Image Processing Application – Spring 2021

Lecturer: Professor Lee Guee Sang

Student: Le Hoai Duy – 218493

**Homework 6**

1. Implement homomorphic filtering in frequency domain with Python. Test the code with more than three samples. Discuss the differences.

**Code:**

import os

import cv2

import numpy as np

from PIL import Image

from matplotlib import pyplot as plt

def fft\_distances(m, n):

    #calculate the distance of each point of the m, n matrix from the center

    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 <= n / 2 else n - i for i in range(n)],

                 dtype=np.float32)

    u.shape = m, 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 homomorphic\_filter(img, rl=0.4, rh=2, c=5, d0=20):

    # compute ln(img)

    img = np.log1p(np.float64(img), dtype=np.float64)

    #compute fft and fft\_shift

    img\_fft = np.fft.fft2(img, axes=(0,1))

    img\_fftshift = np.fft.fftshift(img\_fft)

    #compute mask filter

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

    filter\_mat = (rh - rl) \* (1 - np.exp(-c \* (duv \* duv) / (d0 \* d0))) + rl

    #apply filter by multiple filter\_mat and fftshift

    dst\_fftshift = np.multiply(img\_fftshift,filter\_mat)

    #compute inverse fft

    dst\_ifftshift = np.fft.ifftshift(dst\_fftshift)

    dst\_ifft = np.fft.ifft2(dst\_ifftshift)

    #combine complex real and imaginary components to form (the magnitude for) the original image again

    dst = np.real(dst\_ifft)

    #compute exponential

    dst = np.exp(dst, dtype=np.float64)

    img\_homomorphic = cv2.normalize(dst, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

    return img\_homomorphic

if \_\_name\_\_ == "\_\_main\_\_":

     path = "./CTI.png"

     img = np.array(Image.open(path).convert('L'))

     img\_new = homomorphic\_filter(img, rl=0.25, rh=1.5, c=5, d0=20)

     fig, axes = plt.subplots(ncols=2, nrows=1,figsize=(12, 10))

     axes[0].imshow(img, cmap = 'gray')

     axes[0].set\_title('Input Image', fontsize=24)

     axes[0].axis('off')

     axes[1].imshow(img\_new, cmap = 'gray')

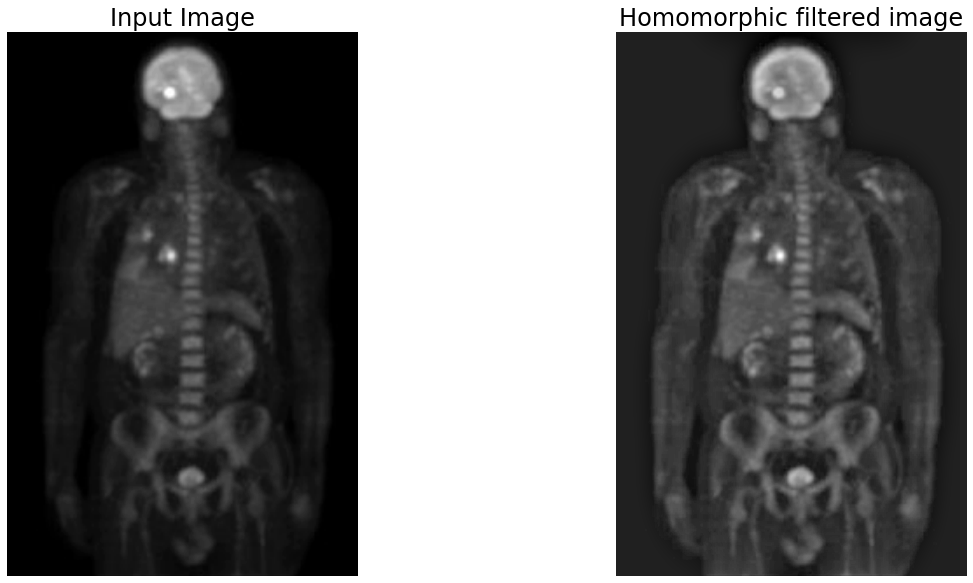
     axes[1].set\_title('Homomorphic filtered image', fontsize=24)

     axes[1].axis('off')

