Contents

1 Basic Test Results

```
Archive:  /tmp/bodek.8PP2i9/impr/ex2/liran.drory/presubmission/submission
  inflating: current/answer_q2.txt
  inflating: current/answer_q3.txt
  inflating: current/README
  inflating: current/sol2.py
  inflating: current/answer_q1.txt
/usr/lib/python3/dist-packages/matplotlib/font_manager.py:280: UserWarning: Matplotlib is building the font cache using fc-l
  'Matplotlib is building the font cache using fc-list. '
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:  liran.drory

submitted files:
sol2.py

answer_q1.txt

answer_q2.txt

answer_q3.txt



answer to q1:
The difference of the magnitude happens according to the boundaries of the image

In the Fourier is cyclic

In the Convolution is not affected by the boundaries
just the frame (outter)
answer to q2:
when the gaussian not in the center the results of the blur image according to the boundry of fourier
will translate cyclic way, so that the image center will be where the center of the gaussian where.

because the gaussian is not in the center each pixel will be calculated from different region

answer to q3:
the difference between both of the blurring is the boundry conditions due to
fourier cyclic colculation (but it is very small)

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

```
liran.drory
sol2.py
answer_q1.txt
answer_q2.txt
answer_q3.txt
```

3 answer q1.txt

```
The difference of the magnitude happens according to the boundaries of the image

In the Fourier is cyclic

In the Convolution is not affected by the boundaries
just the frame (outter)
```

4 answer q2.txt

```
when the gaussian not in the center the results of the blur image according to the boundry of fourier
will translate cyclic way, so that the image center will be where the center of the gaussian where.

because the gaussian is not in the center each pixel will be calculated from different region
```

5 answer q3.txt

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

6 sol2.py

```
__author__ = "Liran Drory"

import numpy as np
from scipy.signal import convolve2d as conv
import matplotlib.pylab as plt
from scipy.misc import imread as imread
from skimage.color import rgb2gray

GRAYSCAL = 1
GRAYSCAL_MATRIX = 2
RGB = 2
RGB_MATRIX = 3
HIEGTH = 0
WIDTH = 1
SHADES_OF_GRAY = 255
COLORFULL = 3


def read_image(filename, representation):
    """
    Read Image and return matrix [0,1] float64
    Gray scale - 2D
    RGB - 3D

    Parameters
    ----------
    :param filename: str
        string containing the image filename to read (PATH)

    :param representation: int
        either 1 or 2 defining whether the output
        should be a greyscale image (1) or an RGB image (2).

    Returns
    -------
    :return numpy array with either 2D matrix or 3D matrix
            describing the pixels of the image

    """

    # loads the image
    im = imread(filename)

    if representation == RGB:
        im_float = im.astype(np.float64)     # pixels to float
        im_float /= 255                      # pixels [0,1]
        return im_float

    if representation == GRAYSCAL:
        im_g = im.astype(np.float64)       # pixels to float
        im_g = rgb2gray(im_g)                 # turn to grey
        return im_g


def DFT(signal):
    """
    Function that return DFT of signal
    if matrix is input: return every row DFT

```

```
    Parameters
    ----------
    :param signal

    Returns
    -------
    :return complex_fourier_signal

    """

    # find the length of the signal
    N = signal.shape[0]
    if signal.ndim == 2:
        M, N = signal.shape

    # calculate DFT matrix
    u, v  = np.meshgrid(np.arange(N), np.arange(N))
    omega = np.exp(-2 * np.pi * 1j / N)
    dft_matrix = np.power(omega, u*v)

    # if it is a matrix of signals
    if signal.ndim == 2:
        # calculate the Fourier Transform
        complex_fourier_signal = np.dot(dft_matrix, signal.transpose())
        return complex_fourier_signal.transpose()

    # calculate the Fourier Transform
    complex_fourier_signal = np.dot(dft_matrix, signal)
    return complex_fourier_signal


def IDFT(fourier_signal):
    """
    Function that return IDFT of fourier signal
    if matrix is input: return every row IDFT

    Parameters
    ----------
    :param fourier_signal

    Returns
    -------
    :return 1/N * signal

    """

    # find the length of the signal
    N = fourier_signal.shape[0]
    if fourier_signal.ndim == 2:
        M, N = fourier_signal.shape

    # calculate IDFT matrix
    u, v  = np.meshgrid(np.arange(N), np.arange(N))
    omega = np.exp(2 * np.pi * 1j / N)
    idft_matrix = np.power(omega, u*v)

