
                        B[:,:,1] = np.multiply(k,padimg[i+1:i+3,j+1:j+3,1])
                        B[:,:,2] = np.multiply(k,padimg[i+1:i+3,j+1:j+3,2])
                        imgcopy[i,j,0] = np.sum(B[:,:,0])
                        imgcopy[i,j,1] = np.sum(B[:,:,1])
```

```
                imgcopy[i,j,2] = np.sum(B[:,:,2])
            return imgcopy

        elif k.shape[0] < k.shape[1]:
            for i in range(r-2):
                for j in range(c-2):
                    B[:,:,0] = np.multiply(k,padimg[i+1:i+2,j+1:j+3,0])
                    B[:,:,1] = np.multiply(k,padimg[i+1:i+2,j+1:j+3,1])
                    B[:,:,2] = np.multiply(k,padimg[i+1:i+2,j+1:j+3,2])
                    imgcopy[i,j,0] = np.sum(B[:,:,0])
                    imgcopy[i,j,1] = np.sum(B[:,:,1])
                    imgcopy[i,j,2] = np.sum(B[:,:,2])
            return imgcopy
        elif k.shape[0] > k.shape[1]:
            for i in range(r-2):
                for j in range(c-2):
                    B[:,:,0] = np.multiply(k,padimg[i+1:i+3,j+1:j+2,0])
                    B[:,:,1] = np.multiply(k,padimg[i+1:i+3,j+1:j+2,1])
                    B[:,:,2] = np.multiply(k,padimg[i+1:i+3,j+1:j+2,2])
                    imgcopy[i,j,0] = np.sum(B[:,:,0])
                    imgcopy[i,j,1] = np.sum(B[:,:,1])
                    imgcopy[i,j,2] = np.sum(B[:,:,2])
            return imgcopy
    elif len(img.shape) == 2:
        B = np.zeros((k.shape[0],k.shape[1]))
        r = padimg.shape[0]
        c = padimg.shape[1]
        if k.shape[0] == k.shape[1] == 3:
            for i in range(r-2):
                for j in range(c-2):
                    B = np.multiply(k,padimg[i:i+3,j:j+3])
                    imgcopy[i,j] = np.sum(B)
            return imgcopy
        elif k.shape[0] == k.shape[1] == 2:
            if k.all() == k_rx.all():
                for i in range(r-2):
                    for j in range(c-2):
                        B = np.multiply(k,padimg[i+1:i+3,j:j+2])
                        imgcopy[i,j] = np.sum(B)
                return imgcopy
            else:
                for i in range(r-2):
                    for j in range(c-2):
                        B = np.multiply(k,padimg[i+1:i+3,j+1:j+3])
                        imgcopy[i,j] = np.sum(B)
                return imgcopy

        elif k.shape[0] < k.shape[1]:
            for i in range(r-2):
```

```
            for j in range(c-2):
                B = np.multiply(k,padimg[i+1:i+2,j+1:j+3])
                imgcopy[i,j] = np.sum(B)
        return imgcopy
    elif k.shape[0] > k.shape[1]:
        for i in range(r-2):
            for j in range(c-2):
                B = np.multiply(k,padimg[i+1:i+3,j+1:j+2])
                imgcopy[i,j] = np.sum(B)
        return imgcopy

'''
Kernels
'''
k_box = (1/9) * np.ones((3,3))
k_x1 = np.matrix(([-1,1]))
k_y1 = np.matrix(([-1],[1]))
k_px = np.matrix(([-1,0,1],[-1,0,1],[-1,0,1]))
k_py = np.matrix(([1,1,1],[0,0,0],[-1,-1,-1]))
k_sx = np.matrix(([-1,0,1],[-2,0,2],[-1,0,1]))
k_sy = np.matrix(([1,2,1],[0,0,0],[-1,-2,-1]))
k_rx = np.matrix(([0,1],[-1,0]))
k_ry = np.matrix(([1,0],[0,-1]))

# Load image
lenaimg = cv2.imread('lena.png',0)
#lenaimg = cv2.imread('lena.png',1)

# Apply Convolution
# Box Filter- Zero Padding
c11 = conv(lenaimg,k_box,0)
s11 = ftou8(c11)
# Fisrt Order X Derivative- Wrap Padding
d11 = conv(lenaimg,k_x1,1)
s12 = spread_linear(d11,256)
# Prewitt X Derivative - Copyedge Padding
e11 = conv(lenaimg,k_px,2)
s13 = spread_linear(e11,256)
# Sobel Y Derivative - Reflect Padding
f11 = conv(lenaimg,k_sy,3)
s14 = spread_linear(f11,256)
# Roberts Y Derivative - Copyedge Padding
g11 = conv(lenaimg,k_ry,2)
s15 = spread_linear(g11,256)



plt.figure(figsize=(10,10))
```

```
plt.subplot(221)
#plt.imshow(cv2.cvtColor(s11, cv2.COLOR_BGR2RGB))
plt.imshow(s11, cmap = 'gray')
plt.xticks([]), plt.yticks([])
plt.title('Box Filter - Pad 0')

plt.subplot(222)
#plt.imshow(cv2.cvtColor(s12, cv2.COLOR_BGR2RGB))
plt.imshow(s12, cmap = 'gray')
plt.xticks([]), plt.yticks([])
plt.title('X Derivative - Wrap')

plt.subplot(223)
plt.imshow(s13, cmap = 'gray')
#plt.imshow(cv2.cvtColor(s13, cv2.COLOR_BGR2RGB))
plt.xticks([]), plt.yticks([])
plt.title('Prewitt X - Copy')

plt.subplot(224)
#plt.imshow(cv2.cvtColor(s14, cv2.COLOR_BGR2RGB))
plt.imshow(s14, cmap = 'gray')
plt.xticks([]), plt.yticks([])
plt.title('Sobel Y - Reflect')
plt.show()



cv2.namedWindow('Original Image', cv2.WINDOW_NORMAL)
cv2.imshow('Original Image',lenaimg)
cv2.waitKey(0)
cv2.destroyAllWindows()

