Contents

1 Basic Test Results

```
Archive:  /tmp/bodek.L6reHy/impr/ex2/liavst2/presubmission/submission
  inflating: current/sol2.py
  inflating: current/README
  inflating: current/answer_q3.txt
  inflating: current/answer_q2.txt
  inflating: current/answer_q1.txt
ex2 presubmission script

  Disclaimer
  ----------
  The purpose of this script is to make sure that your code is compliant
  with the exercise API and some of the requirements
  The script does not test the quality of your results.
  Don't assume that passing this script will guarantee that you will get
  a high grade in the exercise

login:  liavst2

submitted files:


========================================================

==== README for ex2: Fourier Transform & Convolution ===

========================================================



================================

=== List of submitted files ====

================================



- README - this file.

- answer_q1.txt - answer to Q1 question in the pdf.

- answer_q2.txt - answer to Q2 question in the pdf.

- answer_q3.txt - answer to Q3 question in the pdf.

- sol2.py - contains the implementation of the functions.

answer to q1:

Answer for Q1:
-------------

The derivation done by the convolution vector [1, 0, -1]
is an approximation, and might lose some data.
However, derivation done by the fourier transform gives
an exact calculation, up to a constant, and therefore
gives a much closer and exact result.

answer to q2:
```

```

Answer for Q2:
-------------

If we remove the ifftshift, the image will be cut into
pieces where every piece will be in a different place
from where it should be. This happens because we did
not shift the image itself, so its origin stayed at
the top left corner, while our kernel is centered in
the center of the image. Because the image is not shifted,
we will get its division to 4 quarters, each one located 2
places clockwise from its original place. For example, the
top right quarter will be the bottom left, etc.
The 4 quarters will be blurred, because of the Gaussian
filter.

answer to q3:

Answer for Q3:
-------------

We can vividly observe the black frame around the
output image of blur_spatial (the convolution blur),
while blur_fourier return an unframed image. This is
because of the "same" argument we add in convolve2d,
which, by its definition, returns only the central
part of the convolution.
section 1.1
DFT and IDFT
section 1.2
2D DFT and IDFT
section 2.1
derivative using convolution
Section 2.2
derivative using convolution
Section 3.1
blur spatial
Section 3.1
blur fourier
all tests Passed.
- Pre-submission script done.

  Please go over the output and verify that there are no failures/warnings.
  Remember that this script tested only some basic technical aspects of your implementation
  It is your responsibility to make sure your results are actually correct and not only
  technically valid.
```

2 README

```
liavst2

========================================================
==== README for ex2: Fourier Transform & Convolution ===
========================================================

================================
=== List of submitted files ====
================================

- README - this file.
- answer_q1.txt - answer to Q1 question in the pdf.
- answer_q2.txt - answer to Q2 question in the pdf.
- answer_q3.txt - answer to Q3 question in the pdf.
- sol2.py - contains the implementation of the functions.
```

3 answer q1.txt

```
Answer for Q1:
-------------

The derivation done by the convolution vector [1, 0, -1]
is an approximation, and might lose some data.
However, derivation done by the fourier transform gives
an exact calculation, up to a constant, and therefore
gives a much closer and exact result.
```

4 answer q2.txt

```
Answer for Q2:
-------------

If we remove the ifftshift, the image will be cut into
pieces where every piece will be in a different place
from where it should be. This happens because we did
not shift the image itself, so its origin stayed at
the top left corner, while our kernel is centered in
the center of the image. Because the image is not shifted,
we will get its division to 4 quarters, each one located 2
places clockwise from its original place. For example, the
top right quarter will be the bottom left, etc.
The 4 quarters will be blurred, because of the Gaussian
filter.
```

5 answer q3.txt

```
Answer for Q3:
-------------

We can vividly observe the black frame around the
output image of blur_spatial (the convolution blur),
while blur_fourier return an unframed image. This is
because of the "same" argument we add in convolve2d,
which, by its definition, returns only the central
part of the convolution.
```

