ECE 558: Digital Imaging Systems PROJECT 02

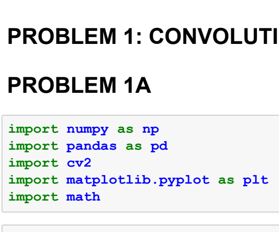
NAME: VIKRAM PANDE

UNITY_ID: vspande

PROBLEM 1: CONVOLUTION FROM SCRATCH

PROBLEM 1A

[illegible]



GETTING ALL THE KERNELS

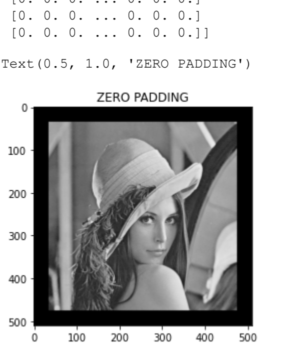
[illegible]

FUNCTION TO GET PADDING

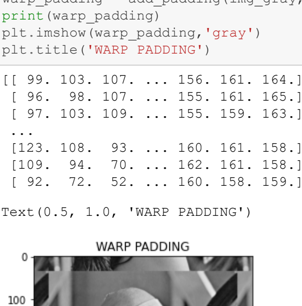
[illegible]

TESTING THE PADDING TYPES

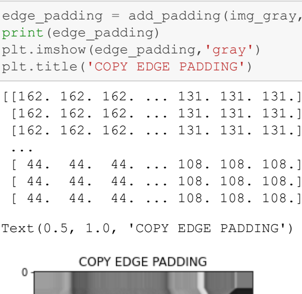
[illegible]



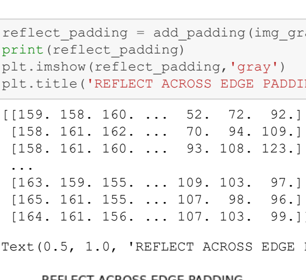
[illegible]



[illegible]



[illegible]



CONVOLUTION FUNCTION

[illegible]

```
    elif len(f.shape) == 3:

        b,g,r = cv2.split(f)

        fb_padded = add_padding(b, w.shape[0], w.shape[1], pad)
        fg_padded = add_padding(g, w.shape[0], w.shape[1], pad)
        fr_padded = add_padding(r, w.shape[0], w.shape[1], pad)

        convolved_bmatrix = np.zeros((b.shape[0],b.shape[1]))
        convolved_gmatrix = np.zeros((g.shape[0],g.shape[1]))
        convolved_rmatrix = np.zeros((r.shape[0],r.shape[1]))

        for r in range(convolved_bmatrix.shape[0]):

            for c in range(convolved_bmatrix.shape[1]):
                convolved_bmatrix[r][c]= np.sum(np.multiply(fb_padded[r:r+w.shape[0],c:c+w.shape[1]],w
))
                convolved_gmatrix[r][c]= np.sum(np.multiply(fg_padded[r:r+w.shape[0],c:c+w.shape[1]],w
))
                convolved_rmatrix[r][c]= np.sum(np.multiply(fr_padded[r:r+w.shape[0],c:c+w.shape[1]],w
))

        convolved_matrix = cv2.merge((convolved_bmatrix,convolved_gmatrix,convolved_rmatrix)).astype(np
.uint8)
    else:
        print("\nInput out of bounds")

    return convolved_matrix
```