#cv2.namedWindow('Rx Derivative-Pad 0', cv2.WINDOW_NORMAL)
#cv2.imshow('Rx Derivative-Pad 0',c11)
#cv2.namedWindow('Rx Derivative-Pad wrap', cv2.WINDOW_NORMAL)
#cv2.imshow('Rx Derivative-Pad wrap',d11)
#cv2.namedWindow('Rx Derivative-Pad copy', cv2.WINDOW_NORMAL)
#cv2.imshow('Rx Derivative-Pad copy',e11)
#cv2.namedWindow('Rx Derivative-Pad reflect', cv2.WINDOW_NORMAL)
#cv2.imshow('Rx Derivative-Pad reflect',f11)
#cv2.waitKey(0)
#cv2.destroyAllWindows()
```

Output –

1. Greyscale Lena Image

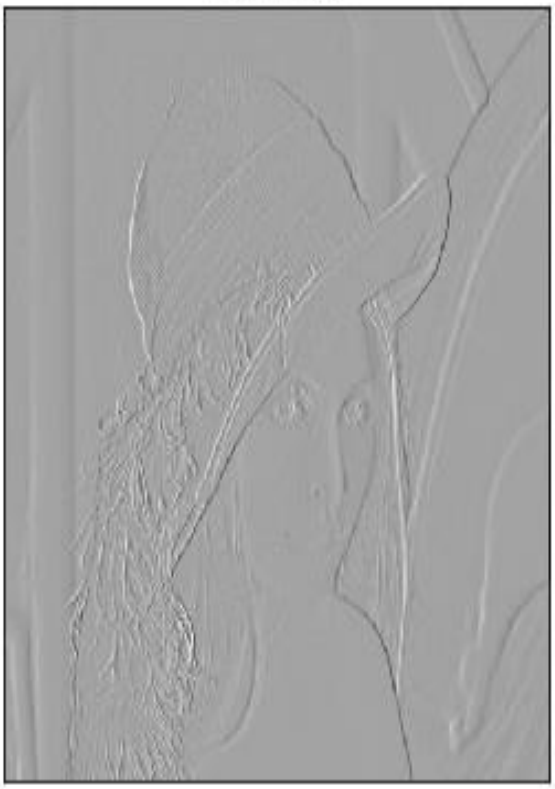
Box - Pad 0

Box - Wrap

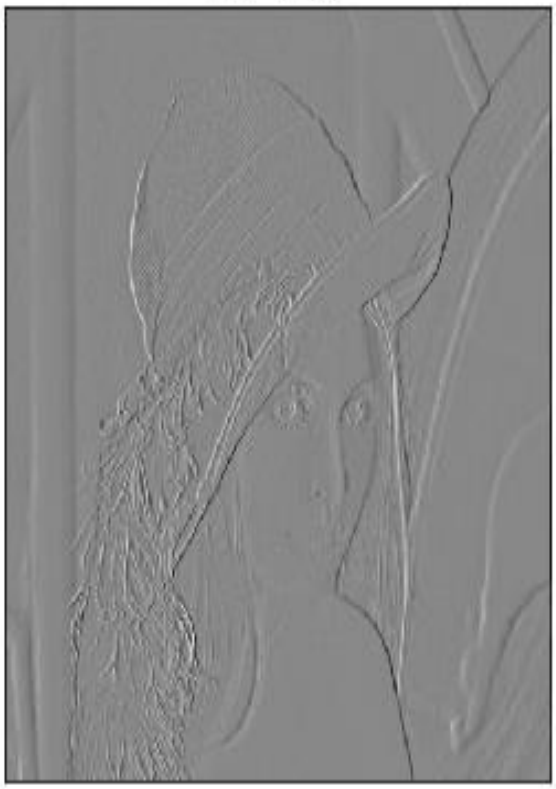
Box - Copy

Box - Reflect

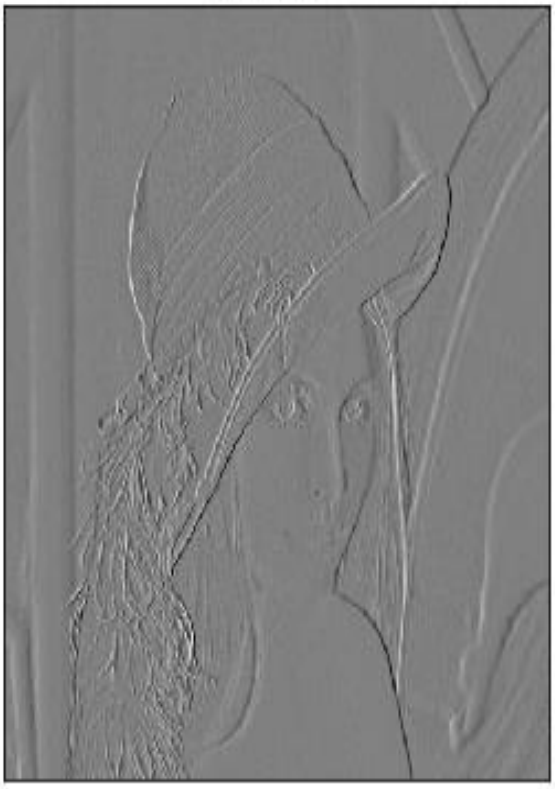
Dx - Pad 0



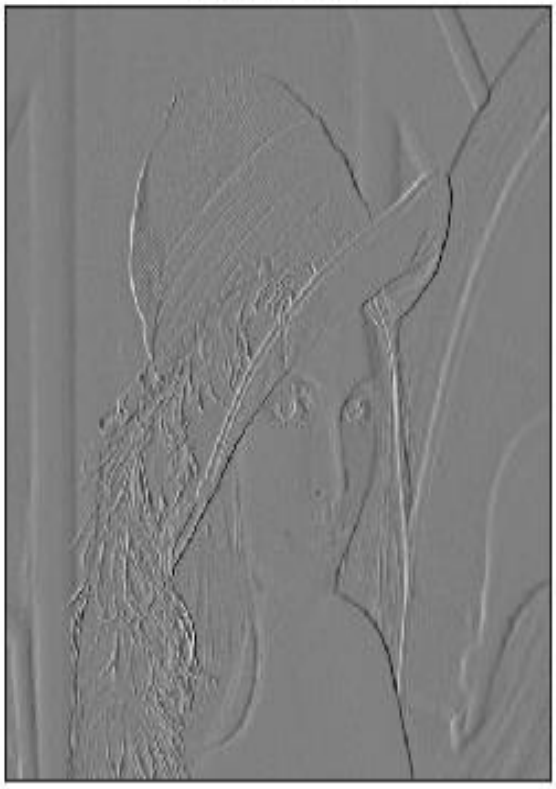
Dx - Wrap



Dx - Copy



Dx - Reflect



Dy - Pad 0

Dy - Wrap

Dy - Copy

Dy - Reflect

Px - Pad 0

Px - Wrap

Px - Copy

Px - Reflect

Py - Pad 0

Py - Wrap

Py - Copy

Py - Reflect

Rx - Pad 0

Rx - Wrap

Rx - Copy

Rx - Reflect

Ry - Pad 0

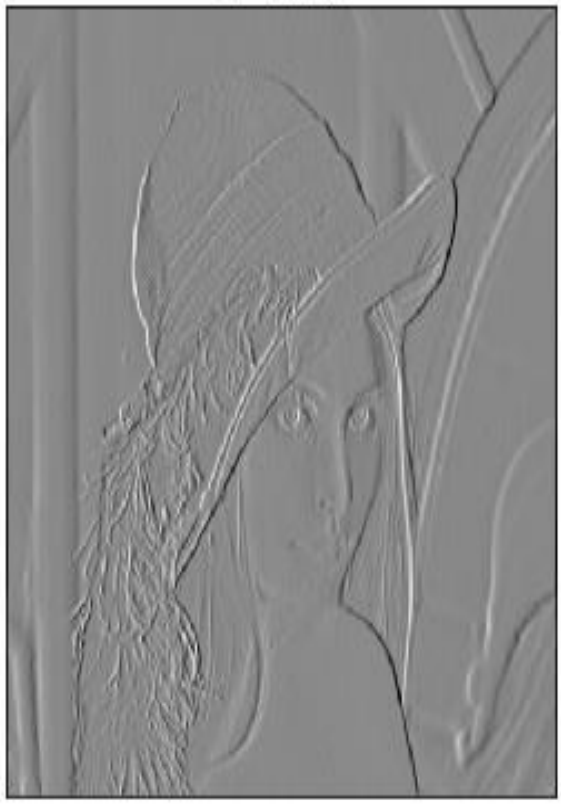
Ry - Wrap

Ry - Copy

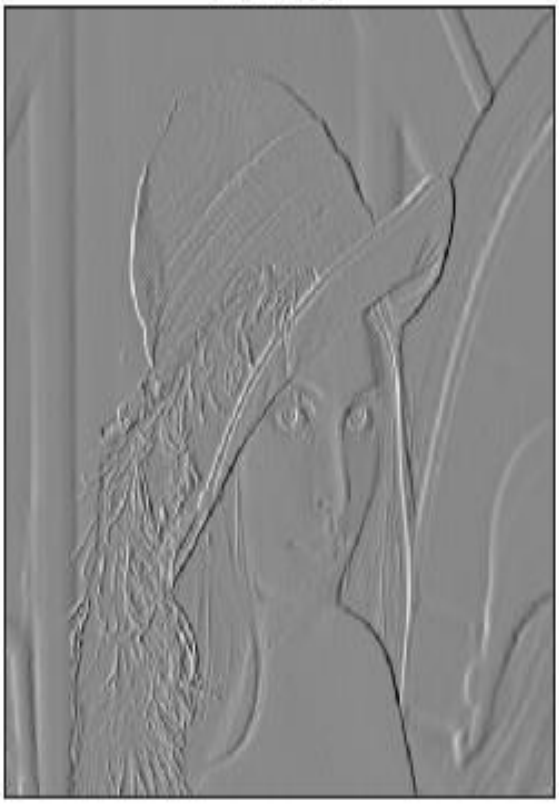
Ry - Reflect

Sx - Pad 0

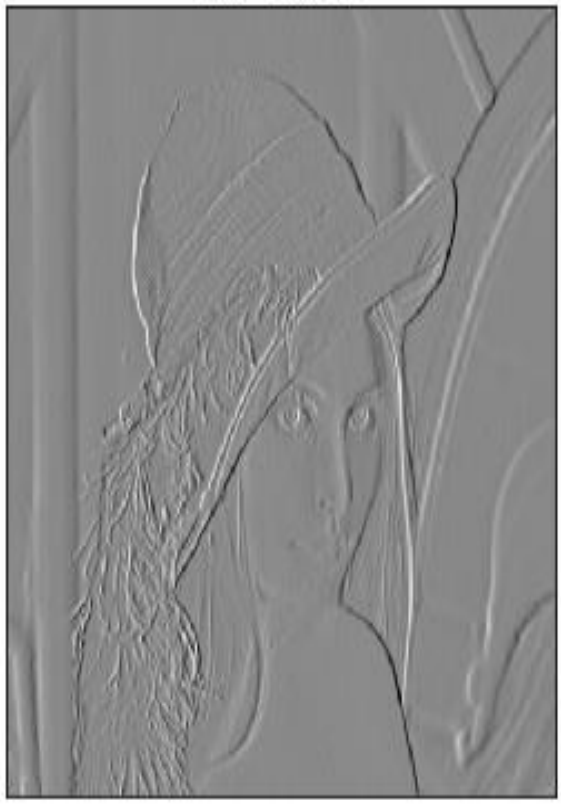
Sx - Wrap



Sx - Copy



Sx - Reflect



Sy - Pad 0

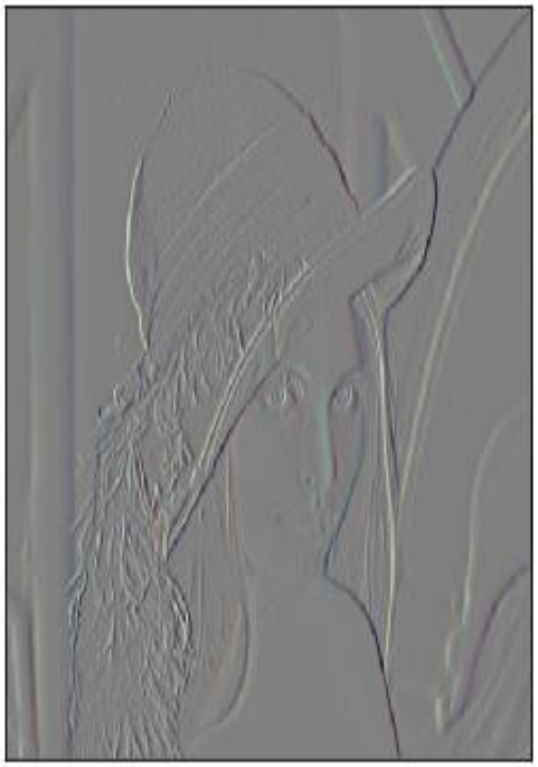
Sy - Wrap

Sy - Copy

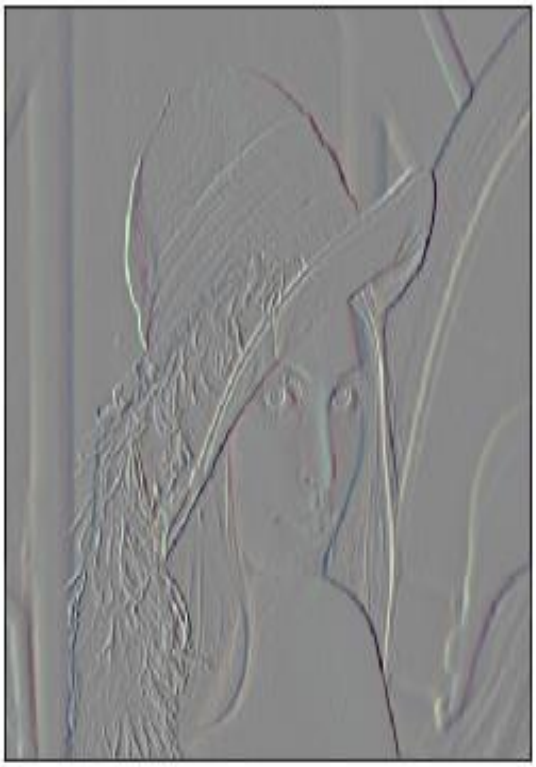
Sy - Reflect

2. Colored Lena Image

Px - Pad 0



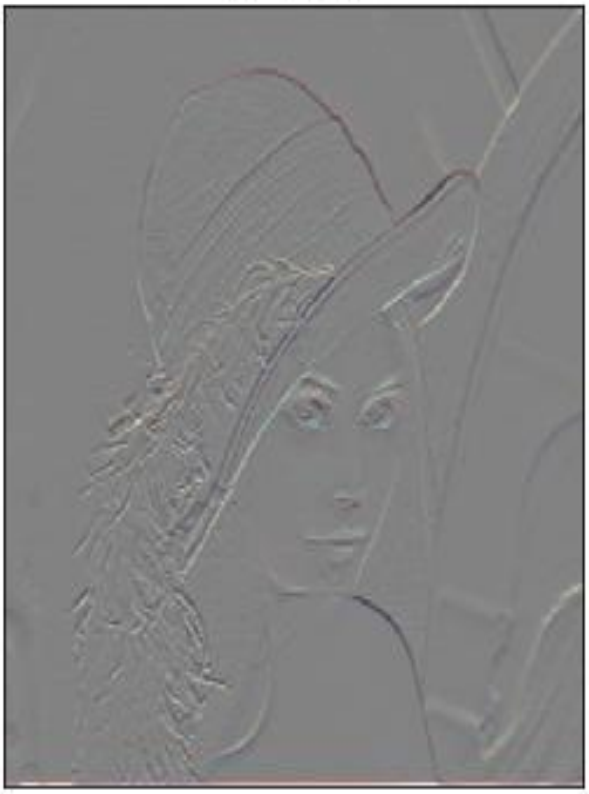
Px - Wrap



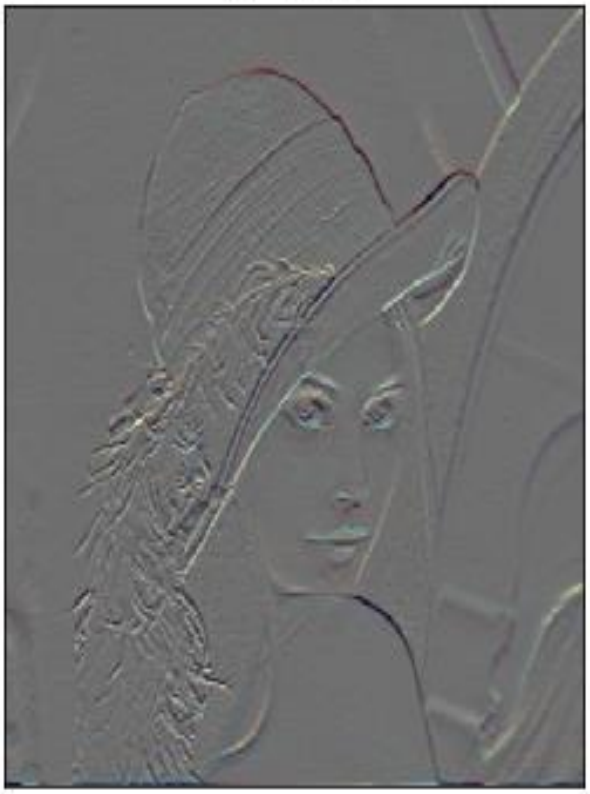
Px - Copy

Px - Reflect

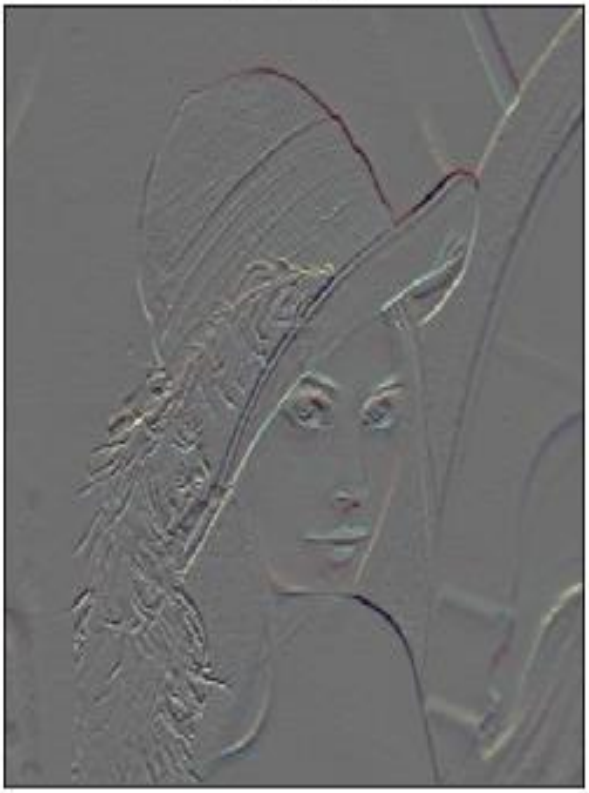
Py - Pad 0



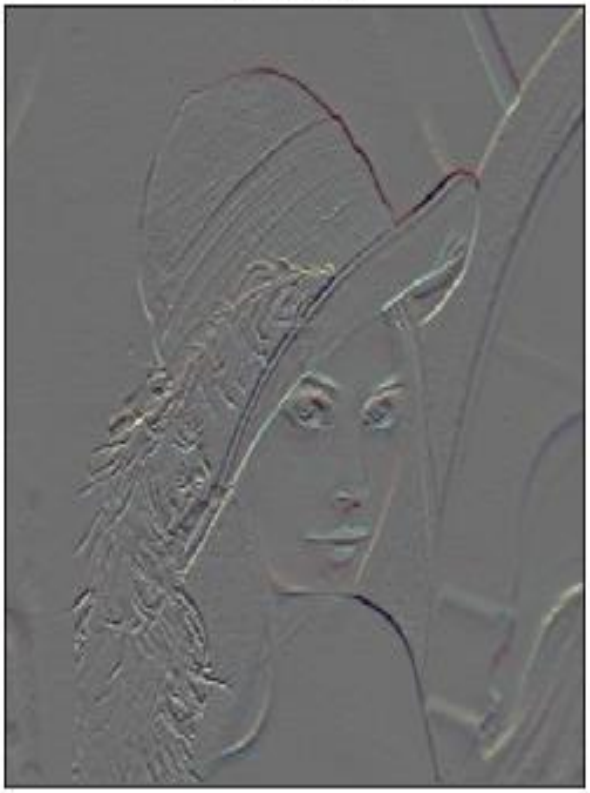
Py - Wrap



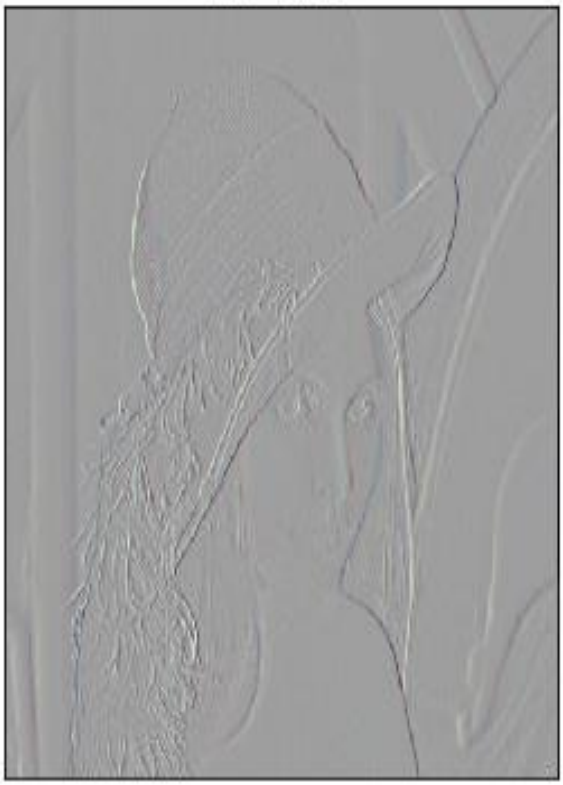
Py - Copy



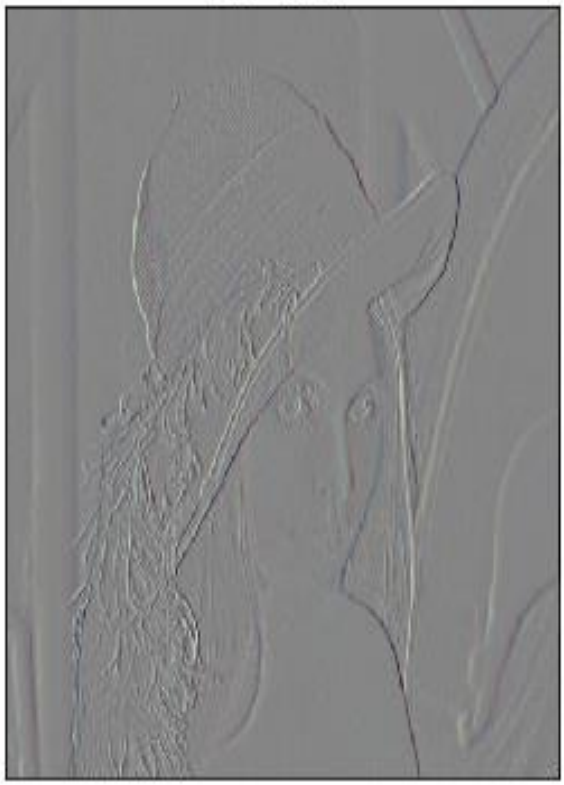
Py - Reflect



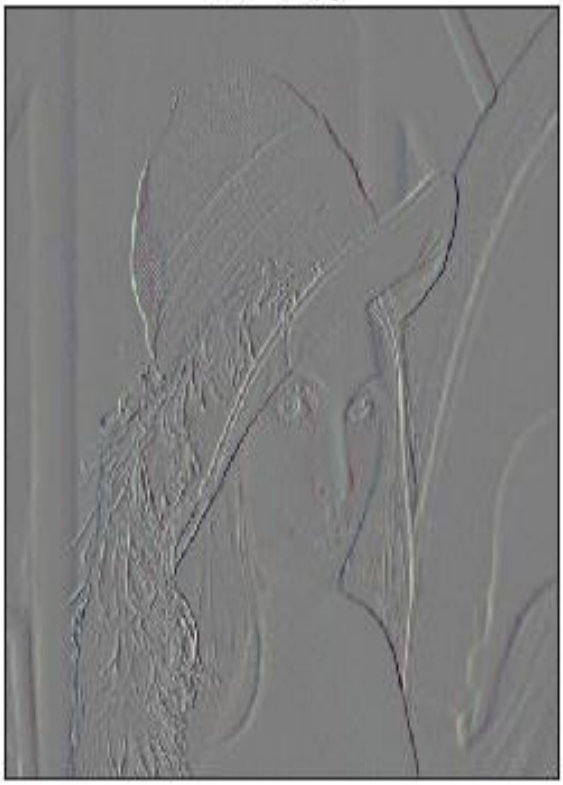
Dx - Pad 0



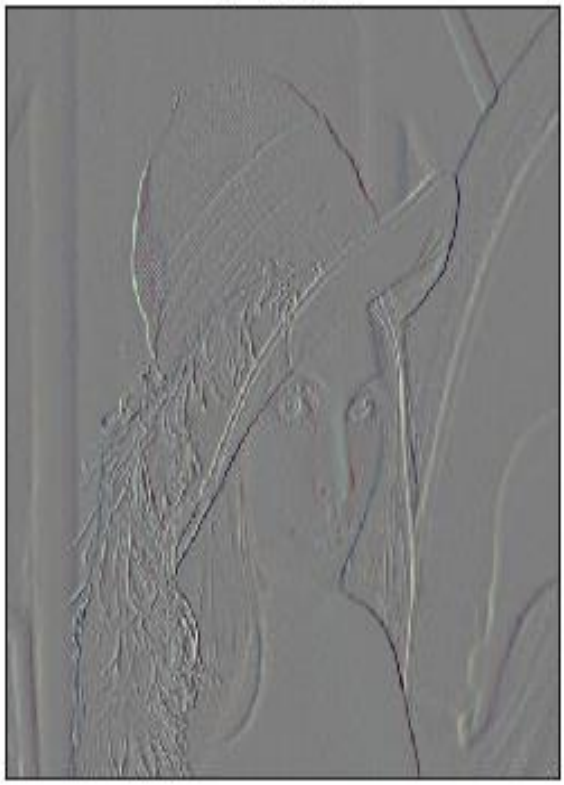
Dx - Wrap



Dx - Copy

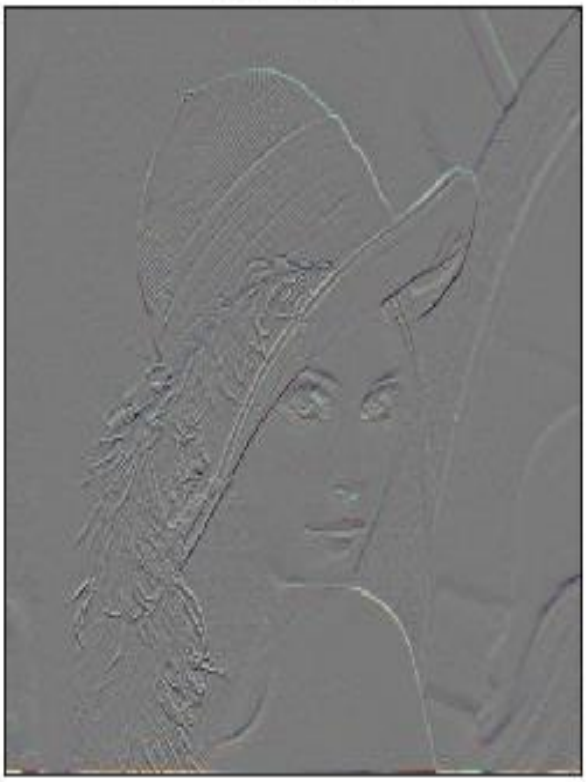


Dx - Reflect

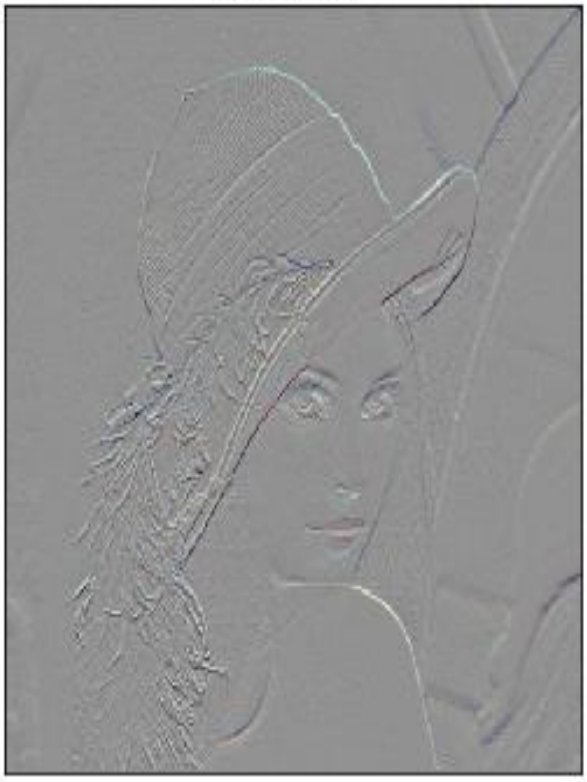


Dy - Pad 0

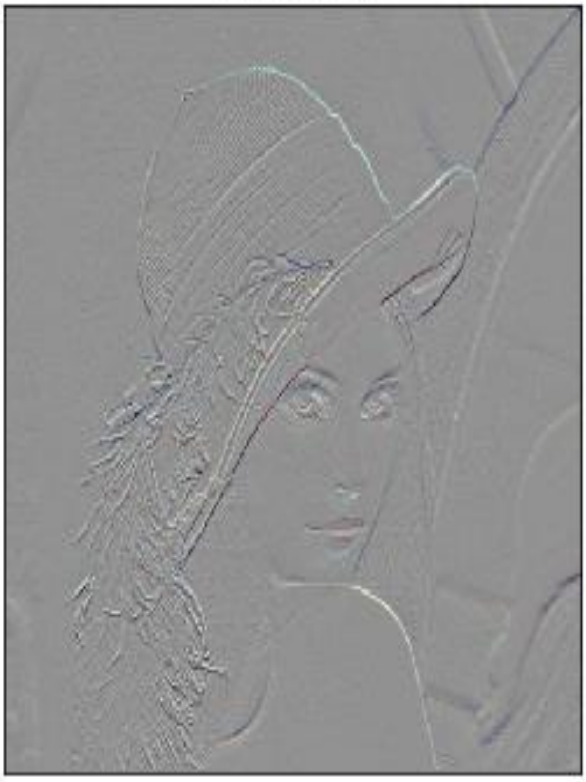
Dy - Wrap



Dy - Copy



Dy - Reflect



Box Filter - Pad 0

Box Filter - Wrap

Box Filter - Copy

Box Filter - Reflect

Ry - Pad 0

Ry - Wrap

Ry - Copy

Ry - Reflect

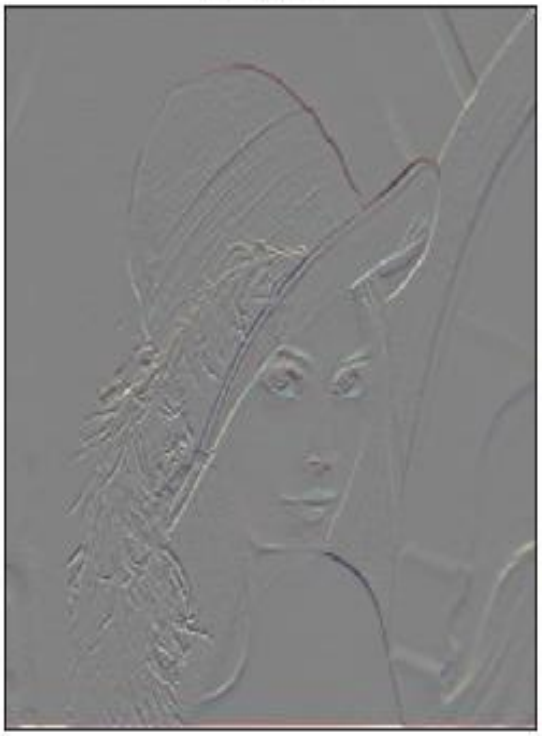
Rx - Pad 0

Rx - Wrap

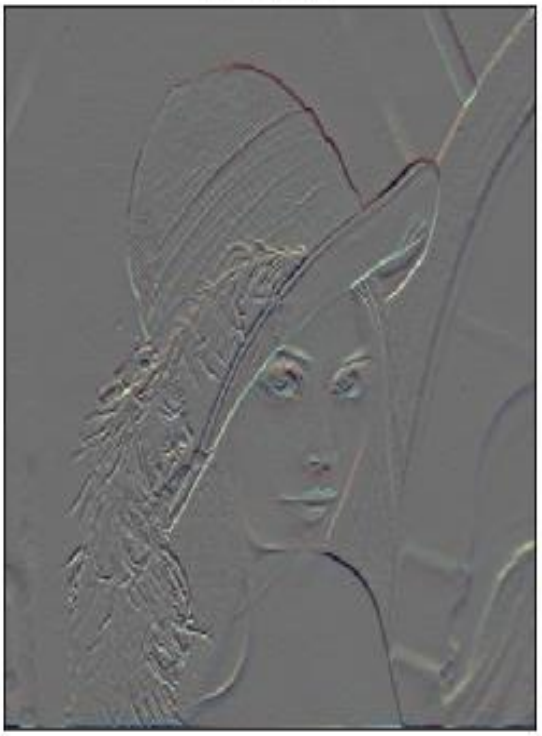
Rx - Copy

Rx - Reflect

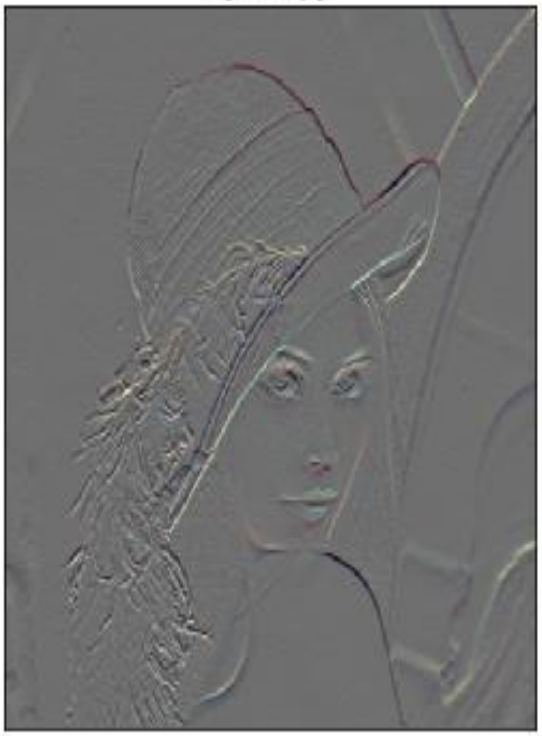
Sy - Pad 0



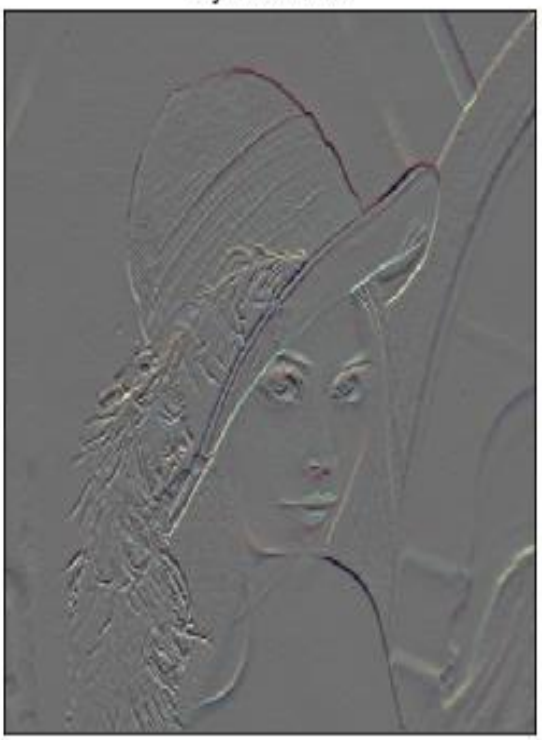
