

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.lfWqDh/impr/ex2/itamakatz/presubmission/submission
2   inflating: current/answer_q1.txt
3   inflating: current/answer_q2.txt
4   inflating: current/answer_q3.txt
5   inflating: current/README
6   inflating: current/sol2.py
7 ex2 presubmission script
8
9   Disclaimer
10  -----
11  The purpose of this script is to make sure that your code is compliant
12  with the exercise API and some of the requirements
13  The script does not test the quality of your results.
14  Don't assume that passing this script will guarantee that you will get
15  a high grade in the exercise
16
17 login: ITAMAKATZ
18
19 submitted files:
20
21
22 ==== README for ex1 ===
23
24
25 List of submitted files:
26
27
28
29 README - this file
30
31 answer_qt1.txt - Answer to question Q1
32
33 answer_qt2.txt - Answer to question Q2
34
35 answer_qt3.txt - Answer to question Q3
36
37 sol2.py - python3 code
38
39
40
41
42 answer to q1:
43 Answer to question Q1:
44
45 Using a kernel in the spatial dimension is just an approximation
46 because we use a perticular kernel, while in the frequency domain we
47 are not restrained to choise of kernel but instead we use the more global
48 derivatives of the frequency domain.
49
50
51 answer to q2:
52 Answer to question Q2:
53
54 Nothing to serious :)
55
56 (spoiler alert - the proof below is pretty cool)
57
58 What happens is that the image is broken into four as if we didn't
59 Use the ifftshift but the intersection of the four blocks is not
```

60 Necessarily in the middle. This happens because like we learned
61 In class, to be able to use fft, we assume that the image is periodic.
62
63 For the more technical explanation, one of the properties of the DFT is
64 that by shifting in the spatial domain we in fact multiply the shifting
65 Amount on the exponent of the W notation. That makes a lot of sense
66 Because shifting is the main operation when computation convolution,
67 and the representation in the frequency domain is a multiplication.
68
69 Back to our subject - since the shifting is a multiplication in the
70 Frequency domain and after that we multiply the filter and the image,
71 you could easily argue that the one that was shifted in the first
72 place was not the kernel but the image!!
73
74 In 1D it would look something like this (sorry for the informality but
75 its a bit hard to write equations like this..) :
76
77
78 $\text{conv}(x[n], \text{ker}[l - m]) \leftrightarrow X[k] * \text{KER}[k] * W^{(-m)} \leftrightarrow \text{conv}(x[n - m], \text{ker}[l])$
79
80 I personally think this is unbelievable! Hot stuff.
81
82 To sum up - this means that by shifting the center of the kernel, it simply
83 Moves the center of the image we are blurring.
84
85 M.A.S.H.A.L.
86 answer to q3:
87 Answer to question Q3:
88
89 Well, the main difference is in the complexity. By using fft
90 we can get a complexity of $O(N^2 * N \log N)$ as opposed to $O(N^3)$
91 using convolution.
92
93 Another difference is that by padding the kernel in the spatial
94 domain, we are in fact interpolating the result of the kernel
95 in the frequency domain to match the dimensions. Since
96 interpolation may not be an exact representation, could lose
97 precision of the kernel.
98
99 Beside that, due to the padding done in the spatial domain, the
100 resulting image has darker edges since at those location
101 most of the matrix multiplications consist of zeros.
102
103 Lastly, there may be a difference in the result due to
104 normalization standards of the procedure. Both in the
105 transformations as well as with the weights of the kernel in
106 the transformation.
107
108
109 section 1.1
110 DFT and IDFT
111 section 1.2
112 2D DFT and IDFT
113 section 2.1
114 derivative using convolution
115 Section 2.2
116 derivative using convolution
117 Section 3.1
118 blur spatial
119 Section 3.1
120 blur fourier
121 all tests Passed.
122 - Pre-submission script done.
123
124 Please go over the output and verify that there are no failures/warnings.
125 Remember that this script tested only some basic technical aspects of your implementation
126 It is your responsibility to make sure your results are actually correct and not only
127 technically valid.

2 README