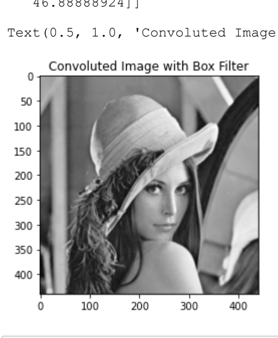
TESTCASES

LENA IMAGE - 9 FILTERS - Zero Padding

In [10]:
```
# Convoluted Image with Box Filter
box_result = conv2(img_gray, box, 0)
print(box_result)
plt.imshow(box_result,'gray')
plt.title('Convoluted Image with Box Filter')
```

```
[[ 72.00000054 107.77777858 107.77777858 ... 111.77777861 102.88888966
   64.66666715]
 [108.0000008  161.66666787 161.66666787 ... 167.22222347 153.88889004
   96.66666739]
 [108.0000008  161.77777898 161.77777898 ... 168.11111236 155.22222338
   97.66666739]
 ...
 [ 31.22222245  47.66666702  49.55555592 ...  99.66666741  99.77777852
   66.22222272]
 [ 29.88888911  47.11111146  50.11111148 ... 102.3333341  103.00000077
   68.8888894 ]
 [ 20.11111126  31.88888913  34.00000025 ...  68.66666718  69.44444496
   46.88888924]]
```

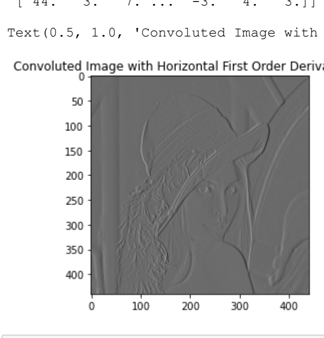
Out[10]: Text(0.5, 1.0, 'Convoluted Image with Box Filter')



In [11]:
```
# Convoluted Image with Horizontal First Order Derivative Filter
foh_result = conv2(img_gray, foh, 0)
print(foh_result)
plt.imshow(foh_result,'gray')
plt.title('Convoluted Image with Horizontal First Order Derivative Filter')
```

```
[[162.   0.  -1. ...   1. -12. -29.]
 [162.   0.  -1. ...   1. -12. -29.]
 [162.   0.  -1. ...   1. -13. -28.]
 ...
 [ 43.   2.   4. ...   6.  -3.  -4.]
 [ 43.   4.   5. ...   0.   2.   1.]
 [ 44.   3.   7. ...  -3.   4.   3.]]
```

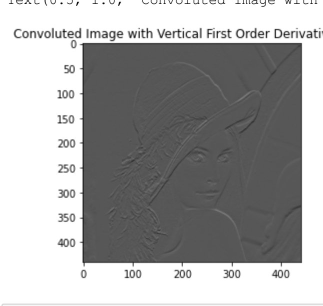
Out[11]: Text(0.5, 1.0, 'Convoluted Image with Horizontal First Order Derivative Filter')



In [12]:
```
# Convoluted Image with Vertical First Order Derivative Filter
fov_result = conv2(img_gray, fov, 0)
print(fov_result)
plt.imshow(fov_result,'gray')
plt.title('Convoluted Image with Vertical First Order Derivative Filter')
```

```
[[162. 162. 161. ... 172. 160. 131.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...  -1.  -2.  -1.]
 ...
 [ -9.  -6.   2. ...   8.   4.   5.]
 [  0.   2.   3. ...  -2.   3.   8.]
 [  1.   0.   2. ...  -1.   1.   3.]]
```

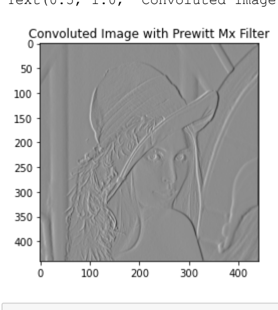
Out[12]: Text(0.5, 1.0, 'Convoluted Image with Vertical First Order Derivative Filter')



In [13]:
```
# Convoluted Image with Prewitt Mx Filter
pmx_result = conv2(img_gray, prewitt, 0)
print(pmx_result)
plt.imshow(pmx_result,'gray')
plt.title('Convoluted Image with Prewitt Mx Filter')
```

```
[[ 324.   -2.    0. ...  -22.  -82. -320.]
 [ 486.   -3.    0. ...  -34. -123. -478.]
 [ 486.   -2.    0. ...  -31. -121. -482.]
 ...
 [ 143.   10.   12. ...    9.   -8. -302.]
 [ 139.   25.   18. ...    6.    3. -310.]
 [  94.   19.   12. ...    3.   10. -209.]]
```

