CS-663 Assignment 2 Q1

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

September 29, 2020

1 (15 points) Image Sharpening

Input images:

- 1/data/superMoonCrop.mat
- 1/data/lionCrop.mat

Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes. Write code for image sharpening using unsharp masking and apply it to both the input images. To compare the original and filtered images, linearly contrast-stretch them to the same intensity range, say, [0, 1] . Tune the parameters (Gaussian standard-deviation parameter and the scaling parameter) to your best judgment, but such that the sharpening in the image is clearly visible. You may use the following Matlab functions: *fspecial()* and *imfilter()*.

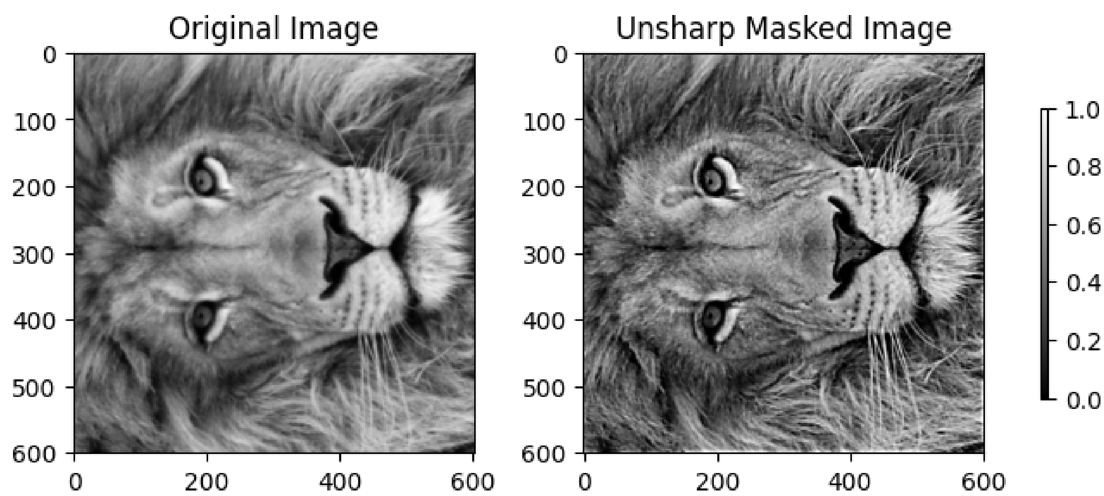
- Write a function *myUnsharpMasking.m* to implement this.
- For each image, show the original and sharpened versions side by side, using the same (gray) colormap.
- Report the tuned parameter values for each image.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from numpy import zeros,zeros_like,array
from numpy import linspace,pi,sqrt,e,power,outer
from math import floor,ceil

from myLinearContrastStretching import myLinearContrastStretching

import h5py

def read_file(filename):
    f = h5py.File(filename,"r")
    out = f.get('imageOrig')
    out = array(out)

    return (out*255.0/np.max(out))

## gaussian filtering
def dnorm(x,mu,sigma):
    """
    Calculate pdf of the gaussian distribution with mean=mu
    and standard deviation = sigma
    input : x(point), mu(mean), sigma(standard deviation)
    ouptut :  pdf of the gaussian distribution at the point x
    """
    return 1 / (sqrt(2 * pi) * sigma) * e ** (-power((x - mu) / sigma, 2) / 2)

def gaussian_kernel(ksize,mu=0,sigma=1,verbose=False):
    """
```

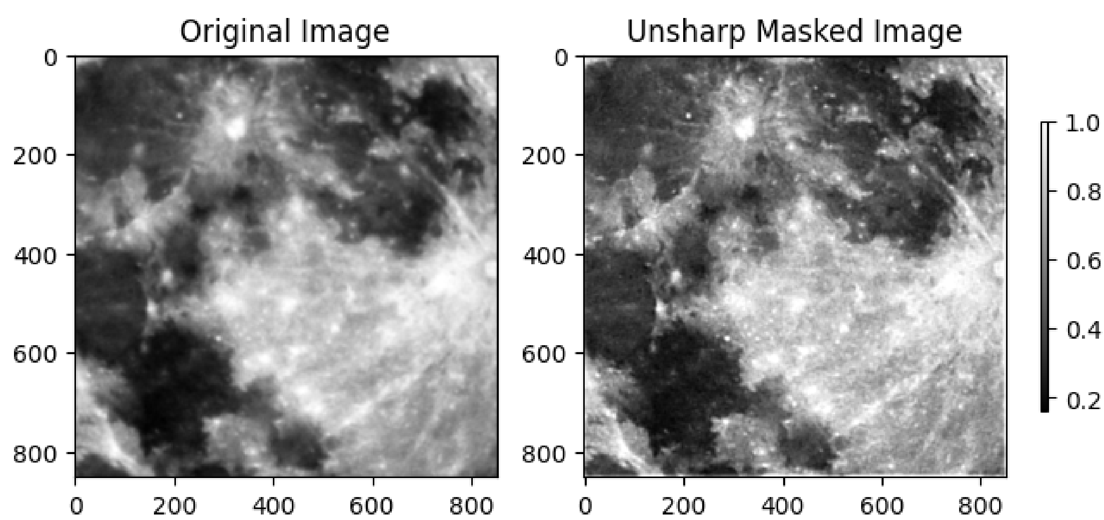
```python
    Create a normalized gaussian kernel with the given kernel size
    and standard deviation (sigma)
    inputs : ksize(for a ksizexksize gaussian filter),
             sigma(standard deviation, default=1)
             mu(mean of gaussian, default=0)
             verbose (to visualize the gaussian kernel)
    output : gaussian ksizexksize kernel filter
    """
    # create the 1-D gaussian kernel
    kernel_1D = linspace(-(ksize // 2), ksize // 2, ksize)
    for i in range(ksize):
        kernel_1D[i] = dnorm(kernel_1D[i], mu, sigma)

    # computers outer product of two 1-D gaussian kernels
    # to produce a 2D Gaussian Kernel
    kernel_2D = outer(kernel_1D.T, kernel_1D.T)
    kernel_2D *= 1.0 / kernel_2D.max()

    if verbose==True:
        plt.figure()
        plt.imshow(kernel_2D,cmap="gray",interpolation="none")
        plt.title("{}x{} Gaussian Kernel".format(ksize,ksize))
        plt.savefig("../images/GaussKernel_{}x{}.png".format(ksize,ksize),
                    bbox_inches="tight",pad=-1)

    return kernel_2D

def truncate(array):
    """
    if any pixel has value > 255 this makes it 255
    and if any pixel is <0 this makes it 0
    """
    r,c = array.shape
    for i in range(r):
        for j in range(c):
            if array[i,j]>255:
                array[i,j] = 255
            elif array[i,j]<0:
                array[i,j] = 0
    return array

def convolution(filename,input_image, kernel, average=False, verbose=False):
    """
    Calculates the convolution of input image with filter kernel
    after zero-padding with the required no. of pixels
    CAN BE USED WITH ANY KERNEL FILTER OF ANY SIZE
    input : image_file : input image file_path
            kernel : the filter kernel
            average : required only if the filter kernel is not normalized (default = False)
            verbose : to show and save the plots (default = False)
    output : the normalized output image after convolution
    Presently, the code works only for grayscale images, the color component will be added.
    """
    # READING THE INPUT IMAGE
    image = input_image.copy()
    name = filename.split("/")[-1].split(".")[0]

    # EXTRACTING THE IMAGE AND KERNEL SHAPES AND INITIALIZING OUTPUT
    image_row, image_col = image.shape
    kernel_row, kernel_col = kernel.shape
    output = zeros(image.shape)

    # CREATING ZERO-PADDED IMAGE
    pad_height = int((kernel_row - 1) / 2)
    pad_width = int((kernel_col - 1) / 2)
```

```
    padded_image = zeros((image_row + (2 * pad_height), image_col +
                         (2 * pad_width)))
    padded_image[pad_height:padded_image.shape[0] - pad_height,
              pad_width:padded_image.shape[1] - pad_width] = image

    # CONVOLUTION OPERATION DONE HERE
    for row in range(image_row):
        for col in range(image_col):
            output[row, col] = np.sum(kernel * padded_image[row:row +
                                   kernel_row, col:col + kernel_col])
            if average:
                output[row, col] /= (kernel_row * kernel_col)
    output = (output/np.max(output)) *255.0

    # SAVE THE PLOTS IF VERBOSE
    if verbose:
        fig,axes = plt.subplots(1,2, constrained_layout=True)
        axes[0].imshow(image,cmap='gray')
        axes[0].axis("on")
        axes[0].set_title("Original Image")
        im = axes[1].imshow(output, cmap='gray')
        axes[1].axis("on")
        axes[1].set_title("Gaussian Blur using {}X{} Kernel".format(kernel_row,
                           kernel_col))
        cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.35)
        #plt.show()
        plt.savefig("../images/"+name+"GaussBlur.png",bbox_inches="tight",
                    pad=-1)

        plt.imsave("../images/"+name+"GaussianBlur{}X{}Kernel.png".format(kernel_row,
        kernel_col),output,cmap="gray")

    return output

def gaussian_blur(filename,input_image, kernel_size, verbose=False):
    #sigma = sqrt(kernel_size)
    # this sigma is used by OpenCV implementation but explanation is not given
    sigma = 0.3*((kernel_size-1)*0.5 - 1) + 0.8
    #sigma = 2.0
    kernel = gaussian_kernel(kernel_size, sigma= sigma, verbose=verbose)
    return convolution(filename,input_image, kernel, average=False, verbose=False)

def plot_images(filename,alpha,kernel,input_image,output_image,cmap="gray"):

    name = filename.split("/")[-1].split(".")[0]

    fig,axes = plt.subplots(1,2, constrained_layout=True)
    axes[0].imshow(input_image/np.max(input_image),cmap=cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image")

    im = axes[1].imshow(output_image/np.max(output_image), cmap=cmap)
    axes[1].axis("on")
    axes[1].set_title("Unsharp Masked Image")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.35)
    plt.savefig("../images/"+name+str(alpha)+"_"+str(kernel)+"UnsharpMask.png",
                bbox_inches="tight",pad=-1)
    plt.cla()


def laplacian(filename,image):
    #out = zeros_like(image)
    #r,c = image.shape
    kernel = array([[0,-1,0],[-1,4,-1],[0,-1,0]])
```

```python
    out = convolution(filename,image, kernel, average=False, verbose=False)
    return out

def unSharpMask(filename,kernel_size,alpha,verbose=True):
    image = read_file(filename)
    gaussianBlurred = gaussian_blur(filename,image,kernel_size,verbose=verbose)
    log = gaussianBlurred
    # log = truncate(laplacian(filename,gaussianBlurred))
    sharp = truncate((1+alpha)*image - alpha*log)
    #image = myLinearContrastStretching(filename,image,[0,np.max(image)],[0,1])
    #sharp = myLinearContrastStretching(filename,sharp,[0,np.max(sharp)],[0,1])
    plot_images(filename,alpha,kernel_size,image,sharp)
```



[a]



[b]

Figure 1: (a) LionCrop Image Unsharp Masking with alpha=1.6 and GaussianStd = 2.6 (b) superMoonCrop Image Unsharp Masking with alpha=2.2 and GaussianStd = 3.2

Tuned Parameters

For the *lionCrop.mat* image , we found the optimum scaling parameter(α) = 1.6 and the Gaussian Standard deviation = 2.6.

For the *superMoonCrop.mat* image , we found the optimum scaling parameter(α) = 2.2 and the Gaussian Standard deviation = 3.2.

myMainScript.py :

```python
from myUnsharpMasking import unSharpMask

files = ["../data/superMoonCrop.mat","../data/lionCrop.mat"]
for file_name in files:
    if "lion" in file_name:
        alpha = 1.6
        kernel = 15
    elif "superMoon" in file_name:
        alpha = 2.2
        kernel = 19

    unSharpMask(file_name,kernel,alpha)
```