Sy - Wrap



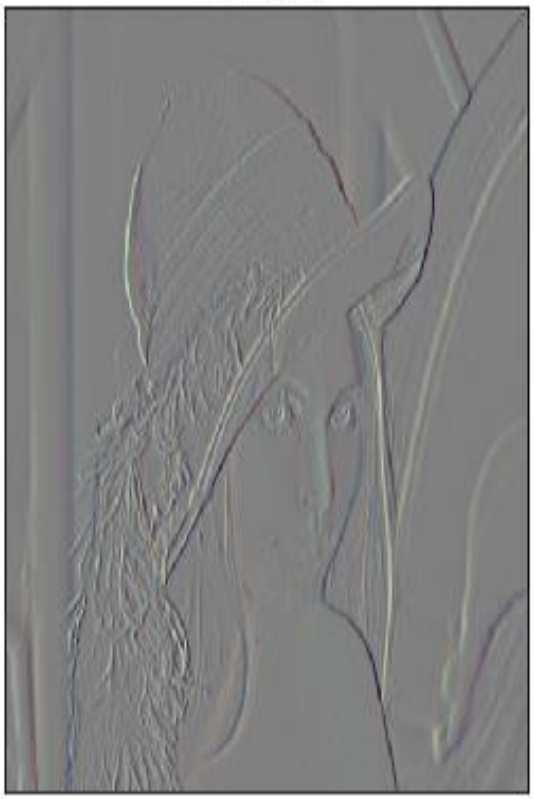
Sy - Copy



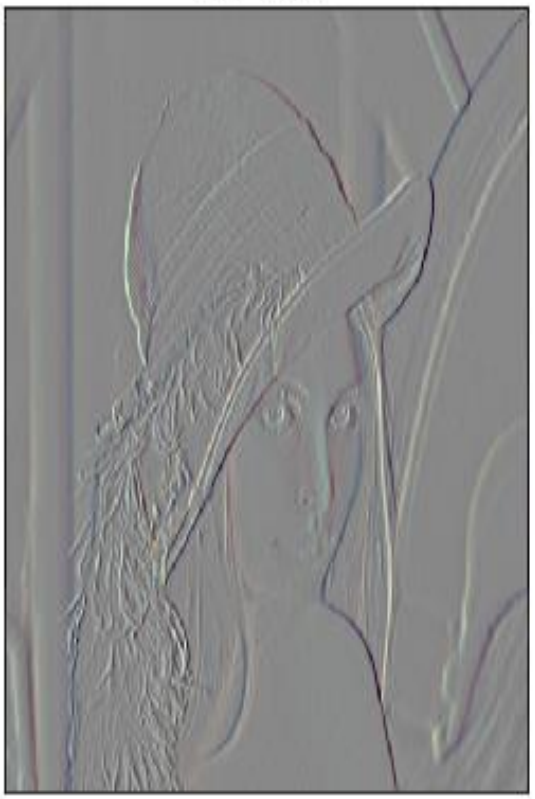
Sy - Reflect



Sx - Pad 0

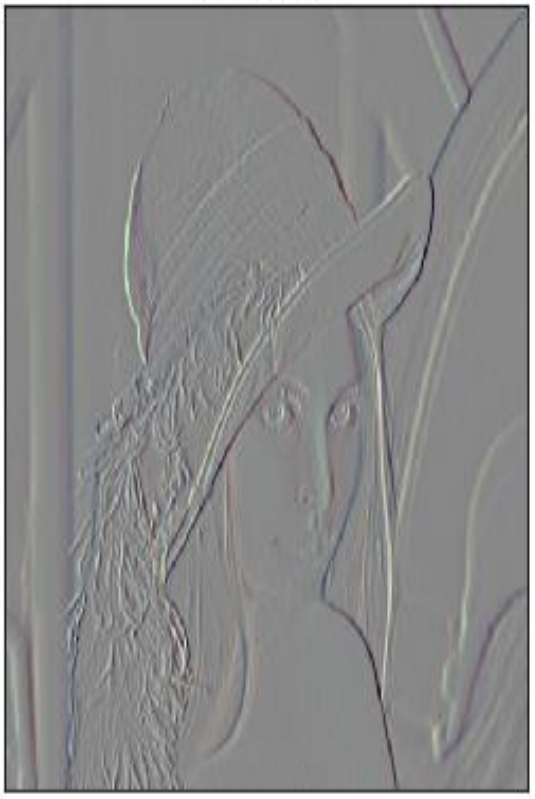


Sx - Wrap



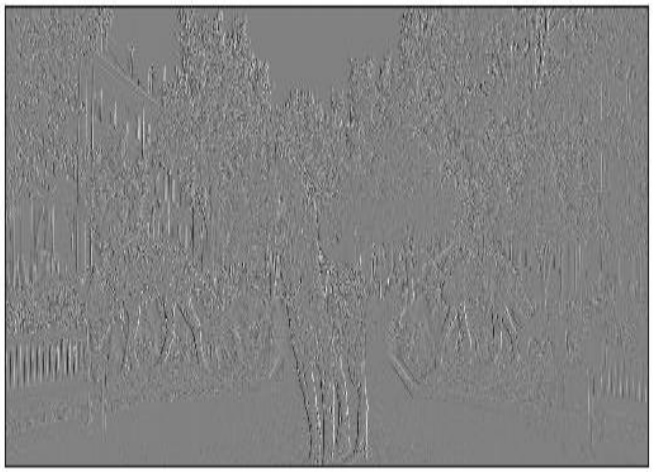
Sx - Copy

Sx - Reflect

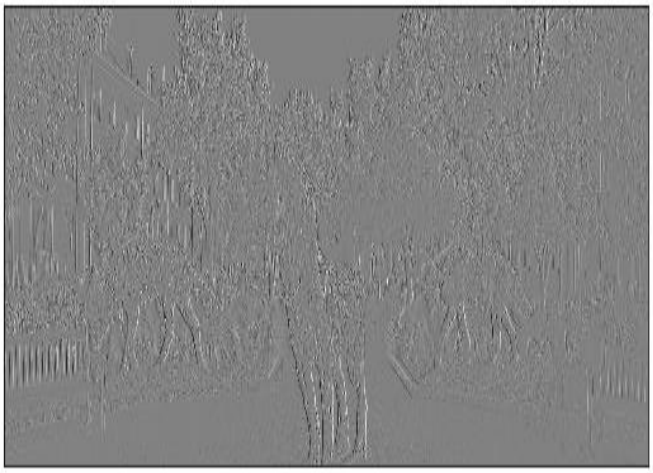


3. Grey Scale Wolves Image

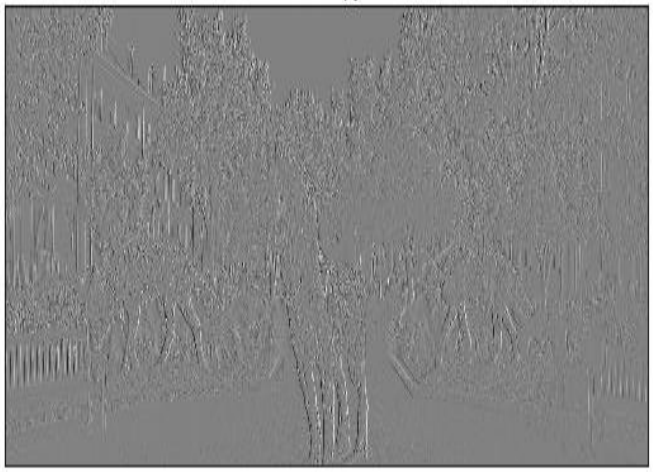
Px - Pad 0



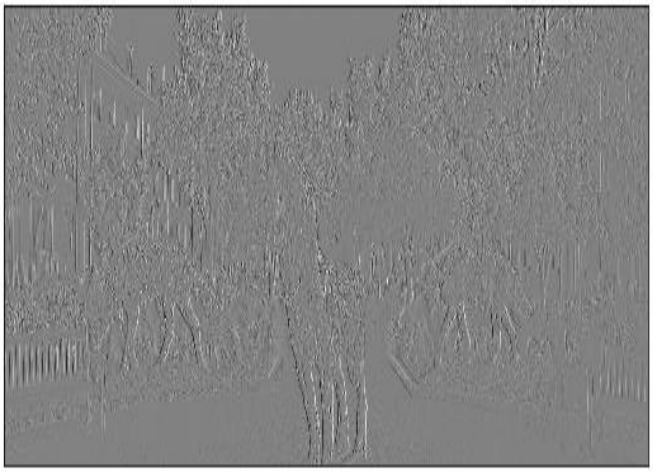
Px - Wrap



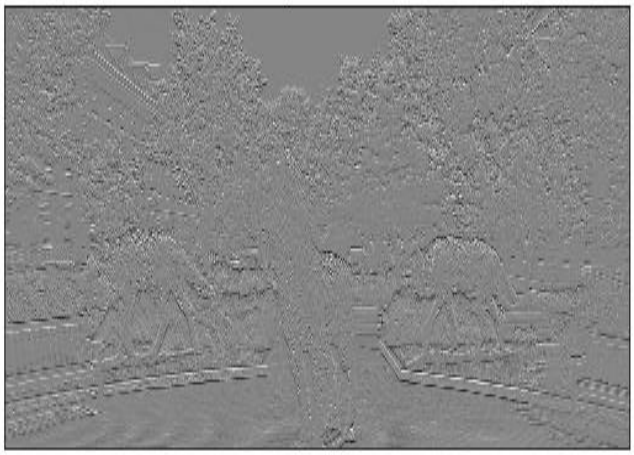
Px - Copy



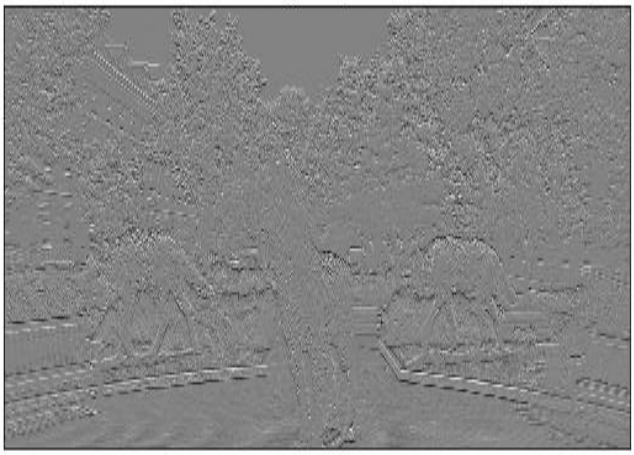
Px - Reflect



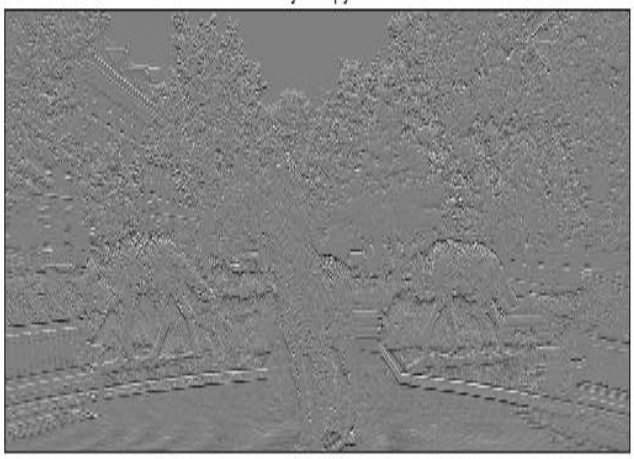
Py - Pad 0



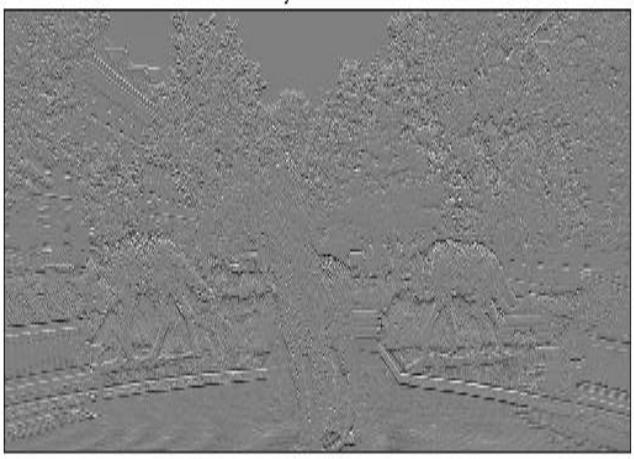
Py - Wrap



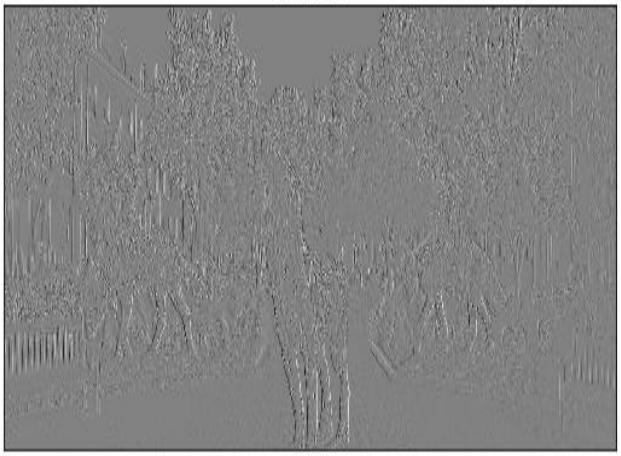
Py - Copy



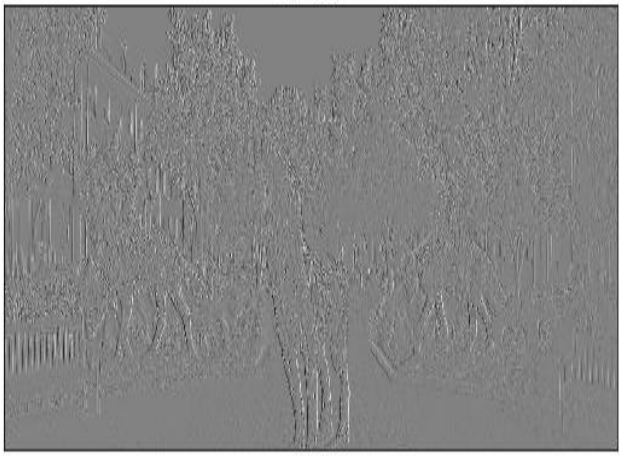
Py - Reflect



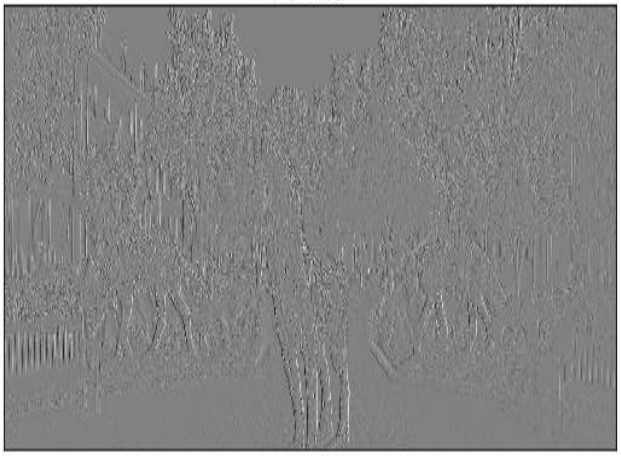
Sx - Pad 0



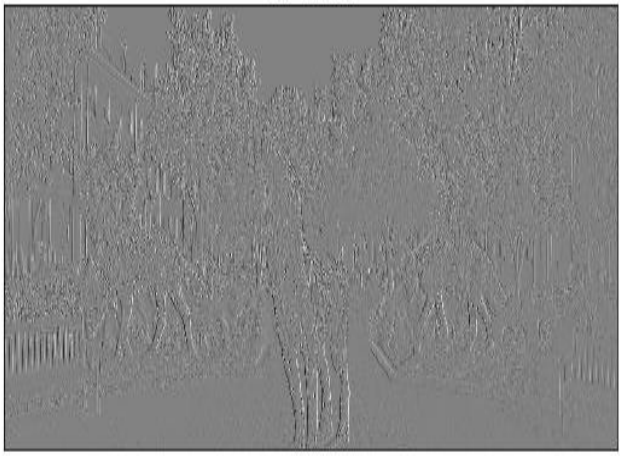
Sx - Wrap



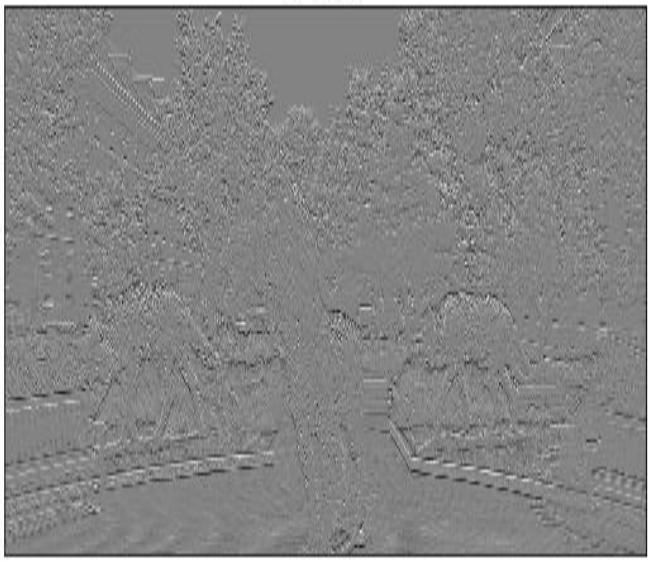
Sx - Copy



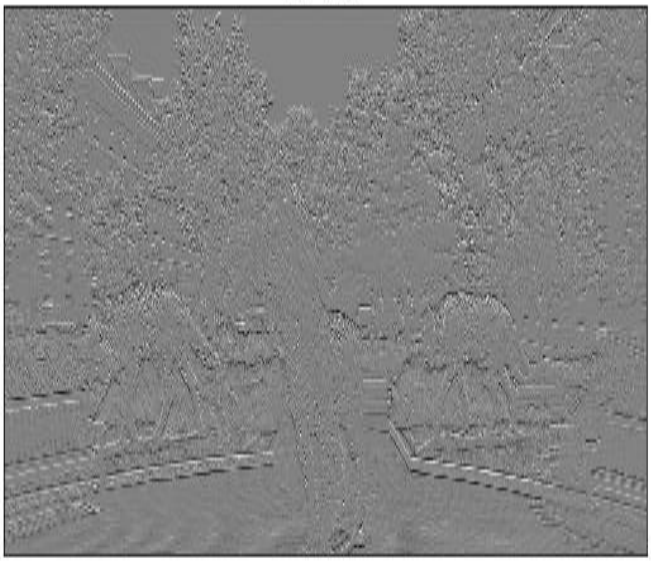
Sx - Reflect



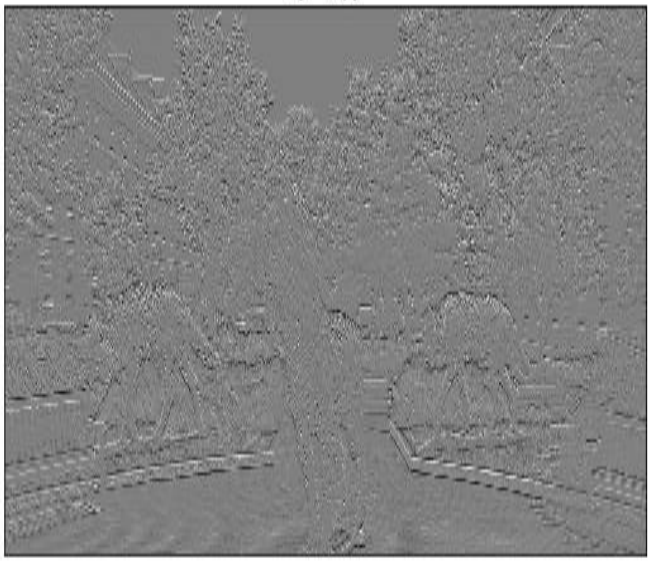
Sy - Pad 0



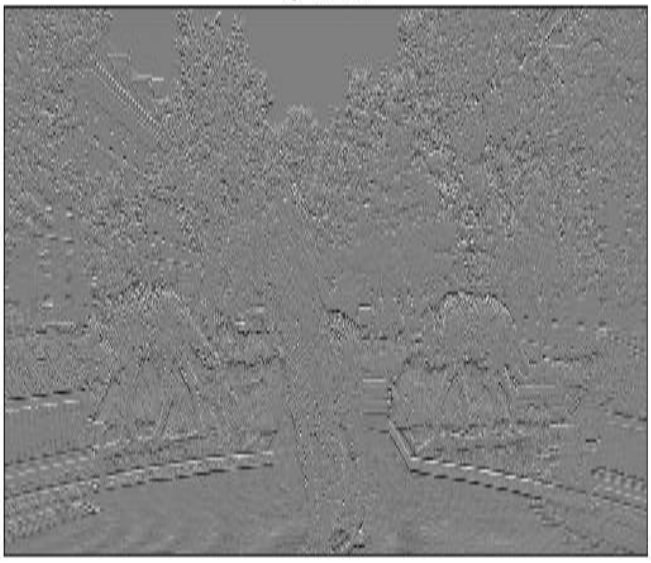
Sy - Wrap



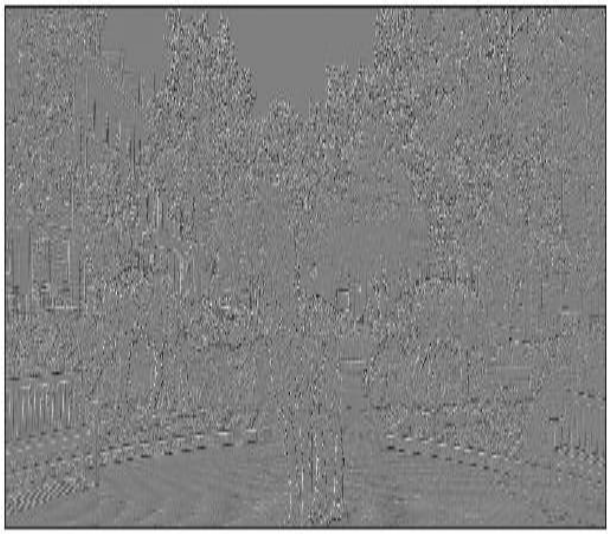
Sy - Copy



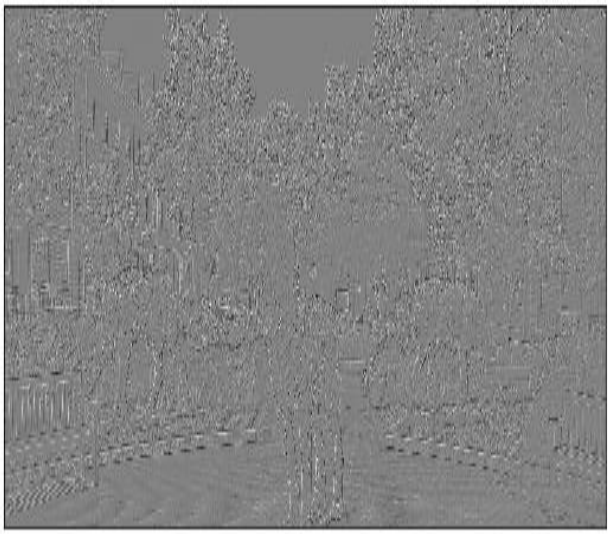
Sy - Reflect



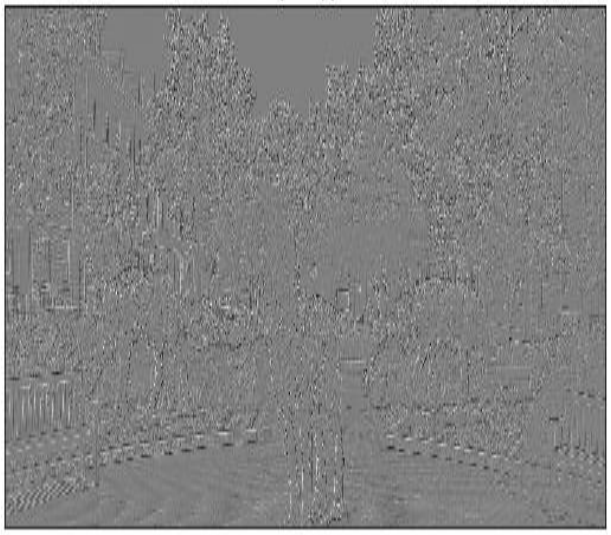
Ry - Pad 0



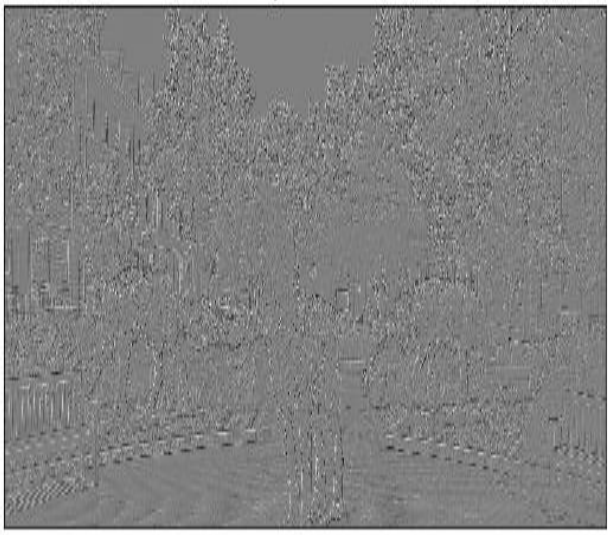
Ry - Wrap



Ry - Copy



Ry - Reflect



Box - Pad 0

Box - Wrap

Box - Copy

Box - Reflect

b.
Solution

Code –

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 18 18:40:37 2019

@author: Vivek Rathi
"""

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

#padding function
def pad(img,typepad):
    rows = img.shape[0]
    cols = img.shape[1]
    if len(img.shape) == 3:
        imgpad = np.zeros((rows+2,cols+2,3), dtype = 'float')
    elif len(img.shape) == 2:
        imgpad = np.zeros((rows+2,cols+2), dtype = 'float')
    imgpad[1:rows+1,1:cols+1] = img
    rows = imgpad.shape[0]
    cols = imgpad.shape[1]
    if typepad == "zero":
        return imgpad
    elif typepad == "wrap":
        imgpad[:,0] = imgpad[:,cols-2]
        imgpad[:,cols-1] = imgpad[:,1]
        imgpad[0,:] = imgpad[rows-2,:]
        imgpad[rows-1,:] = imgpad[1,:]
        imgpad[0,0] = imgpad[rows-2,cols-2]
        imgpad[rows-1,0] = imgpad[1,cols-2]
        imgpad[0,cols-1] = imgpad[rows-2,1]
        imgpad[rows-1,cols-1] = imgpad[1,1]
        return imgpad