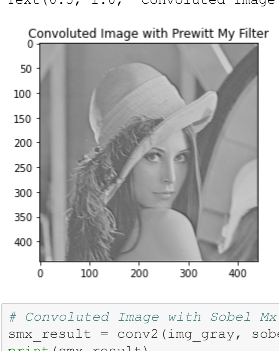
Out[13]: Text(0.5, 1.0, 'Convoluted Image with Prewitt Mx Filter')



In [14]:
```
# Convoluted Image with Prewitt My Filter
pmy_result = conv2(img_gray, prewitt_t, 0)
print(pmy_result)
plt.imshow(pmy_result,'gray')
plt.title('Convoluted Image with Prewitt My Filter')
```

```
[[   0. -163. -161. ... -183. -201. -291.]
 [ 324.  322.  324. ...  320.  264.    3.]
 [ 324.  323.  323. ...  320.  260.   -9.]
 ...
 [ 107.  112.  103. ...  186.  184.  -20.]
 [  91.  100.   97. ...  203.  204.  -15.]
 [  90.  142.  152. ...  308.  311.  209.]]
```

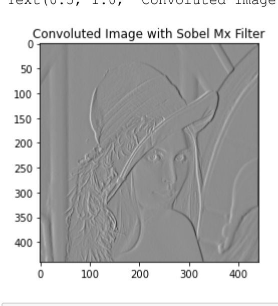
Out[14]: Text(0.5, 1.0, 'Convoluted Image with Prewitt My Filter')



In [15]:
```
# Convoluted Image with Sobel Mx Filter
smx_result = conv2(img_gray, sobel, 0)
print(smx_result)
plt.imshow(smx_result,'gray')
plt.title('Convoluted Image with Sobel Mx Filter')
```

```
[[ 486.   -3.    0. ...  -33. -123. -480.]
 [ 648.   -4.    0. ...  -45. -164. -638.]
 [ 648.   -3.    0. ...  -43. -162. -640.]
 ...
 [ 188.   16.   18. ...   12.  -15. -403.]
 [ 186.   34.   24. ...    8.    6. -414.]
 [ 141.   29.   18. ...    4.   17. -314.]]
```

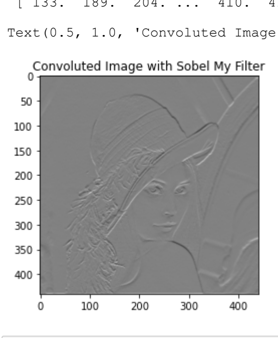
Out[15]: Text(0.5, 1.0, 'Convoluted Image with Sobel Mx Filter')



In [16]:
```
# Convoluted Image with Sobel My Filter
smy_result = conv2(img_gray, sobel_t, 0)
print(smy_result)
plt.imshow(smy_result,'gray')
plt.title('Convoluted Image with Sobel My Filter')
```

```
[[-486. -647. -646. ... -675. -623. -422.]
 [   0.    0.    0. ...    5.    6.    4.]
 [   0.   -1.   -2. ...  -11.  -16.  -14.]
 ...
 [  22.   12.   -8. ...  -28.  -33.  -33.]
 [  -4.  -10.  -14. ...   -4.  -16.  -26.]
 [ 133.  189.  204. ...  410.  415.  314.]]
```

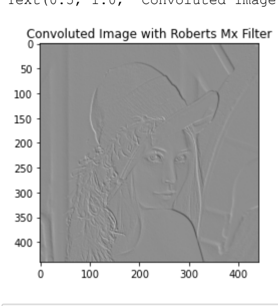
Out[16]: Text(0.5, 1.0, 'Convoluted Image with Sobel My Filter')



In [17]:
```
# Convoluted Image with Robert Mx Filter
rmx_result = conv2(img_gray, robert, 0)
print(rmx_result)
plt.imshow(rmx_result,'gray')
plt.title('Convoluted Image with Roberts Mx Filter')
```

```
[[   0. -162. -162. ... -171. -172. -160.]
 [ 162.    0.   -1. ...    1.  -12.  -29.]
 [ 162.    0.   -1. ...    2.  -11.  -27.]
 ...
 [  52.    8.    2. ...   -2.   -7.   -9.]
 [  43.    2.    2. ...    2.   -1.   -7.]
 [  43.    3.    5. ...   -2.    3.    0.]]
```