6 sol2.py

```
##############################################################
# FILE: sol2.py
# WRITER: Liav Steinberg
# EXERCISE : Image Processing ex2
##############################################################


import numpy as np
from scipy.signal import convolve2d
from scipy.misc import imread
from skimage.color import rgb2gray


#--------------------------helpers----------------------------#

def read_image(filename, representation):
    ########################################################
    # reads an image with the given representation
    ########################################################
    image = imread(filename).astype(np.float32) / 255
    return image if representation == 2 else rgb2gray(image)


def transform_matrix(size, action):
    #########################################################
    # Helper function to calculate the dft / idft matrices
    #########################################################
    omega = np.exp(-2 * np.pi * 1J / size) if action == "dft" \
        else np.exp(2 * np.pi * 1J / size)
    index1, index2 = np.meshgrid(np.arange(size), np.arange(size))
    mat = np.power(omega, index1 * index2)
    return mat.astype(np.complex128)


def create_gauss_kernel(k_size):
    #########################################################
    # Helper function to calculate gaussian kernel
    #########################################################
    base = kernel = np.array([[1, 1]])
    for i in range(2, k_size):
        kernel = convolve2d(kernel, base).astype(np.float32)
    gauss_kernel = np.dot(kernel.reshape(k_size, 1), kernel).astype(np.float32)
    # normalize so that coefficients will sum up to 1
    gauss_kernel /= np.sum(gauss_kernel)
    return gauss_kernel


#--------------------------1.1----------------------------#

def DFT(signal):
    #########################################################
    # Performs DFT on a given signal
    #########################################################
    DFT_left_matrix = transform_matrix(signal.shape[0], "dft")
    return np.dot(DFT_left_matrix, signal)


def IDFT(fourier_signal):
    #########################################################
```

```python
    # Performs inverted DFT on a given signal
    ###########################################################
    IDFT_left_matrix = transform_matrix(fourier_signal.shape[0], "idft")
    return np.dot(IDFT_left_matrix, fourier_signal) / fourier_signal.shape[0]

#-------------------------1.2----------------------------#

def DFT2(image):
    ###########################################################
    # Performs DFT on a given image (2d matrix)
    ###########################################################
    DFT_right_matrix = transform_matrix(image.shape[1], "dft")
    return np.dot(DFT(image), DFT_right_matrix)


def IDFT2(fourier_image):
    ###########################################################
    # Performs IDFT on a given image (2d matrix)
    ###########################################################
    IDFT_right_matrix = transform_matrix(fourier_image.shape[1], "idft")
    return np.dot(IDFT(fourier_image), IDFT_right_matrix) / fourier_image.shape[1]

#-------------------------2.1----------------------------#

def conv_der(im):
    ############################################################
    # calculates the magnitude of the derivatives of a given
    # image using convolutions with the apt kernels
    ############################################################
    dx = np.array([[1, 0, -1]])
    dy = dx.reshape(3, 1)
    x_der = convolve2d(im, dx, mode="same")
    y_der = convolve2d(im, dy, mode="same")
    return np.sqrt(x_der**2 + y_der**2)

#-------------------------2.2----------------------------#

def fourier_der(im):
    ############################################################
    # calculates the magnitude of the derivatives of a given
    # image using fourier transform
    ############################################################
    # shifting coefficients properly
    rows = np.arange(-im.shape[0] / 2, im.shape[0] / 2) if not im.shape[0] % 2\
        else np.arange(-im.shape[0] / 2, im.shape[0] / 2 - 1)
    cols = np.arange(-im.shape[1] / 2, im.shape[1] / 2) if not im.shape[1] % 2\
        else np.arange(-im.shape[1] / 2, im.shape[1] / 2 - 1)
    #calculating the derivatives
    u, v = np.meshgrid(cols, rows)
    x_der = 2 * np.pi * 1J * IDFT2(u * np.fft.fftshift(DFT2(im))) / im.size
    y_der = 2 * np.pi * 1J * IDFT2(v * np.fft.fftshift(DFT2(im))) / im.size
    return np.sqrt(np.abs(x_der)**2 + np.abs(y_der)**2)

#-------------------------3.1----------------------------#

def blur_spatial(im, kernel_size):
    ############################################################
    # implements gaussian blurring on a given image using a
    # gaussian filter calculated by approximating binomial
    # coefficients, and convolving it with the image
    ############################################################
    if kernel_size == 1:
        return im
    gauss_kernel = create_gauss_kernel(kernel_size)
    return convolve2d(im, gauss_kernel, mode="same")

#-------------------------3.2----------------------------#

```

```python
def blur_fourier(im, kernel_size):
    ##########################################################
    # implements gaussian blurring on a given image using a
    # gaussian filter calculated by approximating binomial
    # coefficients, and calculating the blur using their
    # fourier transforms
    ##########################################################
    if kernel_size == 1:
        return im
    y_dim = im.shape[0]
    x_dim = im.shape[1]
    y_cen = int(np.floor(float(y_dim) / 2) + 1)
    x_cen = int(np.floor(float(x_dim) / 2) + 1)
    kernel_cen = int(np.floor(float(kernel_size) / 2) + 1)
    # calculating how much padding is needed
    padding = ((y_cen-kernel_cen+1, y_dim-y_cen-kernel_cen),
               (x_cen-kernel_cen+1, x_dim-x_cen-kernel_cen))
    # pad the kernel with zeros so it match in size with the image
    # and shift it to the (0,0) of the image
    gauss_kernel = create_gauss_kernel(kernel_size)
    pad_kernel = np.fft.ifftshift(np.pad(gauss_kernel, padding, "constant"))
    # calculating fourier transforms
    fourier_im = DFT2(im)
    fourier_kernel = DFT2(pad_kernel)
    # calculate the blur using the transforms
    blur_im = np.real(IDFT2(fourier_im * fourier_kernel))
    return blur_im.astype(np.float32)

#----------------------------end-----------------------------#
```