    elif typepad == "copyedge":
        imgpad[0,:] = imgpad[1,:]
        imgpad[rows-1,:] = imgpad[rows-2,:]
        imgpad[:,0] = imgpad[:,1]
        imgpad[:,cols-1] = imgpad[:,cols-2]
        imgpad[0,0] = imgpad[1,1]
        imgpad[rows-1,0] = imgpad[rows-2,1]
        imgpad[0,cols-1] = imgpad[1,cols-2]
        imgpad[rows-1,cols-1] = imgpad[rows-2,cols-2]
        return imgpad
    elif typepad == "reflect":
```

```
        imgpad[0,:] = imgpad[2,:]
        imgpad[rows-1,:] = imgpad[rows-3,:]
        imgpad[:,0] = imgpad[:,2]
        imgpad[:,cols-1] = imgpad[:,cols-3]
        imgpad[0,0] = imgpad[2,2]
        imgpad[rows-1,0] = imgpad[rows-3,2]
        imgpad[0,cols-1] = imgpad[2,cols-3]
        imgpad[rows-1,cols-1] = imgpad[rows-3,cols-3]
        return imgpad

# convolution function
def conv(img,k,p):
    imgcopy = img.astype(np.float32)
    if p == 0:
        padimg = pad(img,"zero")
    elif p == 1:
        padimg = pad(img,"wrap")
    elif p == 2:
        padimg = pad(img,"copyedge")
    elif p == 3:
        padimg = pad(img,"reflect")

    if len(img.shape) == 3:
        B = np.zeros((k.shape[0],k.shape[1],3))
        r = padimg.shape[0]
        c = padimg.shape[1]
        if k.shape[0] == k.shape[1] == 3:
            for i in range(r-2):
                for j in range(c-2):
                    B[:,:,0] = np.multiply(k,padimg[i:i+3,j:j+3,0])
                    B[:,:,1] = np.multiply(k,padimg[i:i+3,j:j+3,1])
                    B[:,:,2] = np.multiply(k,padimg[i:i+3,j:j+3,2])
                    imgcopy[i,j,0] = np.sum(B[:,:,0])
                    imgcopy[i,j,1] = np.sum(B[:,:,1])
                    imgcopy[i,j,2] = np.sum(B[:,:,2])
            return imgcopy
        elif k.shape[0] == k.shape[1] == 2:
            if k.all == k_rx.all():
                for i in range(r-2):
                    for j in range(c-2):
                        B[:,:,0] = np.multiply(k,padimg[i+1:i+3,j:j+2,0])
                        B[:,:,1] = np.multiply(k,padimg[i+1:i+3,j:j+2,1])
                        B[:,:,2] = np.multiply(k,padimg[i+1:i+3,j:j+2,2])
                        imgcopy[i,j,0] = np.sum(B[:,:,0])
                        imgcopy[i,j,1] = np.sum(B[:,:,1])
                        imgcopy[i,j,2] = np.sum(B[:,:,2])
                return imgcopy
            else:
                for i in range(r-2):
```