    # if it is a matrix of fourier signals
    if fourier_signal.ndim == 2:
        # calculate the Fourier Transform
        signal = np.dot(idft_matrix, fourier_signal.transpose())
        return 1/N * signal.transpose()

    # calculate the inverse Fourier Transform
    signal = np.dot(idft_matrix, fourier_signal)
    return 1/N * signal


def DFT2(image):
```

```
    """
    Function that return 2D DFT of image

    Parameters
    ----------
    :param image (matrix)

    Returns
    -------
    :return fourier_image

    """

    M, N = image.shape

    # build the dft2_matrix transform
    omega_y = np.exp(-2 * np.pi * 1j / M)
    u, v = np.meshgrid(np.arange(M), np.arange(M))
    dft2_matrix = np.power(omega_y, u*v)

    # calculate the 2D fourier transform
    fourier_image = np.dot(dft2_matrix, DFT(image))

    return fourier_image


def IDFT2(fourier_image):
    """
    Function that return 2D IDFT of an image

    Parameters
    ----------
    :param fourier_image

    Returns
    -------
    :return image

    """

    M, N = fourier_image.shape
    # build the idft2_matrix transform
    omega_y = np.exp(2 * np.pi * 1j / M)
    u, v = np.meshgrid(np.arange(M), np.arange(M))
    idft2_matrix = np.power(omega_y, u*v)

    # calculate the 2D inverse fourier transform
    return 1/M * np.dot(idft2_matrix, IDFT(fourier_image))


def conv_der(im):
    """
    derivative of an image using convolution

    Parameters
    ----------
    :param im

    Returns
    -------
    :return magnitude of the derivative

    """

    # set der x/y matrix
    der_x = np.array([[1, 0, -1]])
    der_y = np.array(der_x.transpose())
    # calculate the derivative to x and y
```

```python
    dx = conv(im, der_x, mode='same')
    dy = conv(im, der_y, mode='same')

    return np.sqrt(np.abs(dx)**2 + np.abs(dy)**2)  # = magnitude


def fourier_der(im):
    """
    derivative of an image using fourier transform

    Parameters
    ----------
    :param im

    Returns
    -------
    :return magnitude of the derivative

    """

    # constants
    M, N = im.shape
    u = np.meshgrid(np.arange(N), np.arange(M))[0] - N//2
    v = np.meshgrid(np.arange(N), np.arange(M))[1] - M//2
    u_der, v_der = (2 * np.pi * 1j / N), (2 * np.pi * 1j / M)

    # calculate dx, dy
    dx = u_der * IDFT2(np.fft.fftshift(u) * DFT2(im))
    dy = v_der * IDFT2(np.fft.fftshift(v) * DFT2(im))

    return np.sqrt(np.abs(dx)**2 + np.abs(dy)**2)  # = magnitude


def gaussian_kernel_factory(kernel_size):
    """
    create gaussian matrix

    Parameters
    ----------
    :param kernel_size

    Returns
    -------
    :return gaussian matrix

    """

    gaussian = binomial_ker = np.array([[1, 1]])
    while gaussian.shape[1] < kernel_size: gaussian = conv(gaussian, binomial_ker)
    gaussian_kernel = np.ones((kernel_size, kernel_size)) * gaussian * gaussian.transpose()

    return 1 / gaussian_kernel.sum() * gaussian_kernel


def blur_spatial(im, kernel_size):
    """
    blur image using gaussian convolution

    Parameters
    ----------
    :param im

    :param kernel_size

    Returns
    -------
    :return blur image

```

```python
    """
    return conv(im, gaussian_kernel_factory(kernel_size), mode='same')


def blur_fourier(im, kernel_size):
    """
    blur image with DFT multiply

    Fourier of im & Fourier of gaussian
    and multiply wisely

    Parameters
    ----------
    :param im
    :param kernel_size

    Returns
    -------
    :return blur image

    """

    # build the kernel with zero padding
    kernel_base = gaussian_kernel_factory(kernel_size)
    window = np.zeros_like(im).astype(np.float64)
    M, N = im.shape
    dx, dy = kernel_base.shape
    x_middle, y_middle = N//2, M//2

    window[(y_middle-dy//2):(y_middle+dy//2+1), (x_middle-dx//2):(x_middle+dx//2+1)] = kernel_base

    # multiply in the freq domain
    return IDFT2(DFT2(im) * DFT2(np.fft.ifftshift(window))).real
```