```
1 ITAMAKATZ
2
3 ==== README for ex1 ===
4
5 List of submitted files:
6
7 README - this file
8 answer_qt1.txt - Answer to question Q1
9 answer_qt2.txt - Answer to question Q2
10 answer_qt3.txt - Answer to question Q3
11 sol2.py - python3 code
```

3 answer q1.txt

```
1 Answer to question Q1:
2
3 Using a kernel in the spatial dimension is just an approximation
4 because we use a perticular kernel, while in the frequency domain we
5 are not restrained to choise of kernel but instead we use the more global
6 derivatives of the frequency domain.
```

4 answer q2.txt

```
1 Answer to question Q2:
2
3 Nothing to serious :)
4
5 (spoiler alert - the proof below is pretty cool)
6
7 What happens is that the image is broken into four as if we didn't
8 Use the ifftshift but the intersection of the four blocks is not
9 Necessarily in the middle. This happens because like we learned
10 In class, to be able to use fft, we assume that the image is periodic.
11
12 For the more technical explanation, one of the properties of the DFT is
13 that by shifting in the spatial domain we in fact multiply the shifting
14 Amount on the exponent of the W notation. That makes a lot of sense
15 Because shifting is the main operation when computation convolution,
16 and the representation in the frequency domain is a multiplication.
17
18 Back to our subject - since the shifting is a multiplication in the
19 Frequency domain and after that we multiply the filter and the image,
20 you could easily argue that the one that was shifted in the first
21 place was not the kernel but the image!!
22
23 In 1D it would look something like this (sorry for the informality but
24 its a bit hard to write equations like this..) :
25
26
27  $\text{conv}(x[n], \text{ker}[l - m]) \leftrightarrow X[k] * \text{KER}[k] * W^{(-m)} \leftrightarrow \text{conv}(x[n - m], \text{ker}[l])$ 
28
29 I personally think this is unbelievable! Hot stuff.
30
31 To sum up - this means that by shifting the center of he kernel, it simply
32 Moves the center of the image we are blurring.
33
34 M.A.S.H.A.L.
```

5 answer q3.txt

```
1 Answer to question Q3:
2
3 Well, the main difference is in the complexity. By using fft
4 we can get a complexity of  $O(N^2 * N \log N)$  as opposed to  $O(N^3)$ 
5 using convolution.
6
7 Another difference is that by padding the kernel in the spatial
8 domain, we are in fact interpolating the result of the kernel
9 in the frequency domain to match the dimensions. Since
10 interpolation may not be an exact representation, could lose
11 precision of the kernel.
12
13 Beside that, due to the padding done in the spatial domain, the
14 resulting image has darker edges since at those location
15 most of the matrix multiplications consist of zeros.
16
17 Lastly, there may be a difference in the result due to
18 normalization standards of the procedure. Both in the
19 transformations as well as with the weights of the kernel in
20 the transformation.
```

6 sol2.py