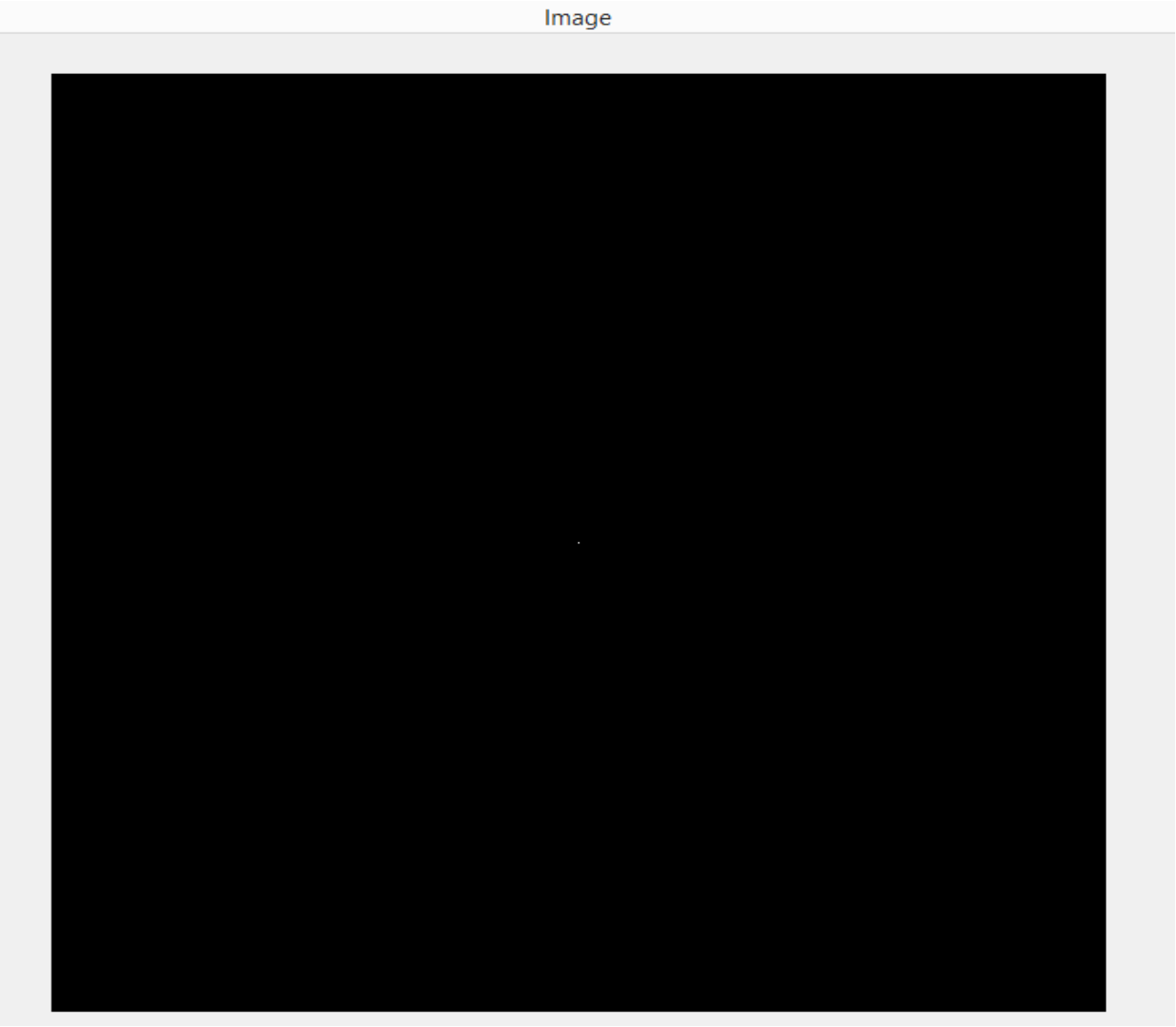
```
            for j in range(c-2):
                B[:,:,0] = np.multiply(k,padimg[i+1:i+3,j+1:j+3,0])
                B[:,:,1] = np.multiply(k,padimg[i+1:i+3,j+1:j+3,1])
                B[:,:,2] = np.multiply(k,padimg[i+1:i+3,j+1:j+3,2])
                imgcopy[i,j,0] = np.sum(B[:,:,0])
                imgcopy[i,j,1] = np.sum(B[:,:,1])
                imgcopy[i,j,2] = np.sum(B[:,:,2])
        return imgcopy

    elif k.shape[0] < k.shape[1]:
        for i in range(r-2):
            for j in range(c-2):
                B[:,:,0] = np.multiply(k,padimg[i+1:i+2,j+1:j+3,0])
                B[:,:,1] = np.multiply(k,padimg[i+1:i+2,j+1:j+3,1])
                B[:,:,2] = np.multiply(k,padimg[i+1:i+2,j+1:j+3,2])
                imgcopy[i,j,0] = np.sum(B[:,:,0])
                imgcopy[i,j,1] = np.sum(B[:,:,1])
                imgcopy[i,j,2] = np.sum(B[:,:,2])
        return imgcopy
    elif k.shape[0] > k.shape[1]:
        for i in range(r-2):
            for j in range(c-2):
                B[:,:,0] = np.multiply(k,padimg[i+1:i+3,j+1:j+2,0])
                B[:,:,1] = np.multiply(k,padimg[i+1:i+3,j+1:j+2,1])
                B[:,:,2] = np.multiply(k,padimg[i+1:i+3,j+1:j+2,2])
                imgcopy[i,j,0] = np.sum(B[:,:,0])
                imgcopy[i,j,1] = np.sum(B[:,:,1])
                imgcopy[i,j,2] = np.sum(B[:,:,2])
        return imgcopy
elif len(img.shape) == 2:
    B = np.zeros((k.shape[0],k.shape[1]))
    r = padimg.shape[0]
    c = padimg.shape[1]
    if k.shape[0] == k.shape[1] == 3:
        for i in range(r-2):
            for j in range(c-2):
                B = np.multiply(k,padimg[i:i+3,j:j+3])
                imgcopy[i,j] = np.sum(B)
        return imgcopy
    elif k.shape[0] == k.shape[1] == 2:
        if k.all() == k_rx.all():
            for i in range(r-2):
                for j in range(c-2):
                    B = np.multiply(k,padimg[i+1:i+3,j:j+2])
                    imgcopy[i,j] = np.sum(B)
            return imgcopy
        else:
            for i in range(r-2):
                for j in range(c-2):
```

