Image Processing Application – Spring 2021

Lecturer: Professor Lee Guee Sang

Student: Le Hoai Duy – 218493

**Homework 5**

1. Select a gray scale image and present FFT results (before and after FFT Shift):

**Code:**

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img = cv.imread('./lena.png',0)

f = np.fft.fft2(img)

FFT\_magnitude\_spectrum = np.log(np.abs(f))

fshift = np.fft.fftshift(f)

FFT\_shift\_magnitude\_spectrum = np.log(np.abs(fshift))

fig, axes = plt.subplots(ncols=3, nrows=1,figsize=(30, 10))

ax0, ax1, ax2 = axes.flat

ax0.imshow(img, cmap = 'gray')

ax0.set\_title('Input Image', fontsize=24)

ax0.axis('off')

ax1.imshow(FFT\_magnitude\_spectrum, cmap = 'gray')

ax1.set\_title('FFT Magnitude Spectrum', fontsize=24)

ax1.axis('off')

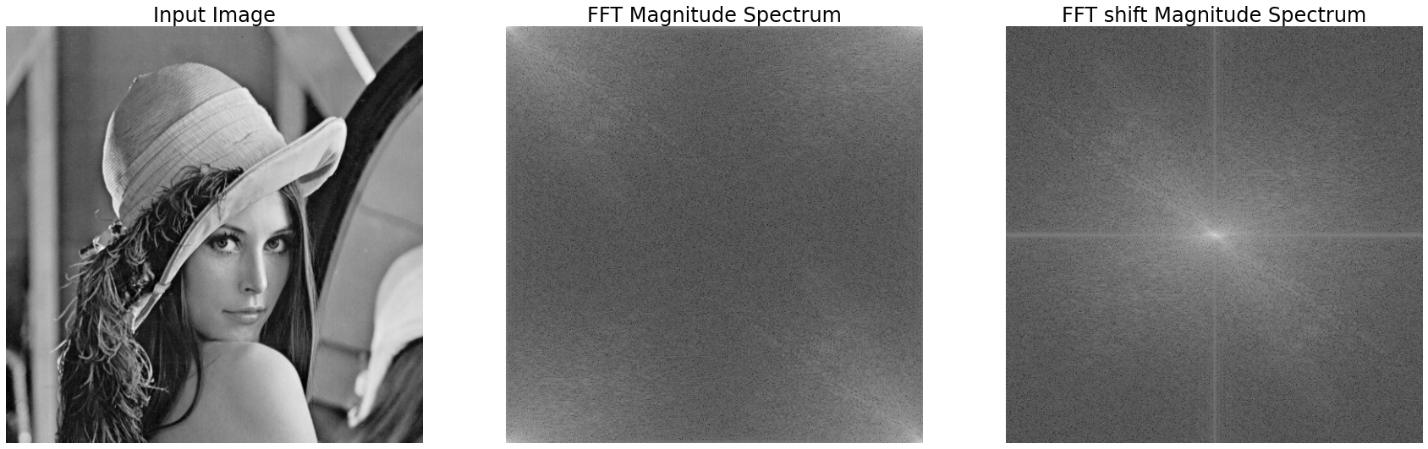
ax2.imshow(FFT\_shift\_magnitude\_spectrum, cmap = 'gray')

ax2.set\_title('FFT shift Magnitude Spectrum', fontsize=24)

ax2.axis('off')

fig.savefig("D:/result\_HW5\_1.png",bbox\_inches='tight')

**Result:**

****

**Figure 1. FFT and FFT Shift result of a grayscale image**

High frequency area

Low frequency area

In term of FFT result, low frequency components located at the corners and high frequency areas located at the center of the coefficients. In contrast, FFT Shift make the low frequency signals located in the center of the image.