```
1  import numpy as np
2  import scipy.special
3  from scipy.misc import imread
4  from skimage.color import rgb2gray
5  from scipy.signal import convolve2d
6
7  # numerical precision to truncate using the round function
8  NUMERIC_ERROR = 13
9
10 def read_image(filename, representation):
11     # filename - file to open as image
12     # representation - is it a B&W or color image
13
14     im = imread(filename)
15     # check if it is a B&W image
16     if(representation == 1):
17         im = rgb2gray(im)
18     # convert to float and normalize
19     return im.astype(np.float32) / 255
20
21 def General_DFT(x, mult):
22     # x - array to transform
23     # mult - 1 or -1 to define if DFT or IDFT
24     # return transform rounded of numerical error
25
26     # compute vander matrix
27     vander = np.vander((np.exp(mult * 2 * np.pi * 1j * np.arange(x.shape[0]) / x.shape[0])), increasing=True)
28     # return vander.dot(x)
29     return vander.dot(x)
30
31 def General_DFT2(x, mult):
32     # x - array to transform
33     # mult - 1 or -1 to define if DFT or IDFT
34     # return transform rounded of numerical error
35
36     # compute vander matrix of both dims to perform  $v_M * x * v_N$ 
37     vander_M = np.vander((np.exp(mult * 2 * np.pi * 1j * np.arange(x.shape[0]) / x.shape[0])), increasing=True)
38     vander_N = np.vander((np.exp(mult * 2 * np.pi * 1j * np.arange(x.shape[1]) / x.shape[1])), increasing=True)
39     return np.around(vander_M.dot(x.dot(vander_N)), NUMERIC_ERROR)
40
41 def DFT(signal):
42     return General_DFT(signal, -1)
43
44 def IDFT(fourier_signal):
45     # round of numerical error and normalize
46     return np.around(General_DFT(fourier_signal, 1), NUMERIC_ERROR) / fourier_signal.shape[0]
47
48
49 def DFT2(image):
50     return General_DFT2(image, -1)
51
52 def IDFT2(image):
53     # normalize by both dimensions
54     return General_DFT2(image, 1) / (image.shape[0] * image.shape[1])
55
56 def conv_der(im):
57     # normalize and compute conv with padding
58     im = im / 255
59     xDer = convolve2d(im, np.array([[1, 0, -1]]), mode='same')
```



```

60     yDer = convolve2d(im, np.array([[1],[0],[-1]]), mode='same')
61
62     # compute power
63     return np.sqrt(np.abs(xDer)**2 + np.abs(yDer)**2)
64
65 def fourier_der(im):
66
67     # compute the frequency derivatives for multiplication
68     u, v = np.meshgrid(np.arange(-im.shape[1] / 2, im.shape[1] / 2 - 1 * im.shape[1] % 2),
69                        np.arange(-im.shape[0] / 2, im.shape[0] / 2 - 1 * im.shape[0] % 2))
70
71     # no need to normalize since using func General_DFT2
72     xDer = 2 * np.pi * 1j * General_DFT2(u * np.fft.fftshift(General_DFT2(im, -1)), 1)
73     yDer = 2 * np.pi * 1j * General_DFT2(v * np.fft.fftshift(General_DFT2(im, -1)), 1)
74
75     # compute power
76     return np.sqrt(np.abs(xDer) ** 2 + np.abs(yDer)**2)
77
78 def create_ker(kernel_size):
79     # kernel_size - odd integer
80     # returns a binomial kernel of size kernel_size X kernel_size approximating a gaussian
81     bin = scipy.special.binom(kernel_size - 1, np.arange(kernel_size)).astype(np.int64)
82     kernel = convolve2d(bin[np.newaxis, :], bin[:, np.newaxis])
83     return kernel / np.sum(kernel)
84
85 def blur_spatial (im, kernel_size):
86     return convolve2d(im, create_ker(kernel_size), mode='same')
87
88 def blur_fourier (im, kernel_size):
89     ker = create_ker(kernel_size)
90
91     center = (int(np.floor(im.shape[0] / 2)), int(np.floor(im.shape[1] / 2)))
92
93     # def ker_f to be the size of im and add it the ker
94     ker_f = im * 0
95     ker_f[np.meshgrid(
96         np.arange(center[0] - int((kernel_size - 1) / 2), center[0] + int((kernel_size - 1) / 2) + 1),
97         np.arange(center[1] - int((kernel_size - 1) / 2), center[1] + int((kernel_size - 1) / 2) + 1))] = ker
98
99     ker_f = np.fft.ifftshift(ker_f)
100     ker_f = DFT2(np.copy(ker_f))
101     im_f = DFT2(im)
102     # return of type float32
103     return (abs(IDFT2(im_f * ker_f))).astype(np.float32)

```