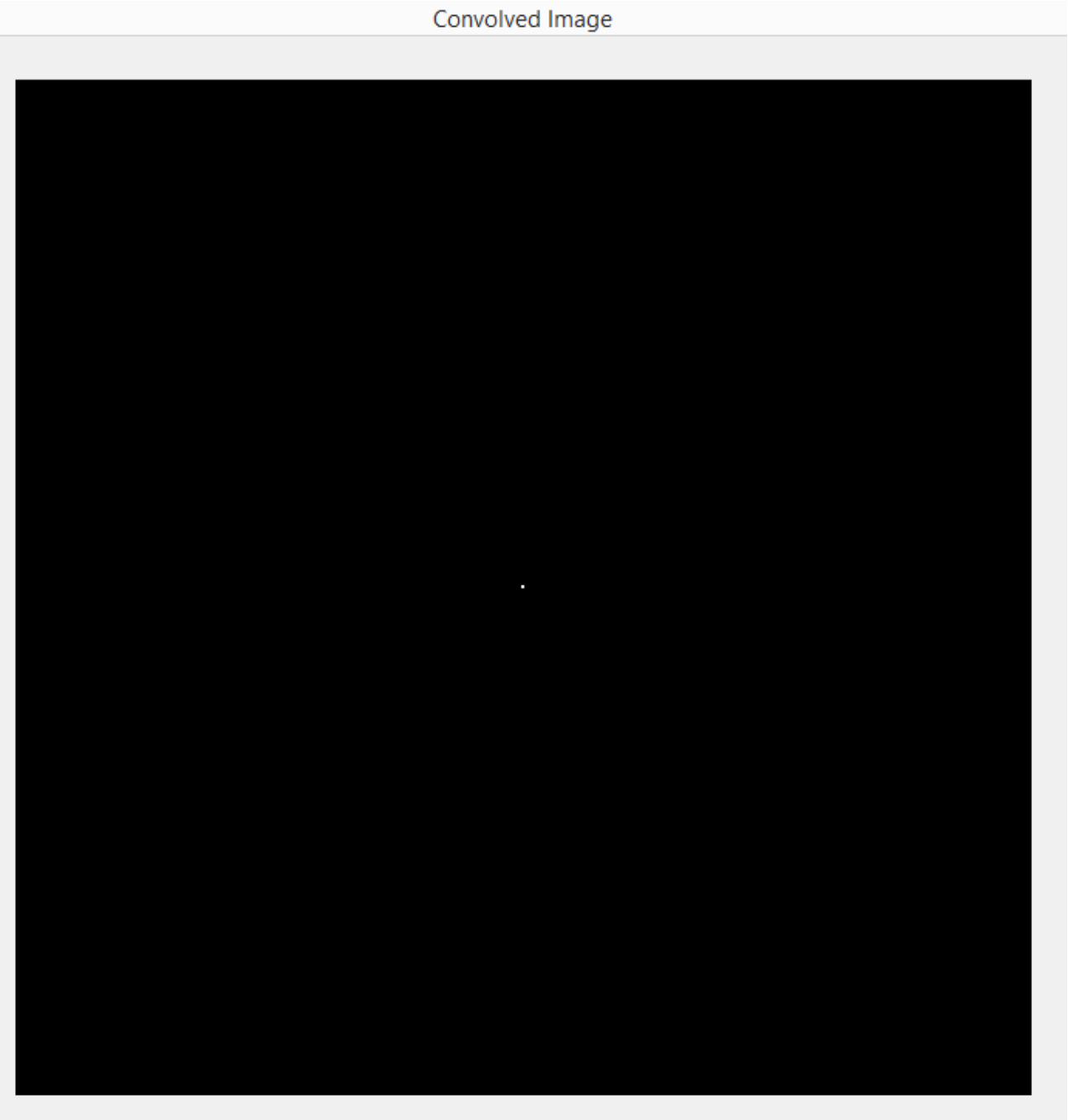
```
                B = np.multiply(k,padimg[i+1:i+3,j+1:j+3])
                imgcopy[i,j] = np.sum(B)
        return imgcopy

    elif k.shape[0] < k.shape[1]:
        for i in range(r-2):
            for j in range(c-2):
                B = np.multiply(k,padimg[i+1:i+2,j+1:j+3])
                imgcopy[i,j] = np.sum(B)
        return imgcopy
    elif k.shape[0] > k.shape[1]:
        for i in range(r-2):
            for j in range(c-2):
                B = np.multiply(k,padimg[i+1:i+3,j+1:j+2])
                imgcopy[i,j] = np.sum(B)
        return imgcopy


'''
Kernels
'''
k_box = (1/9) * np.ones((3,3))
k_x1 = np.matrix(([-1,1]))
k_y1 = np.matrix(([-1],[1]))
k_px = np.matrix(([-1,0,1],[-1,0,1],[-1,0,1]))
k_py = np.matrix(([1,1,1],[0,0,0],[-1,-1,-1]))
k_sx = np.matrix(([-1,0,1],[-2,0,2],[-1,0,1]))
k_sy = np.matrix(([1,2,1],[0,0,0],[-1,-2,-1]))
k_rx = np.matrix(([0,1],[-1,0]))
k_ry = np.matrix(([1,0],[0,-1]))

#create image
img = np.zeros((1024,1024))
img[511,511] = 1

imgc = conv(img,k_box,0)

cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
cv2.imshow('Image',img)
cv2.namedWindow('Convolved Image', cv2.WINDOW_NORMAL)
cv2.imshow('Convolved Image',imgc)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output-

The filter is indeed performing convolution. It is clearly seen in the output image, where the pixel values i.e. 8-D neighbors of (511,511) pixel are changed from 1 to 0.111.

In convolution, each pixel value is changed as per the filter and its neighbors. It is evident from the snap below that, all 9 pixels including (511,511) are changed as per the filter, which proves it is performing convolution.





Pixel Values adjacent to (511,511)