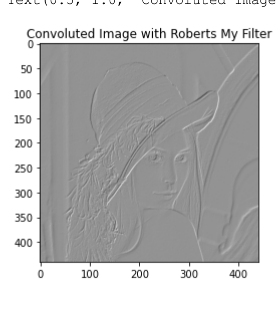
Out[17]: Text(0.5, 1.0, 'Convoluted Image with Roberts Mx Filter')



In [18]:
```
# Convoluted Image with Robert Mx Filter
rmy_result = conv2(img_gray, robert_t, 0)
print(rmy_result)
plt.imshow(rmy_result,'gray')
plt.title('Convoluted Image with Roberts My Filter')
```

```
[[-162. -162. -161. ... -172. -160. -131.]
 [-162.    0.    1. ...   -1.   12.   29.]
 [-162.    0.    1. ...    0.   14.   30.]
 ...
 [ -43.    7.    2. ...  -11.   -5.    0.]
 [ -43.   -4.   -7. ...   -4.    0.   -4.]
 [ -44.   -4.   -7. ...    1.   -3.   -4.]]
```

Out[18]: Text(0.5, 1.0, 'Convoluted Image with Roberts My Filter')



Wolves Image - 3 Kernels - 3 Paddings - 1 Color

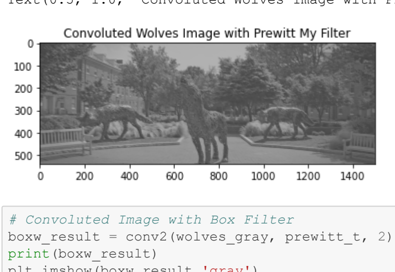
In [19]:
```
wolves = cv2.imread('wolves.png')
wolves_gray = cv2.cvtColor(wolves, cv2.COLOR_BGR2GRAY)
print(len(wolves.shape))
```

```
3
```

In [20]:
```
# Convoluted Image with Prewitt My Filter
pwmy_result = conv2(wolves_gray, prewitt_t, 1)
print(pwmy_result)
plt.imshow(pwmy_result,'gray')
plt.title('Convoluted Wolves Image with Prewitt My Filter')
```

```
[[244. 297. 291. ...  77.  92. 162.]
 [ 17.  11.  62. ...  34.  52.  34.]
 [-38.  16.  73. ...  70.  57. -47.]
 ...
 [211. 231. 227. ...  59.  55. 192.]
 [223. 240. 240. ...  69.  69. 208.]
 [237. 317. 325. ... 115. 106. 166.]]
```

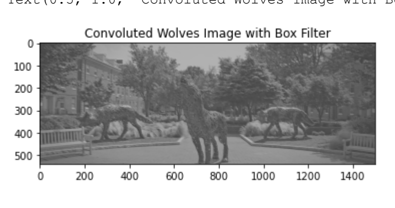
Out[20]: Text(0.5, 1.0, 'Convoluted Wolves Image with Prewitt My Filter')



In [21]:
```
# Convoluted Image with Box Filter
boxw_result = conv2(wolves_gray, prewitt_t, 2)
print(boxw_result)
plt.imshow(boxw_result,'gray')
plt.title('Convoluted Wolves Image with Box Filter')
```

```
[[ 19.   7.  18. ...  16.  29.  35.]
 [ 10.  11.  62. ...  34.  52.  45.]
 [ -3.  16.  73. ...  70.  57.   2.]
 ...
 [233. 231. 227. ...  59.  55.  50.]
 [239. 240. 240. ...  69.  69.  68.]
 [212. 213. 212. ...  94.  95.  95.]]
```

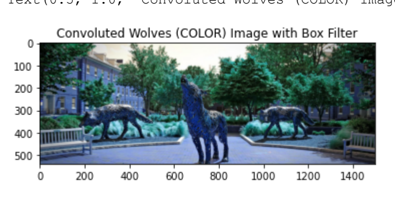
Out[21]: Text(0.5, 1.0, 'Convoluted Wolves Image with Box Filter')



