	<pre># Reading the Image img = cv2.imread('lena.png') img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) plt.imshow(img_gray,'gray') plt.show()</pre>
3]:	
	<pre>foh = np.array([[-1,1]]) #Horizontal First Order Derivative fov = np.array([[-1],[1]]) #Vertical First Order Derivative prewitt = np.array([[-1,0,1],[-1,0,1],[-1,0,1]]) #Prewitt Mx prewitt_t = np.array([[1,1,1],[0,0,0],[-1,-1,1]]) #Prewitt My sobel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]]) #Sobel Mx sobel_t = np.array([[1,2,1],[0,0,0],[-1,-2,-1]]) #Sobel My robert = np.array([[0,1],[-1,0]]) #Roberts Mx robert_t = np.array([[1,0],[0,-1]]) #Roberts My</pre> FUNCTION TO GET PADDING  def add_padding(f: np.array,wr: int,wc: int, pad: int) -> np.array:     """     Adds padding to the Input Matrix     Args:     1) Input Image -> f
	<pre>2) Kernel Size -&gt; w 3) Padding type -&gt; 0 for clip/zero-padding (Default)</pre>
	<pre>#Zero Padding extended_padding = (fr + topmost + downmost, fc + leftmost + rightmost) padded_image = np.zeros(extended_padding) #Fitting original image into zero padded matrix padded_image[topmost : topmost + fr, leftmost : leftmost + fc] = f  #Warp if pad == 1:     if topmost != 0:         padded_image[0:topmost,:]=padded_image[-1*(topmost + downmost):topmost+f.shape[0],:]     if downmost != 0:         padded_image[-1*(downmost) : , : ] = padded_image[topmost : topmost + downmost, :]     if rightmost != 0:         padded_image[ : ,-1*(rightmost) : ] = padded_image[ : ,leftmost : leftmost + rightmost]     if leftmost != 0:         padded_image[ : ,0 : leftmost] = padded_image[ : ,-1*(leftmost+rightmost) : leftmost + rape[1]]</pre>
	<pre>#Copy Edge elif pad == 2:     if topmost != 0:         padded_image[0 : topmost, : ] = padded_image[[topmost], : ]     if downmost != 0:         padded_image[-1*(downmost): , : ] = padded_image[[-1*downmost-1], :]     if rightmost != 0:         padded_image[ : ,-1*(rightmost) : ] = padded_image[ : ,[-1*(rightmost)-1]]     if leftmost != 0:         padded_image[ : ,0 : leftmost] = padded_image[ : ,[leftmost]]  # elif pad == 3:     if topmost != 0:         padded_image[0 : topmost, : ] = np.flip(padded_image[topmost : 2*topmost, :],axis = 0)     if downmost != 0:         padded_image[-1*(downmost) : , : ] = np.flip(padded_image[-2*(downmost) : -1*(downmost)], axis = 0)  if rightmost != 0:     padded_image[ : ,-1*(rightmost) : ] = np.flip(padded_image[ : , -2*(rightmost) : -1*(rightmost)], axis = 1)     if leftmost != 0:         padded_image[ : ,0 : leftmost] = np.flip(padded_image[ : ,leftmost : 2* leftmost],axis = return padded_image</pre>
	<pre>k = 71 zero_padding = add_padding(img_gray,71,71, 0) print(zero_padding) plt.imshow(zero_padding,'gray') plt.title('ZERO PADDING')  [[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.] [0. 0. 0 0. 0. 0.] [0. 0. 0 0. 0. 0.] [0. 0. 0 0. 0. 0.] [0. 0. 0 0. 0. 0.]</pre> Text(0.5, 1.0, 'ZERO PADDING')
	<pre>warp_padding = add_padding(img_gray,71,k, 1) print(warp_padding) plt.imshow(warp_padding,'gray')</pre>
6]:	plt.title('WARP PADDING')  [[ 99. 103. 107 156. 161. 164.]   [ 96. 98. 107 155. 161. 165.]   [ 97. 103. 109 155. 159. 163.]    [123. 108. 93 160. 161. 158.]   [109. 94. 70 162. 161. 158.]   [ 92. 72. 52 160. 158. 159.]]  Text(0.5, 1.0, 'WARP PADDING')  WARP PADDING  O  O  O  O  O  O  O  O  O  O  O  O  O