```
In [18]: img[509:514,509:514]
Out[18]:
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
In [19]: imgc[509:514,509:514]
Out[19]:
array([[ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.11111111,  0.11111111,  0.11111111,  0.        ],
       [ 0.        ,  0.11111111,  0.11111111,  0.11111111,  0.        ],
       [ 0.        ,  0.11111111,  0.11111111,  0.11111111,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ]],
```

P2.

Solution-
Code

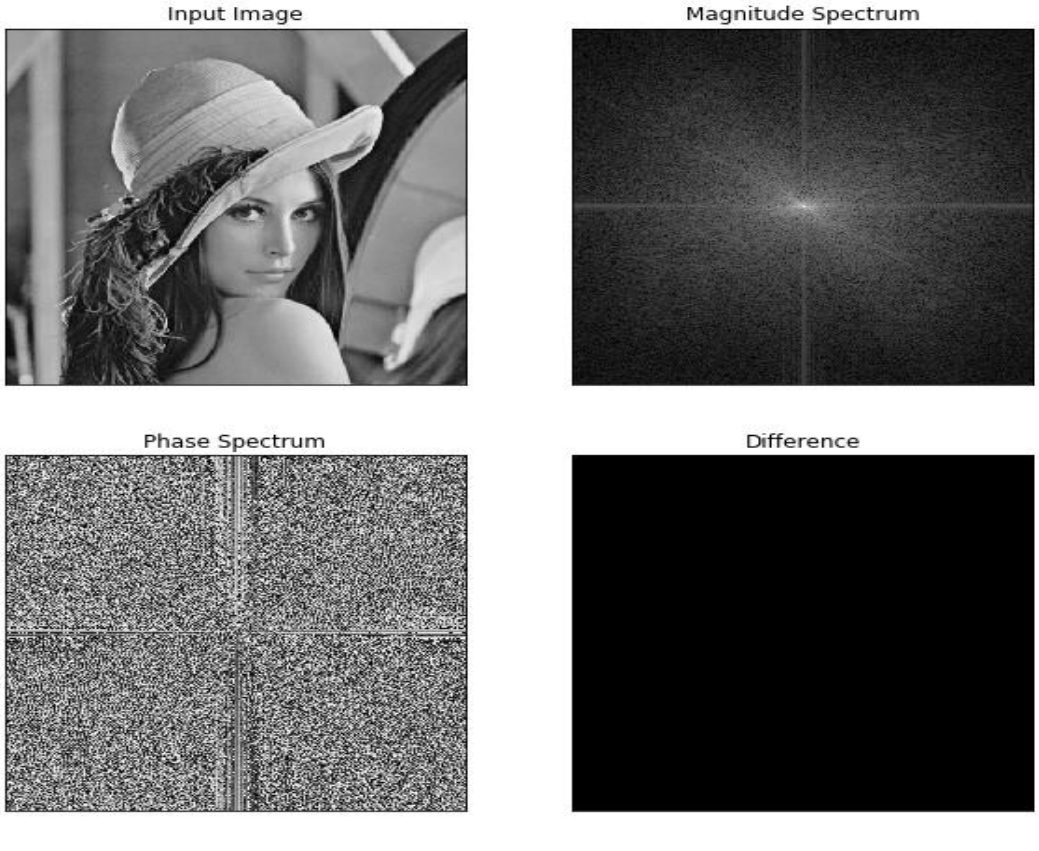
```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 13 21:05:29 2019