COVOLUTION OF COLOR IMAGE

In [22]:
```
# Convoluted Color Image with Sobel Mx Filter
bc_result = conv2(wolves, box, 0)
#print(swmx_result)
plt.imshow(bc_result,'gray')
plt.title('Convoluted Wolves (COLOR) Image with Box Filter')
```

Out[22]: Text(0.5, 1.0, 'Convoluted Wolves (COLOR) Image with Box Filter')

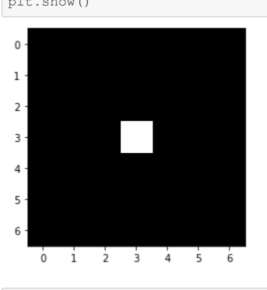


Problem 1B

In [23]:
```
unit_impulse = np.zeros((1024,1024))
unit_impulse[512,512] = 1
print(unit_impulse[509:516,509:516])
```

```
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
```

In [24]:
```
# Visualising
plt.imshow(unit_impulse[509:516,509:516], cmap = 'gray')
plt.show()
```



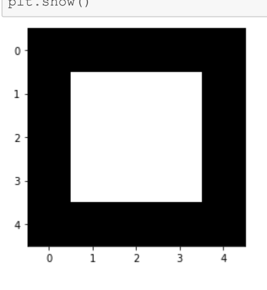
In [25]:
```
#BOX FILTER
kernel1 = 1/16*np.ones((4,4))
impulse_conv = conv2(unit_impulse, kernel1,1)
print(impulse_conv[510:516,510:516])
```

```
[[0.     0.     0.     0.     0.     0.    ]
 [0.     0.0625 0.0625 0.0625 0.0625 0.    ]
 [0.     0.0625 0.0625 0.0625 0.0625 0.    ]
 [0.     0.0625 0.0625 0.0625 0.0625 0.    ]
 [0.     0.0625 0.0625 0.0625 0.0625 0.    ]
 [0.     0.     0.     0.     0.     0.    ]]
```

In [26]:
```
kernel2 = 1/9*np.ones((3,3))
impulse_conv = conv2(unit_impulse, kernel2,0)
print(impulse_conv[510:515,510:515])
```

```
[[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.        ]]
```

In [27]:
```
#VIsualising
plt.imshow(impulse_conv[510:515,510:515],'gray')
plt.show()
```



Explanation - As we can see the convolution is working, the impulse i.e. the center is changing with the values of the Kernel applied.

In [ ]:

PROBLEM 2

Problem 2A

In [1]:
```
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread('lena.png')
img_copy = img.copy()
```

In [3]:
```
#Converting the Image into GrayScale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img_gray,'gray')
plt.title('Gray Scaled Lena Image')
```

Out[3]: `Text(0.5, 1.0, 'Gray Scaled Lena Image')`

In [4]:
```
img_gray
```

Out[4]:
```
array([[162, 162, 161, ..., 172, 160, 131],
       [162, 162, 161, ..., 172, 160, 131],
       [162, 162, 161, ..., 171, 158, 130],
       ...,
       [ 43,  45,  49, ..., 104, 101,  97],
       [ 43,  47,  52, ..., 102, 104, 105],
       [ 44,  47,  54, ..., 101, 105, 108]], dtype=uint8)
```

In [5]:
```
#Function to downscale the Image in range [0,1]
def downscale(f: np.array, min_input: int, max_input: int, min_output: int, max_output: int):
    """
    Input:
        f -> Input Image
        min_input -> Minimum input value
        max_input -> Maximum Input Value
        min_output -> Minimum output value
        max_output -> Maximum output Value
    Returns:
        Scaled Down Image -> np.array

    """
    # Similar implementation as Homework
    downscaled_image = (max_output - min_output) * (f / (max_input - min_input))
    return downscaled_image
```