2. Select a grayscale image and perform

1) Ideal, Butterworth, Gaussian low pass filtering

2) Ideal, Butterworth, Gaussian high pass filtering

Input image:



**Code:**

import cv2

import numpy as np

'''

opencv Demonstration of high-pass and low-pass filtering of ideal filtering, Butterworth filtering and Gaussian filtering

Filter Parameters window: Filter parameters window

-d0: filter size D0

-flag: filter type

    0-ideal filtering

    1-Butterworth filtering

    2-Gaussian filtering

-n: order of Butterworth filtering

-lh: low-pass filtering or high-pass filtering

'''

def fft(img):

    '''Fourier transform the image and return the frequency matrix after transposition'''

    assert img.ndim == 2, 'img should be gray.'

    rows, cols = img.shape[:2]

         # Calculate the optimal size

    nrows = cv2.getOptimalDFTSize(rows)

    ncols = cv2.getOptimalDFTSize(cols)

         # According to the new size, create a new transformed image

    nimg = np.zeros((nrows, ncols))

    nimg[:rows, :cols] = img

         # Fourier transform

    fft\_mat = cv2.dft(np.float32(nimg), flags=cv2.DFT\_COMPLEX\_OUTPUT)

         # Transposition, the low frequency part moves to the middle, the high frequency part moves to the surrounding

    return np.fft.fftshift(fft\_mat)

def fft\_image(fft\_mat):

    '''Convert frequency matrix to visual image'''

         # Add 1 to the log function to avoid log (0).

    log\_mat = cv2.log(1 + cv2.magnitude(fft\_mat[:, :, 0], fft\_mat[:, :, 1]))

         # Standardized to between 0 ~ 255

    cv2.normalize(log\_mat, log\_mat, 0, 255, cv2.NORM\_MINMAX)

    return np.uint8(np.around(log\_mat))

def ifft(fft\_mat):

    '''Inverse Fourier transform, return inverse transform image'''

         # Reverse transposition, the low frequency part moves to the surrounding, the high frequency part moves to the middle

    f\_ishift\_mat = np.fft.ifftshift(fft\_mat)

         # Inverse Fourier Transform

    img\_back = cv2.idft(f\_ishift\_mat)

         # Convert complex number to amplitude, sqrt (re ^ 2 + im ^ 2)

    img\_back = cv2.magnitude(\*cv2.split(img\_back))

         # Standardized to between 0 ~ 255

    cv2.normalize(img\_back, img\_back, 0, 255, cv2.NORM\_MINMAX)

    return np.uint8(np.around(img\_back))

def fft\_distances(m, n):

    '''

    Calculate the distance of each point of the m, n matrix from the center

    See page 93 of "Digital Image Processing MATLAB Edition. Gonzalez"

    '''

    u = np.array([i if i <= m / 2 else m - i for i in range(m)],

                 dtype=np.float32)

    v = np.array([i if i <= m / 2 else m - i for i in range(m)],

                 dtype=np.float32)

    v.shape = n, 1

         # The distance from each point to the upper left corner of the matrix

    ret = np.sqrt(u \* u + v \* v)

         # The distance of each point from the center of the matrix

    return np.fft.fftshift(ret)

def lpfilter(flag, rows, cols, d0, n):

    '''

    Low-pass filter

    @param flag: filter type

    0-ideal low-pass filtering

    1-Butterworth low-pass filtering

    2-Gaussian low-pass filtering

    @param rows: the height of the filtered matrix

    @param cols: the width of the filtered matrix

    @param d0: filter size D0

    @param n: order of Butterworth low-pass filtering

    @return filter matrix

    '''

    assert d0 > 0, 'd0 should be more than 0.'

    filter\_mat = None

         # Ideal low-pass filtering

    if flag == 0:

        filter\_mat = np.zeros((rows, cols, 2), np.float32)

        cv2.circle(filter\_mat, (int(rows / 2), int(cols / 2)),

                   d0, (1, 1, 1), thickness=-1)

         # Butterworth low-pass filtering

    elif flag == 1:

        duv = fft\_distances(\*fft\_mat.shape[:2])

        filter\_mat = 1 / (1 + np.power(duv / d0, 2 \* n))

                 # fft\_mat has 2 channels, real and imaginary

                 # fliter\_mat also requires 2 channels

        filter\_mat = cv2.merge((filter\_mat, filter\_mat))

         # Gaussian low-pass filtering

    else:

        duv = fft\_distances(\*fft\_mat.shape[:2])

        filter\_mat = np.exp(-(duv \* duv) / (2 \* d0 \* d0))