@author: Vivek Rathi
"""

import numpy as np
import cv2
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

'''
DFT2 - calculate 2d fft of i/p
'''
def DFT2(img):
    if (img.dtype != complex):
        fimg = img.astype(np.float)
    else:
        fimg = np.copy(img)
    fimg = fimg / fimg.max()
    fftx = np.fft.fft(fimg,axis=0)
    fftxy = np.fft.fft(fftx,axis=1)
    return fftxy

'''
getimg - gives o/p as uint8
'''
def getimg(g,img):
    v = abs(g) * img.max()
    v = np.round(v).astype(np.uint8)
    return v
'''
shift - shifts fourier spectrum to the centre, i.e. low frequencies
'''
def shift(img):
    h = img.astype(np.float)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            h[i,j] = img[i,j]* (-1)**((i+j))
    return h
'''
IDFT2 - inv fft using fft
'''
def IDFT2(fft2d):
```

```
    f_conj = np.conj(fft2d)
    ffxy = DFT2(f_conj)
    ffxy = ffxy / (fft2d.shape[0]*fft2d.shape[1])
    ffxy = ffxy / (abs(ffxy)).max()
    #im = np.conj(ffxy)
    im = np.conj(ffxy)
    return im
# read image
limg = cv2.imread('wolves.png',0)
lshift = shift(limg)
fl = DFT2(lshift)

#wimg = cv2.imread('wolves.png',0)
#wshift = shift(wimg)
#fw = DFT2(lshift)

# magnitude and phase spectrum
ms_l = np.log(1+np.abs(fl))

ps_l = np.angle(fl)

# IDFT
gl = IDFT2(fl)
img_gl = getimg(gl,limg)

# difference
d_l = limg - img_gl

# Plots

plt.figure(figsize=(10,10))

plt.subplot(221)
plt.imshow(limg, cmap = 'gray')
plt.xticks([]), plt.yticks([])
plt.title('Input Image')

plt.subplot(222)
plt.imshow(ms_l, cmap = 'gray')
plt.xticks([]), plt.yticks([])
plt.title('Magnitude Spectrum')

plt.subplot(223)
plt.imshow(ps_l, cmap = 'gray')
plt.xticks([]), plt.yticks([])
plt.title('Phase Spectrum')

plt.subplot(224)
plt.imshow(d_l, cmap = 'gray')
```

```
plt.xticks([]), plt.yticks([])
plt.title('Difference')
plt.show()
```

Output

1. Lena Image



2. Wolves Image