In [6]:
```
downscaled_img = downscale(img_gray,0,255,0,1)
downscaled_img
```

Out[6]:
```
array([[0.63529412, 0.63529412, 0.63137255, ..., 0.6745098 , 0.62745098,
        0.51372549],
       [0.63529412, 0.63529412, 0.63137255, ..., 0.6745098 , 0.62745098,
        0.51372549],
       [0.63529412, 0.63529412, 0.63137255, ..., 0.67058824, 0.61960784,
        0.50980392],
       ...,
       [0.16862745, 0.17647059, 0.19215686, ..., 0.40784314, 0.39607843,
        0.38039216],
       [0.16862745, 0.18431373, 0.20392157, ..., 0.4       , 0.40784314,
        0.41176471],
       [0.17254902, 0.18431373, 0.21176471, ..., 0.39607843, 0.41176471,
        0.42352941]])
```

FUNCTION TO IMPLEMENT DFT

In [7]:
```
def DFT2(f: np.array) -> np.array:
    """
    Input:
        f - Input Downscaled Gray Image -> np.array
    Returns:
        DFT of Image -> np.array
    """
    #initializing the result with complex np matrix
    dfft = np.zeros(f.shape,dtype = complex)
    #Applying 1DFFT to each row of the image
    for i in range(f.shape[0]):
        dfft[i] = np.fft.fft(f[i])
    #Getting the transpose of the output fft rows matrix
    dfft = np.transpose(dfft)
    #Applying 1DFFT to the rows of the Transposed matrix i.e columns of original matrix

    for j in range(f.shape[1]):
        dfft[j] = np.fft.fft(dfft[j])
    #Transposing back to get the original matrix
    dfft = np.transpose(dfft)

    return dfft
```

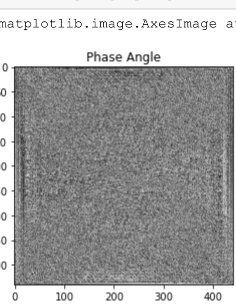
In [8]:
```
dft_image = DFT2(downscaled_img)
dft_image
```

Out[8]:
```
array([[ 9.41791569e+04   +0.j        , -1.32605209e+03+6928.25982131j,
         4.45443656e+03-3279.8744371j , ...,
        -6.91412640e+02-1838.48708722j,  4.45443656e+03+3279.8744371j ,
        -1.32605209e+03-6928.25982131j],
       [-8.37846358e+01-3545.76432875j, -4.60583826e+03+4606.16030726j,
        -3.20398724e+03 +217.75367606j, ...,
         1.89208255e+03+2351.68297678j, -1.02786238e+03+1296.80098955j,
         1.02025196e+02+2741.432367j  ],
       [-1.30229196e+03 -475.37875408j, -7.95709426e+02 -387.1093878j ,
         1.93767503e+03+1311.96674252j, ...,
         1.02391884e+03+2684.36743816j, -2.09555155e+03 -509.04882679j,
         2.37833148e+03 -182.71616886j],
       ...,
       [ 1.49437911e+03 -627.39365192j,  2.13376241e+03+2046.81793011j,
        -1.52536297e+03 -580.14634242j, ...,
         2.87895939e+03 +316.70305662j,  3.76828529e+02-1965.96854393j,
        -1.56015112e+02+1749.10229252j],
       [-1.30229196e+03 +475.37875408j,  2.37833148e+03 +182.71616886j,
        -2.09555155e+03 +509.04882679j, ...,
        -1.50090169e+03-2181.3107552j ,  1.93767503e+03+1311.96674252j,
        -7.95709426e+02 +387.1093878j ],
       [-8.37846358e+01+3545.76432875j,  1.02025196e+02-2741.432367j  ,
        -1.02786238e+03-1296.80098955j, ...,
        -1.48190250e+03+5509.4430389j , -3.20398724e+03 -217.75367606j,
        -4.60583826e+03-4606.16030726j]])
```

SPECTRUM AND PHASE ANGLE

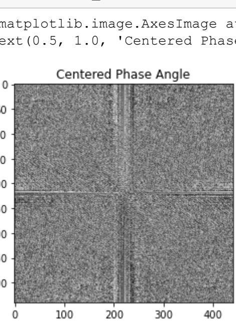
In [9]:
```
#To obtain the Phase angle of the image

# Phase Angle
pa = np.angle(dft_image)
pa
plt.imshow(pa,'gray'),plt.title('Phase Angle')
```