	<pre>adge_padding = add_padding(img_gray, 71, 71, 2) print(edge_padding) plt.imshow(edge_padding, 'gray') plt.title('COPY_EDGE_PADDING')  [[162. 162. 162 131. 131. 131.] [162. 162. 162 131. 131. 131.]</pre>
7]:	[162. 162. 162 131. 131. 131.] [44. 44. 44 108. 108. 108.] [44. 44. 44 108. 108. 108.] [44. 44. 44 108. 108.]  Text(0.5, 1.0, 'COPY EDGE PADDING')  COPY EDGE PADDING  100
	<pre>reflect_padding = add_padding(img_gray, k, k, 3) print(reflect_padding) plt.imshow(reflect_padding, 'gray') plt.title('REFLECT ACROSS EDGE PADDING')  [[159. 158. 160 52. 72. 92.] [158. 161. 162 70. 94. 109.] [158. 161. 160 93. 108. 123.] [163. 159. 155 109. 103. 97.] [165. 161. 155 107. 98. 96.]</pre>
3]:	[164. 161. 156 107. 103. 99.]]  Text(0.5, 1.0, 'REFLECT ACROSS EDGE PADDING')  REFLECT ACROSS EDGE PADDING  100 -
9]:[	CONVOLUTION FUNCTION  # CONVOLUTION FUNCTION  def conv2(f: np.array,w: np.array,pad: int):     """      Adds padding to the Input Matrix     Args:     1) Input Image -> f     2) Kernel -> w     3) Padding type -> 0 for clip/zero-padding (Default)
	<pre>Convoluted Image (np.array)  Gray Image is a two component Image. In case of three component images i.e. RGB, we need to spl ach component    and apply convolution seperately. Once the convolution is applied, we will merge the components ether and return    the convoluted RGB image.    """  # FOR GRAY IMAGES if len(f.shape) &lt; 3:     f_padded = add_padding(f,w.shape[0],w.shape[1], pad)         convolved_matrix = np.zeros((f.shape[0], f.shape[1]))         for r in range(convolved_matrix.shape[0]):</pre>
	<pre>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]):</pre>
	<pre>convolved_rmatrix[r][c] = np.sum(np.multiply(fr_padded[r:r+w.shape[0],c:c+w.shape[1]))  convolved_matrix = cv2.merge((convolved_bmatrix,convolved_gmatrix,convolved_rmatrix)).astyp .uint8) else:     print("\nInput out of bounds")  return convolved_matrix  TESTCASES  LENA IMAGE - 9 FILTERS - Zero Padding</pre>
	# 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] [108.000008 161.66666787 161.66666787 167.22222347 153.88889004 96.66666739] [108.000008 161.77777898 161.77777898 168.1111236 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 ]
0]:	[ 20.11111126
	# 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.
	Convoluted Image with Horizontal First Order Derivative Filter  50 - 100 - 150 - 200 - 250 - 300 - 300 - 400 - 100 - 200 300 400
	# 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.
	# Convoluted Image with Prewitt Mx Filter pmx_result = conv2(img_gray, prewitt, 0) print(pmx_result) plt.imshow(pmx result, 'gray')
	plt.imshow(pmx_result,'gray') plt.title('Convoluted Image with Prewitt Mx Filter')  [[ 3242.
	# 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')  [[ 0163161183201291.] [ 324. 322. 324 320. 264. 3.]
1]:	[ 324. 322. 324 320. 264. 3.] [ 324. 323. 323 320. 2609.] [ 107. 112. 103 186. 18420.] [ 91. 100. 97 203. 20415.] [ 90. 142. 152 308. 311. 209.]]  Text(0.5, 1.0, 'Convoluted Image with Prewitt My Filter')  Convoluted Image with Prewitt My Filter  50- 200- 250-
	# Convoluted Image with Sobel Mx Filter  smx_result = conv2(img_gray, sobel, 0)  print(smx_result)  plt.inshow(smx_result,'gray')  plt.title('Convoluted Image with Sobel Mx Filter')  [[ 4863.  033123480.]  [ 6484.  045164638.]  [ 6483.  043162640.]   [ 188.  16.  18 1215403.]