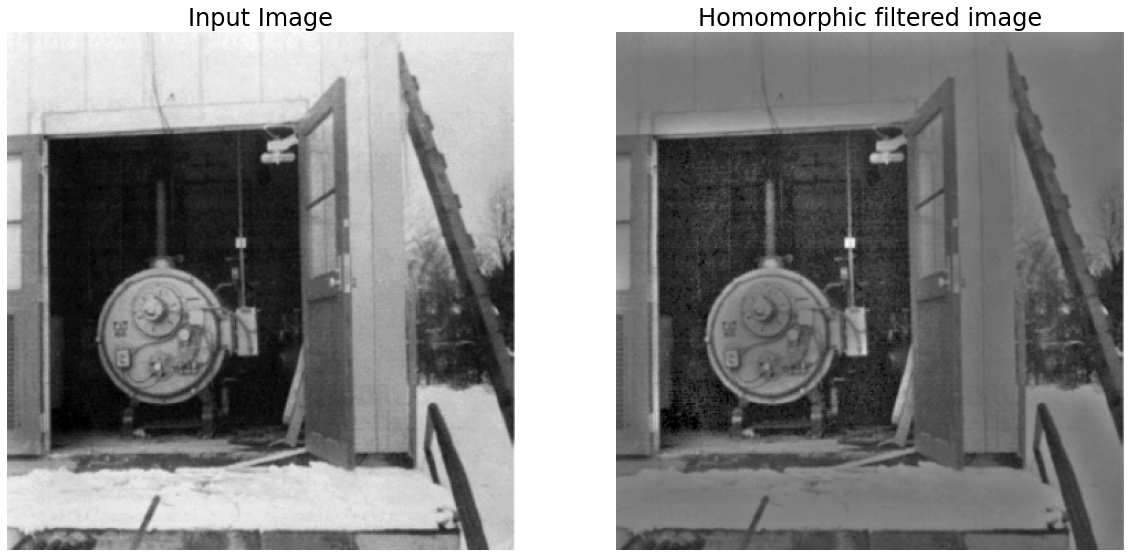
     fig.savefig("D:/result\_HW6\_1.png",bbox\_inches='tight')

The homomorphic function is obtained by using a slightly modified form of the Gaussian High Pass Filter function:

**Result:**

****

**Figure 1.** Filtered image obtained by the above function with rl=0.3, rh=0.75, c=5, d0=15. The original image is slightly burred and some parts are obscured, specially at arm part. The homomorphic function (obtained by using a slightly modified form of the Gaussian High Pass Filter function) enhances the hidden parts and improves the image's contrast. More details in the image can be seen.



**Figure 2.** Filtered image obtained by the above function with rl=0.3, rh=0.75, c=5, d0=15. More details in the room can be seen.

****

**Figure 3.** Filtered image obtained by the above function with rl=0.75, rh=1.25, c=5, d0=20

2. Implement sharpening with Laplacian, unsharp masking and high boost filtering in frequency domain with Python. Test the code with more than three samples. Discuss the differences.

**Code:**

import os

import cv2

import numpy as np

from PIL import Image

from matplotlib import pyplot as plt

import math

def fft\_distances(m, n):

    #calculate the distance of each point of the m, n matrix from the center

    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 <= n / 2 else n - i for i in range(n)],

                 dtype=np.float32)

    u.shape = m, 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 laplacian\_filter(img):

    #compute fft and fft\_shift

    img\_fft = np.fft.fft2(img, axes=(0,1))

    img\_fftshift = np.fft.fftshift(img\_fft)

    #compute mask filter

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

    filter\_mat = 4\*(math.pi\*math.pi)\*(duv \* duv)

    #apply filter by multiple filter\_mat and fftshift

    dst\_fftshift = np.multiply(img\_fftshift,filter\_mat)

    #compute inverse fft

    dst\_ifftshift = np.fft.ifftshift(dst\_fftshift)

    dst\_ifft = np.fft.ifft2(dst\_ifftshift)

    #combine complex real and imaginary components to form (the magnitude for) the original image again

    dst = np.abs(dst\_ifft)

    delta = cv2.normalize(dst, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

    laplacian = img - delta

    return laplacian

def unsharp\_filter(img,k):

    #compute fft and fft\_shift

    img\_fft = np.fft.fft2(img, axes=(0,1))

    img\_fftshift = np.fft.fftshift(img\_fft)

    #compute mask filter

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

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

    #apply filter by multiple filter\_mat and fftshift

    dst\_fftshift = np.multiply(img\_fftshift,filter\_mat)

    #compute inverse fft

    dst\_ifftshift = np.fft.ifftshift(dst\_fftshift)

    dst\_ifft = np.fft.ifft2(dst\_ifftshift)

    #combine complex real and imaginary components to form (the magnitude for) the original image again

    dst = np.abs(dst\_ifft)

    img\_unsharp = cv2.normalize(dst, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

    return img\_unsharp

if \_\_name\_\_ == "\_\_main\_\_":

     path = "./moon.jpg"

     img = np.array(Image.open(path).convert('L'))

     #Laplacian

     img\_new\_laplacian = laplacian\_filter(img)

     #unsharp

     img\_new\_unsharp = unsharp\_filter(img,1)

     #high-boot

     img\_new\_highboot = unsharp\_filter(img,3)

     fig, axes = plt.subplots(ncols=2, nrows=1,figsize=(20, 10))

     axes[0].imshow(img, cmap = 'gray')

     axes[0].set\_title('Input Image', fontsize=24)

     axes[0].axis('off')

     axes[1].imshow(img\_new\_laplacian, cmap = 'gray')

     axes[1].set\_title('Laplacian filtered image', fontsize=24)

     axes[1].axis('off')

     fig.savefig("D:/result\_HW6\_4.png",bbox\_inches='tight')

     fig, axes = plt.subplots(ncols=2, nrows=1,figsize=(20, 10))

     axes[0].imshow(img, cmap = 'gray')

     axes[0].set\_title('Input Image', fontsize=24)

     axes[0].axis('off')

     axes[1].imshow(img\_new\_unsharp, cmap = 'gray')

     axes[1].set\_title('Unsharp filtered image', fontsize=24)

     axes[1].axis('off')

     fig.savefig("D:/result\_HW6\_5.png",bbox\_inches='tight')

     fig, axes = plt.subplots(ncols=2, nrows=1,figsize=(20, 10))

     axes[0].imshow(img, cmap = 'gray')

     axes[0].set\_title('Input Image', fontsize=24)

     axes[0].axis('off')

     axes[1].imshow(img\_new\_highboot, cmap = 'gray')

     axes[1].set\_title('High-boot filtered image', fontsize=24)

     axes[1].axis('off')

     fig.savefig("D:/result\_HW6\_6.png",bbox\_inches='tight')

**Results:**







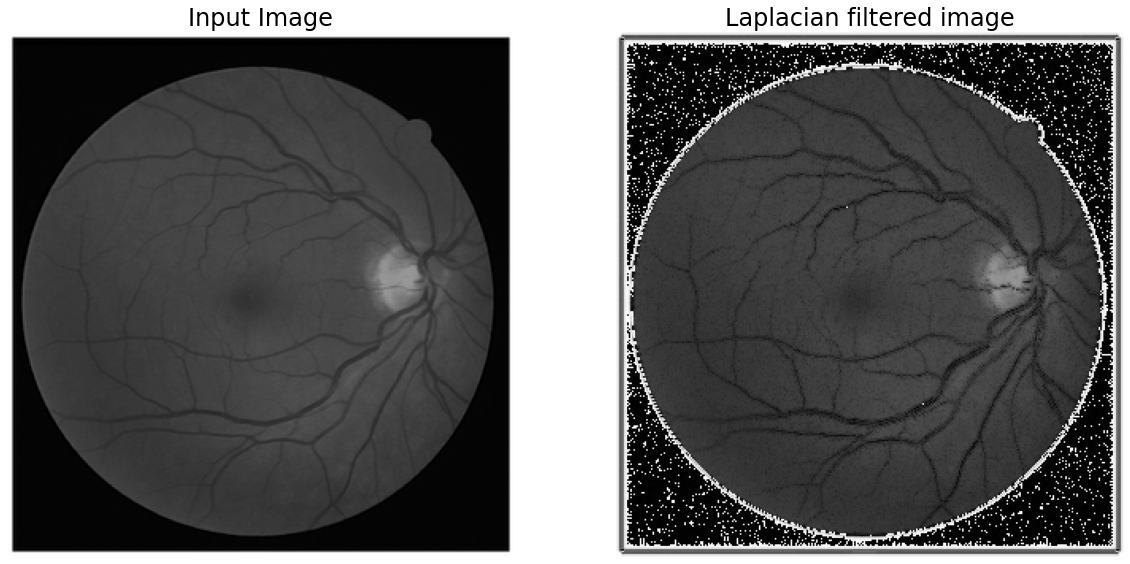
**Figure 4.** Images enhanced using Laplacian, unsharp masking and high-boot in the frequency domain

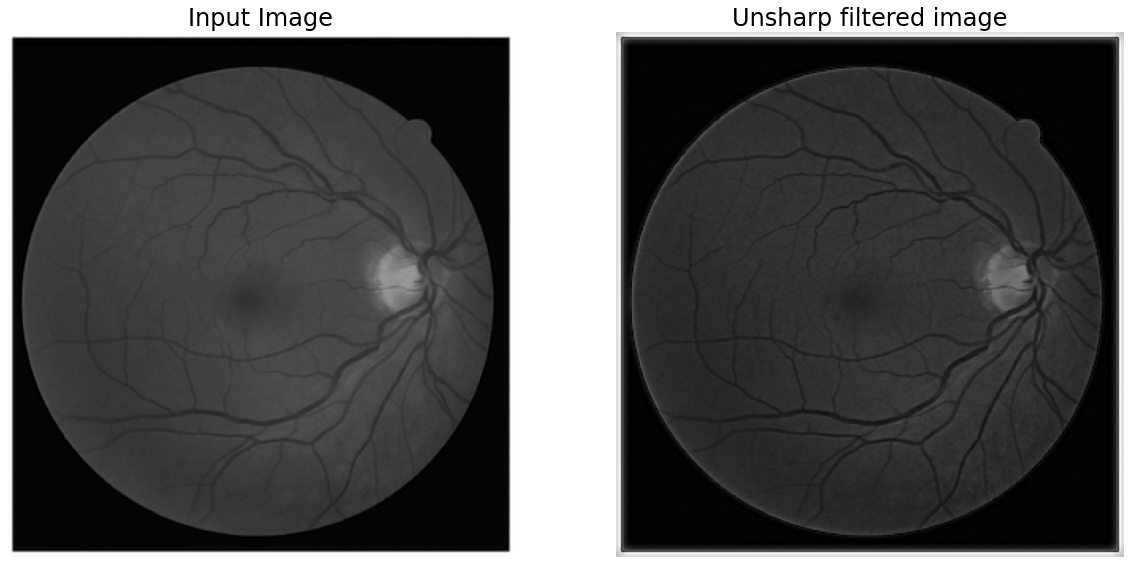


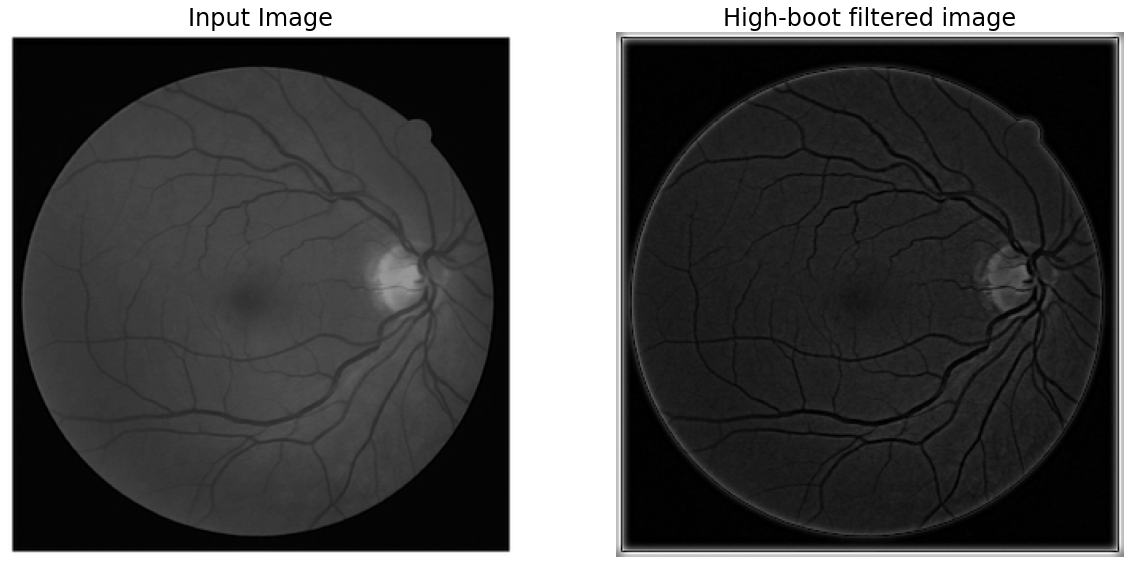




**Figure 5.** Images enhanced using Laplacian, unsharp masking and high-boot in the frequency domain







**Figure 6.** Images enhanced using Laplacian, unsharp masking and high-boot in the frequency domain

**Analysis the result:** All 3 methods Laplacian mask, unsharp mask and high boost filtering in frequency domain increase the contrast of the image as well as sharpen the edges, same with in spatial domain.