Out[9]: `(<matplotlib.image.AxesImage at 0x15ae225f8e0>, Text(0.5, 1.0, 'Phase Angle'))`



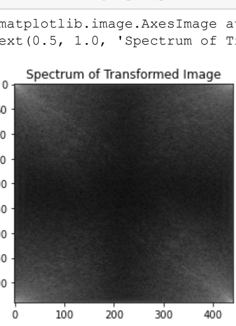
In [10]:
```
# Shifted Phase Angle
pa_shifted = np.fft.fftshift(pa)
plt.imshow(pa_shifted,'gray'),plt.title('Centered Phase Angle')
```

Out[10]:
```
(<matplotlib.image.AxesImage at 0x15ae244fdf0>,
 Text(0.5, 1.0, 'Centered Phase Angle'))
```



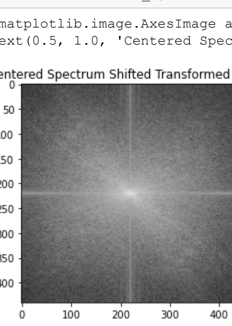
In [11]:
```
# Spectrum
#Transforming the Image with DFT
f = np.log(1+abs(dft_image))
plt.imshow(f,'gray'),plt.title('Spectrum of Transformed Image')
```

Out[11]:
```
(<matplotlib.image.AxesImage at 0x15ae27cef70>,
 Text(0.5, 1.0, 'Spectrum of Transformed Image'))
```



In [12]:
```
centered_spectrum = np.fft.fftshift(f)
centered_spectrum = np.log(1+abs(centered_spectrum))
plt.imshow(centered_spectrum,'gray'),plt.title('Centered Spectrum Shifted Transformed Image')
```

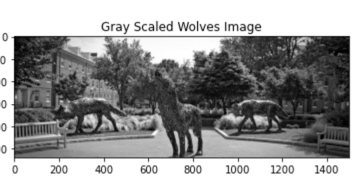
Out[12]:
```
(<matplotlib.image.AxesImage at 0x15ae2b46130>,
 Text(0.5, 1.0, 'Centered Spectrum Shifted Transformed Image'))
```



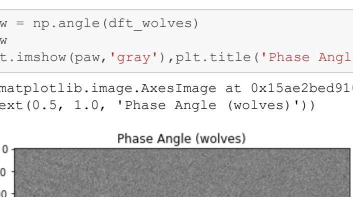
In the first images of spectrum and phase angle above, we can see the white spots at the corner of the image, they are low frequency components. We will take the low frequency components to the centre from corners so that the centre part will be lower frequency and others will be higher frequency components. For that, we will use "np.fft.fftshift(input_image). HENCE, shifting is done"

In [13]: [illegible]

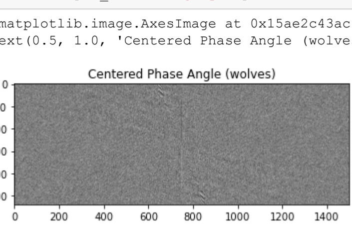
Out[13]: `Text(0.5, 1.0, 'Gray Scaled Wolves Image')`



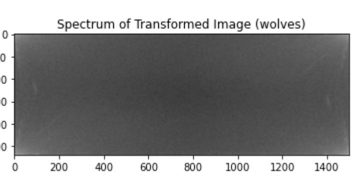
[illegible]



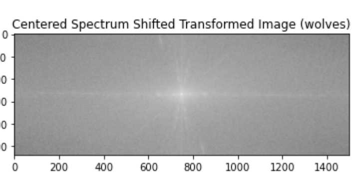
[illegible]



[illegible]