                 # fft\_mat has 2 channels, real and imaginary

                 # fliter\_mat also requires 2 channels

        filter\_mat = cv2.merge((filter\_mat, filter\_mat))

    return filter\_mat

def hpfilter(flag, rows, cols, d0, n):

    '''

    High-pass filter

    @param flag: filter type

    0-ideal high-pass filtering

    1-Butterworth high-pass filtering

    2-Gaussian high-pass filtering

    @param rows: the height of the filtered matrix

    @param cols: the width of the filtered matrix

    @param d0: filter size D0

    @param n: the order of Butterworth high-pass filtering

    @return filter matrix

    '''

    assert d0 > 0, 'd0 should be more than 0.'

    filter\_mat = None

         # Ideal high-pass filtering

    if flag == 0:

        filter\_mat = np.ones((rows, cols, 2), np.float32)

        cv2.circle(filter\_mat, (int(rows / 2), int(cols / 2)),

                   d0, (0, 0, 0), thickness=-1)

         # Butterworth high-pass filtering

    elif flag == 1:

        duv = fft\_distances(rows, cols)

         # duv has a value of 0 (the center is 0 from the center). To avoid division by 0, set the center to 0.000001

        duv[int(rows / 2), int(cols / 2)] = 0.000001

        filter\_mat = 1 / (1 + np.power(d0 / duv, 2 \* n))

                 # fft\_mat has 2 channels, real and imaginary

                 # fliter\_mat also requires 2 channels

        filter\_mat = cv2.merge((filter\_mat, filter\_mat))

         # Gaussian high-pass filtering

    else:

        duv = fft\_distances(\*fft\_mat.shape[:2])

        filter\_mat = 1 - np.exp(-(duv \* duv) / (2 \* d0 \* d0))

                 # fft\_mat has 2 channels, real and imaginary

                 # fliter\_mat also requires 2 channels

        filter\_mat = cv2.merge((filter\_mat, filter\_mat))

    return filter\_mat

if \_\_name\_\_ == '\_\_main\_\_':

    filter\_list = ["Ideal","Butterworth","Gaussian"]

    img = cv2.imread('./lena.png', 0)

    fft\_mat = fft(img)

    # Lowpass filtering

    fig, ax = plt.subplots(ncols=3, nrows=1,figsize=(30, 10))

    for i in range(3):

        filter\_mat\_lp = lpfilter(i, fft\_mat.shape[0], fft\_mat.shape[1], 10, 1)

        filtered\_mat\_lp = filter\_mat\_lp \* fft\_mat

        filtered\_lp\_img = ifft(filtered\_mat\_lp)

        ax[i].imshow(filtered\_lp\_img, cmap = 'gray')

        ax[i].set\_title(filter\_list[i] + 'Lowpass Filtered Image', fontsize=24)

        ax[i].axis('off')

    fig.savefig("D:/result\_HW5\_2.png",bbox\_inches='tight')

    # Highpass filtering

    fig, ax = plt.subplots(ncols=3, nrows=1,figsize=(30, 10))

    for i in range(3):

        filter\_mat\_hp = hpfilter(i, fft\_mat.shape[0], fft\_mat.shape[1], 10, 1)

        filtered\_mat\_hp = filter\_mat\_hp \* fft\_mat

        filtered\_hp\_img = ifft(filtered\_mat\_hp)

        ax[i].imshow(filtered\_hp\_img, cmap = 'gray')

        ax[i].set\_title(filter\_list[i] + 'Highpass Filtered Image', fontsize=24)

        ax[i].axis('off')

    fig.savefig("D:/result\_HW5\_3.png",bbox\_inches='tight')

**Results:**

****



**Figure 3.** Result of Ideal, Butterworth, Gaussian low pass filter with Do = 10 (above) and D0 = 20 (below)





**Figure 3.** Result of Ideal, Butterworth, Gaussian high pass filter with D0 = 10 (above) and D0 = 20 (below)

**Analysis of result:**

* Low Pass Filtering blurs the image.
* High Pass Filtering is an edge detection operation. It makes the details of the image more obvious.
* Ringing effect can be obviously seen in Ideal Filters (both Lowpass and Highpass). This phenomenon is greatly reduced in Butterworth filters and Gaussian filter (both Lowpass and Highpass).