]:	[ 188. 16. 18 1215403.] [ 186. 34. 24 8. 6414.] [ 141. 29. 18 4. 17314.]]  Text(0.5, 1.0, 'Convoluted Image with Sobel Mx Filter')  Convoluted Image with Sobel Mx Filter  50- 100- 200- 250- 300- 350-
	# 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')  [[-486647646675623422.] [
]:	Text(0.5, 1.0, 'Convoluted Image with Sobel My Filter')  Convoluted Image with Sobel My Filter  50  100  200  350  400
	# 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')  [[ 0162162171172160.] [ 162.
]:	Convoluted Image with Roberts Mx Filter  100  200  250  300  400  # Convoluted Image with Robert Mx Filter  rmy_result = conv2 (img_gray, robert_t, 0)
	<pre>rmy_result = conv2(img_gray, robert_t, 0) print(rmy_result) plt.imshow(rmy_result,'gray') plt.title('Convoluted Image with Roberts My Filter')  [[-162162161172160131.] [-162.</pre>
	100 - 150 - 250 - 250 - 300 - 300 - 400  Wolves Image - 3 Kernels - 3 Paddings - 1 Color  wolves = cv2.imread('wolves.png')
)]:[	<pre>wolves = cv2.imread('wolves.png') wolves_gray = cv2.cvtColor(wolves, cv2.COLOR_BGR2GRAY) print(len(wolves.shape))  # 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. 5747.]</pre>
۱):[	[237. 317. 325 115. 106. 166.]]  Text(0.5, 1.0, 'Convoluted Wolves Image with Prewitt My Filter')  Convoluted Wolves Image with Prewitt My Filter  200 200 200 400 600 800 1000 1200 1400  # Convoluted Image with Box Filter boxw_result = conv2(wolves_gray, prewitt_t, 2) print(boxw_result) plt.imshow(boxw_result, 'gray')
2]:	COVOLUTION OF COLOR IMAGE  # 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')  Text(0.5, 1.0, 'Convoluted Wolves (COLOR) Image with Box Filter')  Convoluted Wolves (COLOR) Image with Box Filter')
8]:	Problem 1B  unit_impulse = np.zeros((1024,1024)) unit_impulse[512,512] = 1 print(unit_impulse[509:516,509:516])
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. 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] [0. 0. 0. 0. 0. 0. 0]  # Visualising plt.imshow(unit_impulse[509:516,509:516], cmap = 'gray') plt.show()  0- 1-
	2 -
5]:	<pre>kernel1 = 1/16*np.ones((4,4)) impulse_conv = conv2(unit_impulse, kernel1,1) print(impulse_conv[510:516,510:516])  [[0.</pre>

PROBLEM 2 **Problem 2A** import numpy as np In [1]: import pandas as pd import cv2 import matplotlib.pyplot as plt In [2]: | img = cv2.imread('lena.png') img\_copy = img.copy() #Converting the Image into GrayScale In [3]: 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') Gray Scaled Lena Image 50 100 150 200 250 300 350 400 100 In [4]: img\_gray Out[4]: array([[162, 162, 161, ..., 172, 160, 131],  $[162, 162, 161, \ldots, 172, 160, 131],$  $[162, 162, 161, \ldots, 171, 158, 130],$  $[43, 45, 49, \ldots, 104, 101, 97],$ [ 43, 47, 52, ..., 102, 104, 105], [ 44, 47, 54, ..., 101, 105, 108]], dtype=uint8) #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,[0.16862745, 0.18431373, 0.20392157, ..., 0.4]0.41176471], [0.17254902, 0.18431373, 0.21176471, ..., 0.39607843, 0.41176471,**FUNCTION TO IMPLEMENT DFT** In [7]: def DFT2(f: np.array) -> np.array: 11 11 11 Input: f - Input Downscaled Gray Image -> np.array 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 - 3545.76432875 j, -4.60583826e + 03 + 4606.16030726 j,-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) plt.imshow(pa,'gray'),plt.title('Phase Angle') Out[9]: (<matplotlib.image.AxesImage at 0x15ae225f8e0>, Text(0.5, 1.0, 'Phase Angle')) Phase Angle 50 100 150 200 250 300 350 400 200 300 400 100 # Shifted Phase Angle In [10]: 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')) Centered Phase Angle 0 50 100 150 200 250 300 350 400 300 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')) Spectrum of Transformed Image 0 50 100 150 200 300 350 400 100 200 300 400 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')) Centered Spectrum Shifted Transformed Image 50 100 150 200 250 300 350 400 0 100 200 300 400 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]: # Wolves Image wolves = cv2.imread('wolves.png') wolves gray = cv2.cvtColor(wolves, cv2.COLOR BGR2GRAY) plt.imshow(wolves\_gray,'gray') plt.title('Gray Scaled Wolves Image') Out[13]: Text(0.5, 1.0, 'Gray Scaled Wolves Image') Gray Scaled Wolves Image 100 200 300 400 500 400 600 800 1000 1200 1400 In [14]: downscaled wolves = downscale(wolves\_gray,0,255,0,1) downscaled wolves Out[14]: array([[0.01960784, 0.00392157, 0.03137255, ..., 0.07843137, 0.0745098, 0.05490196], [0.00392157, 0.03921569, 0.01568627, ..., 0.07843137, 0.04705882,[0.01176471, 0.01960784, 0.01960784, ..., 0.01568627, 0.00784314,0.01960784], [0.44705882, 0.44705882, 0.44705882, ..., 0.133333333, 0.1372549,[0.41176471, 0.41176471, 0.41176471, ..., 0.16470588, 0.17254902,0.17647059], 0.39607843, 0.39607843, ..., 0.14509804, 0.15294118, [0.4 0.15686275]]) In [15]: dft wolves = DFT2(downscaled wolves) dft wolves , -43779.52839391-33084.28644852j, Out[15]: array([[309749.69411765 +0.j -14588.54690431 -2617.35851031j, ..., -9596.71775755 +4111.57656599j, -14588.54690431 +2617.35851031j, -43779.52839391+33084.28644852j], [ 60326.78347039 +8892.10496815j, -34297.26423775 +5098.73972996j, 2150.60226237 -6131.17970098j, ..., -12472.41023507+11976.53044243j, -2621.14348949 -8360.04910609j, -25652.92495484 +8288.91090502j], [ 13252.9833408 +835.68788497j, -10709.34052319+12325.54974752j, -1940.52669912-10258.88679056j, ..., -3475.18055363 + 1242.605219j, -5917.53675298 - 7589.77996631j, 2558.83272453+17235.68910706j], [ 2586.05154279 +60.05017844j, 6939.14797489 -3688.72255071j, -8783.21091252 +7432.63540663j, ..., -2642.90862199 +2428.84450562j, 7579.1367225 -4271.37954036j, -8995.27295488 -6607.14483309j], [ 13252.9833408 -835.68788497j, 2558.83272453-17235.68910706j, -5917.53675298 +7589.77996631j, ..., 6195.79996278-12121.99364773j, -1940.52669912+10258.88679056j, -10709.34052319-12325.54974752j], [ 60326.78347039 -8892.10496815j, -25652.92495484 -8288.91090502j, -2621.14348949 +8360.04910609j, ..., -2542.94239322-22574.44097422j, 2150.60226237 +6131.17970098j, -34297.26423775 -5098.73972996j]]) In [16]: paw = np.angle(dft wolves) plt.imshow(paw, 'gray'), plt.title('Phase Angle (wolves)') Out[16]: (<matplotlib.image.AxesImage at 0x15ae2bed910>, Text(0.5, 1.0, 'Phase Angle (wolves)')) Phase Angle (wolves) 0 100 200 300 400 500 400 600 1000 1200 1400 200 800 In [17]: # Shifted Phase Angle paw shifted = np.fft.fftshift(paw) plt.imshow(paw shifted, 'gray'), plt.title('Centered Phase Angle (wolves)') Out[17]: (<matplotlib.image.AxesImage at 0x15ae2c43ac0>, Text(0.5, 1.0, 'Centered Phase Angle (wolves)')) Centered Phase Angle (wolves) 0 100 200 300 400 500 200 400 600 800 1000 1200 1400 In [18]: # Spectrum #Transforming the Image with DFT fw = np.log(1+abs(dft wolves))Out[18]: (<matplotlib.image.AxesImage at 0x15ae2c9cd00>, Text(0.5, 1.0, 'Spectrum of Transformed Image (wolves)')) Spectrum of Transformed Image (wolves) 0 100 200 300 400 500 600 1200 1400 200 400 1000 In [19]: centered\_spectrum\_w = np.fft.fftshift(fw) centered\_spectrum\_w = np.log(1+abs(centered\_spectrum\_w)) plt.imshow(centered\_spectrum\_w,'gray'),plt.title('Centered Spectrum Shifted Transformed Image (wolves)' Out[19]: (<matplotlib.image.AxesImage at 0x15ae310d070>, Text(0.5, 1.0, 'Centered Spectrum Shifted Transformed Image (wolves)')) Centered Spectrum Shifted Transformed Image (wolves) 100 200 300 400 500 200 400 1000 **Problem 2B IDFT Function** 1-D DFT and Inverse DFT (IDFT) • **DFT:**  $F[\mu] = \sum_{x=0}^{M-1} f[x] e^{-\frac{i2\pi\mu x}{M}}, \ \mu = 0, 1, 2, \dots, M-1$ • IDFT:  $f[x] = \frac{1}{M} \sum_{\mu=0}^{M-1} F[\mu] e^{\frac{i2\pi\mu x}{M}}, x = 0, 1, 2, \dots, M-1$ • Express IDFT using DFT: How to computer 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** In [20]: #According to the snippet above, implementing the same function def IDFT2(f: np.array): Inputs: f - input\_dft\_image -> np.array Returns: IDFT Image -> np.array # Using the formula given in the lecture slides f conjugate = np.imag(f) + complex('j') \* np.real(f) idft conjugate = DFT2(f conjugate) idft = np.imag(idft\_conjugate) + complex('j') \* np.real(idft\_conjugate) idft = idft/(f.shape[0] \* f.shape[1]) return np.real(idft) In [21]: g = IDFT2 (dft image)plt.imshow(g,cmap = 'gray') plt.title("IDFT Output") # We can see that when we use IDFT function on our DFT image, we will get back our downscaled image IDFT Output 0 50 100 150 200 250 300 350 400 100 0 200 300 400 **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],  $[0, 0, 0, \ldots, 0, 0, 0],$  $[0, 0, 0, \ldots, 0, 0, 0],$  $[0, 0, 0, \ldots, 0, 0, 0],$ [0, 0, 0, ..., 0, 0, 0]], dtype=uint8) In [23]: plt.imshow(d, cmap = 'gray') plt.title("Difference") plt.show() Difference 0 50 100 150 200 250 300 350 400 100 400 qw = IDFT2 (dft wolves)In [24]: plt.imshow(gw,cmap = 'gray') plt.title("IDFT Output (wolves)") plt.show() IDFT Output (wolves) 100 200 300 400 500 200 400 600 800 1000 1200 1400 fw = downscaled wolves In [25]: dw = ((fw - gw).astype(np.uint8))Out[25]: array([[0, 0, 0, ..., 0, 0],  $[0, 0, 0, \ldots, 0, 0, 0],$  $[0, 0, 0, \ldots, 0, 0, 0],$  $[0, 0, 0, \ldots, 0, 0, 0],$  $[0, 0, 0, \ldots, 0, 0, 0],$ [0, 0, 0, ..., 0, 0, 0]], dtype=uint8) In [27]: plt.imshow(dw,cmap = 'gray') plt.title("Difference (wolves)") plt.show() Difference (wolves) 0 100 200 300 400 500 200 400 600 800 1000 1200 1400 In [ ]: