

Contents

| | | |
|---|--------------------|---|
| 1 | Basic Test Results | 2 |
| 2 | README | 4 |
| 3 | answer q1.txt | 5 |
| 4 | answer q2.txt | 6 |
| 5 | answer q3.txt | 7 |
| 6 | sol2.py | 8 |

1 Basic Test Results

```
1 Archive: /tmp/bodek.8PP2i9/impr/ex2/liran.drory/presubmission/submission
2   inflating: current/answer_q2.txt
3   inflating: current/answer_q3.txt
4   inflating: current/README
5   inflating: current/sol2.py
6   inflating: current/answer_q1.txt
7 /usr/lib/python3/dist-packages/matplotlib/font_manager.py:280: UserWarning: Matplotlib is building the font cache using fc-list.
8   'Matplotlib is building the font cache using fc-list.'
9 ex2 presubmission script
10
11 Disclaimer
12 -----
13 The purpose of this script is to make sure that your code is compliant
14 with the exercise API and some of the requirements
15 The script does not test the quality of your results.
16 Don't assume that passing this script will guarantee that you will get
17 a high grade in the exercise
18
19 login: liran.drory
20
21 submitted files:
22 sol2.py
23
24 answer_q1.txt
25
26 answer_q2.txt
27
28 answer_q3.txt
29
30
31
32 answer to q1:
33 The difference of the magnitude happens according to the boundaries of the image
34
35 In the Fourier is cyclic
36
37 In the Convolution is not affected by the boundaries
38 just the frame (outter)
39 answer to q2:
40 when the gaussian not in the center the results of the blur image according to the boundry of fourier
41 will translate cyclic way, so that the image center will be where the center of the gaussian where.
42
43 because the gaussian is not in the center each pixel will be calculated from different region
44
45 answer to q3:
46 the difference between both of the blurring is the boundry conditions due to
47 fourier cyclic colculation (but it is very small)
48
49 section 1.1
50 DFT and IDFT
51 section 1.2
52 2D DFT and IDFT
53 section 2.1
54 derivative using convolution
55 Section 2.2
56 derivative using convolution
57 Section 3.1
58 blur spatial
59 Section 3.1
```

```
60 blur fourier
61 all tests Passed.
62 - Pre-submission script done.
63
64 Please go over the output and verify that there are no failures/warnings.
65 Remember that this script tested only some basic technical aspects of your implementation
66 It is your responsibility to make sure your results are actually correct and not only
67 technically valid.
```

2 README

```
1  liran.drory
2  sol2.py
3  answer_q1.txt
4  answer_q2.txt
5  answer_q3.txt
```

3 answer q1.txt

```
1 The difference of the magnitude happens according to the boundaries of the image
2
3 In the Fourier is cyclic
4
5 In the Convolution is not affected by the boundaries
6 just the frame (outter)
```

4 answer q2.txt

```
1  when the gaussian not in the center the results of the blur image according to the boundry of fourier
2  will translate cyclic way, so that the image center will be where the center of the gaussian where.
3
4  because the gaussian is not in the center each pixel will be calculated from different region
```

5 answer q3.txt

- 1 the difference between both of the blurring is the boundry conditions due to
- 2 fourier cyclic colculation (but it is very small)

6 sol2.py

```
1  __author__ = "Liran Drory"
2
3  import numpy as np
4  from scipy.signal import convolve2d as conv
5  import matplotlib.pyplot as plt
6  from scipy.misc import imread as imread
7  from skimage.color import rgb2gray
8
9  GRAYSCAL = 1
10 GRAYSCAL_MATRIX = 2
11 RGB = 2
12 RGB_MATRIX = 3
13 HIEGTH = 0
14 WIDTH = 1
15 SHADES_OF_GRAY = 255
16 COLORFULL = 3
17
18
19 def read_image(filename, representation):
20     """
21     Read Image and return matrix [0,1] float64
22     Gray scale - 2D
23     RGB - 3D
24
25     Parameters
26     -----
27     :param filename: str
28         string containing the image filename to read (PATH)
29
30     :param representation: int
31         either 1 or 2 defining whether the output
32         should be a greyscale image (1) or an RGB image (2).
33
34     Returns
35     -----
36     :return numpy array with either 2D matrix or 3D matrix
37         describing the pixels of the image
38     """
39
40
41     # loads the image
42     im = imread(filename)
43
44     if representation == RGB:
45         im_float = im.astype(np.float64) # pixels to float
46         im_float /= 255 # pixels [0,1]
47         return im_float
48
49     if representation == GRAYSCAL:
50         im_g = im.astype(np.float64) # pixels to float
51         im_g = rgb2gray(im_g) # turn to grey
52         return im_g
53
54
55 def DFT(signal):
56     """
57     Function that return DFT of signal
58     if matrix is input: return every row DFT
59
```



```

60     Parameters
61     -----
62     :param signal
63
64     Returns
65     -----
66     :return complex_fourier_signal
67
68     """
69
70     # find the length of the signal
71     N = signal.shape[0]
72     if signal.ndim == 2:
73         M, N = signal.shape
74
75     # calculate DFT matrix
76     u, v = np.meshgrid(np.arange(N), np.arange(N))
77     omega = np.exp(-2 * np.pi * 1j / N)
78     dft_matrix = np.power(omega, u*v)
79
80     # if it is a matrix of signals
81     if signal.ndim == 2:
82         # calculate the Fourier Transform
83         complex_fourier_signal = np.dot(dft_matrix, signal.transpose())
84         return complex_fourier_signal.transpose()
85
86     # calculate the Fourier Transform
87     complex_fourier_signal = np.dot(dft_matrix, signal)
88     return complex_fourier_signal
89
90
91 def IDFT(fourier_signal):
92     """
93     Function that return IDFT of fourier signal
94     if matrix is input: return every row IDFT
95
96     Parameters
97     -----
98     :param fourier_signal
99
100    Returns
101    -----
102    :return 1/N * signal
103
104    """
105
106    # find the length of the signal
107    N = fourier_signal.shape[0]
108    if fourier_signal.ndim == 2:
109        M, N = fourier_signal.shape
110
111    # calculate IDFT matrix
112    u, v = np.meshgrid(np.arange(N), np.arange(N))
113    omega = np.exp(2 * np.pi * 1j / N)
114    idft_matrix = np.power(omega, u*v)
115
116    # if it is a matrix of fourier signals
117    if fourier_signal.ndim == 2:
118        # calculate the Fourier Transform
119        signal = np.dot(idft_matrix, fourier_signal.transpose())
120        return 1/N * signal.transpose()
121
122    # calculate the inverse Fourier Transform
123    signal = np.dot(idft_matrix, fourier_signal)
124    return 1/N * signal
125
126
127 def DFT2(image):

```

```

128     """
129     Function that return 2D DFT of image
130
131     Parameters
132     -----
133     :param image (matrix)
134
135     Returns
136     -----
137     :return fourier_image
138
139     """
140
141     M, N = image.shape
142
143     # build the dft2_matrix transform
144     omega_y = np.exp(-2 * np.pi * 1j / M)
145     u, v = np.meshgrid(np.arange(M), np.arange(M))
146     dft2_matrix = np.power(omega_y, u*v)
147
148     # calculate the 2D fourier transform
149     fourier_image = np.dot(dft2_matrix, DFT(image))
150
151     return fourier_image
152
153
154 def IDFT2(fourier_image):
155     """
156     Function that return 2D IDFT of an image
157
158     Parameters
159     -----
160     :param fourier_image
161
162     Returns
163     -----
164     :return image
165
166     """
167
168     M, N = fourier_image.shape
169     # build the idft2_matrix transform
170     omega_y = np.exp(2 * np.pi * 1j / M)
171     u, v = np.meshgrid(np.arange(M), np.arange(M))
172     idft2_matrix = np.power(omega_y, u*v)
173
174     # calculate the 2D inverse fourier transform
175     return 1/M * np.dot(idft2_matrix, IDFT(fourier_image))
176
177
178 def conv_der(im):
179     """
180     derivative of an image using convolution
181
182     Parameters
183     -----
184     :param im
185
186     Returns
187     -----
188     :return magnitude of the derivative
189
190     """
191
192     # set der x/y matrix
193     der_x = np.array([[1, 0, -1]])
194     der_y = np.array(der_x.transpose())
195     # calculate the derivative to x and y

```

```

196     dx = conv(im, der_x, mode='same')
197     dy = conv(im, der_y, mode='same')
198
199     return np.sqrt(np.abs(dx)**2 + np.abs(dy)**2) # = magnitude
200
201
202 def fourier_der(im):
203     """
204     derivative of an image using fourier transform
205
206     Parameters
207     -----
208     :param im
209
210     Returns
211     -----
212     :return magnitude of the derivative
213
214     """
215
216     # constants
217     M, N = im.shape
218     u = np.meshgrid(np.arange(N), np.arange(M))[0] - N//2
219     v = np.meshgrid(np.arange(N), np.arange(M))[1] - M//2
220     u_der, v_der = (2 * np.pi * 1j / N), (2 * np.pi * 1j / M)
221
222     # calculate dx, dy
223     dx = u_der * IDFT2(np.fft.fftshift(u) * DFT2(im))
224     dy = v_der * IDFT2(np.fft.fftshift(v) * DFT2(im))
225
226     return np.sqrt(np.abs(dx)**2 + np.abs(dy)**2) # = magnitude
227
228
229 def gaussian_kernel_factory(kernel_size):
230     """
231     create gaussian matrix
232
233     Parameters
234     -----
235     :param kernel_size
236
237     Returns
238     -----
239     :return gaussian matrix
240
241     """
242
243     gaussian = binomial_ker = np.array([[1, 1]])
244     while gaussian.shape[1] < kernel_size: gaussian = conv(gaussian, binomial_ker)
245     gaussian_kernel = np.ones((kernel_size, kernel_size)) * gaussian * gaussian.transpose()
246
247     return 1 / gaussian_kernel.sum() * gaussian_kernel
248
249
250 def blur_spatial(im, kernel_size):
251     """
252     blur image using gaussian convolution
253
254     Parameters
255     -----
256     :param im
257
258     :param kernel_size
259
260     Returns
261     -----
262     :return blur image
263

```

```

264     """
265     return conv(im, gaussian_kernel_factory(kernel_size), mode='same')
266
267
268 def blur_fourier(im, kernel_size):
269     """
270     blur image with DFT multiply
271
272     Fourier of im & Fourier of gaussian
273     and multiply wisely
274
275     Parameters
276     -----
277     :param im
278     :param kernel_size
279
280     Returns
281     -----
282     :return blur image
283
284     """
285
286     # build the kernel with zero padding
287     kernel_base = gaussian_kernel_factory(kernel_size)
288     window = np.zeros_like(im).astype(np.float64)
289     M, N = im.shape
290     dx, dy = kernel_base.shape
291     x_middle, y_middle = N//2, M//2
292
293     window[(y_middle-dy//2):(y_middle+dy//2+1), (x_middle-dx//2):(x_middle+dx//2+1)] = kernel_base
294
295     # multiply in the freq domain
296     return IDFT2(DFT2(im) * DFT2(np.fft.ifftshift(window))).real

```