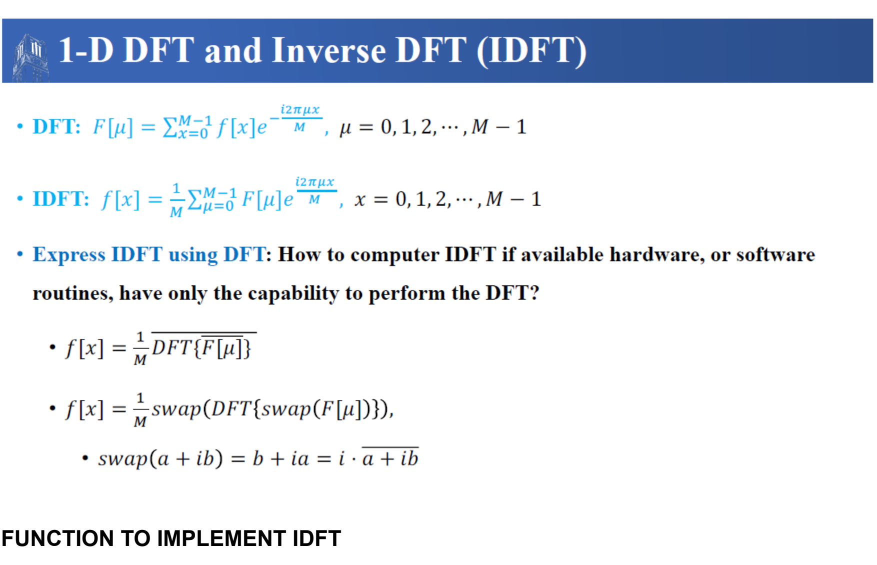


[illegible]



Problem 2B

IDFT Function

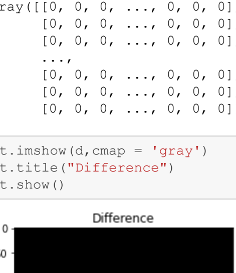


- **DFT:** $F[\mu] = \sum_{x=0}^{M-1} f[x] e^{-j\frac{2\pi\mu x}{M}}$, $\mu = 0, 1, 2, \ldots, M-1$
- **IDFT:** $f[x] = \frac{1}{M}\sum_{\mu=0}^{M-1} F[\mu] e^{j\frac{2\pi\mu x}{M}}$, $x = 0, 1, 2, \ldots, M-1$

- **Express IDFT using DFT: How to compute IDFT if available hardware, or software routines, have only the capability to perform the DFT?**
  - $f[x] = \frac{1}{M} \overline{DFT\{\overline{F[\mu]}\}}$
  - $f[x] = \frac{1}{M} swap(DFT\{swap(F[\mu])\})$,
    - $swap(a + ib) = b + ia = i \cdot \overline{a + ib}$

FUNCTION TO IMPLEMENT IDFT

[illegible]

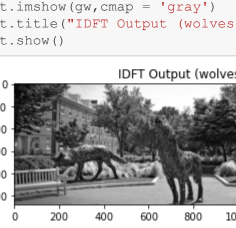


VERIFICATION OF IDFT

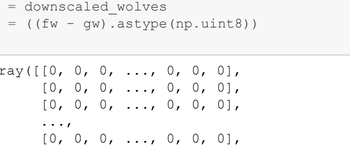
In [22]:
```
# Testing the difference
f = downscaled_img
d = ((f - g).astype(np.uint8))
d
```

Out[22]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

In [23]:
```
plt.imshow(d,cmap = 'gray')
plt.title("Difference")
plt.show()
```



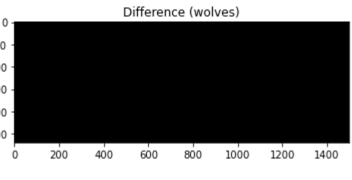
In [24]:
```
gw = IDFT2(dft_wolves)
plt.imshow(gw,cmap = 'gray')
plt.title("IDFT Output (wolves)")
plt.show()
```



In [25]:
```
fw = downscaled_wolves
dw = ((fw - gw).astype(np.uint8))
dw
```

Out[25]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

In [27]:
```
plt.imshow(dw,cmap = 'gray')
plt.title("Difference (wolves)")
plt.show()
```



In [ ]: