CS-663 Assignment 2 Q2

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

September 29, 2020

2 (30 points) Edge-preserving Smoothing using Bilateral Filtering.

Input images:

1. *2/data/barbara.mat*
2. *2/data/grass.png*
3. *2/data/honeyCombReal.png*

Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes.
Corrupt the image with independent and identically-distributed additive zero-mean Gaussian noise with standard deviation set to 5% of the intensity range. Note: in *Matlab*, *randn()* gives random numbers drawn independently from a Gaussian with mean 0 and standard deviation 1 .
Write code for bilateral filtering (standard "slow" algorithm is also fine) and apply it (one pass over all pixels) to all the input images. For efficiency in Matlab, the code should, ideally, have maximum 2 "for" loops to go over the rows and columns of the image. At a specific pixel "p", the data collection with a window, weight computations, and weighted averaging can be performed without using loops.
Define the root-mean-squared difference ($RMSD$) as the square root of the average, over all pixels, of the squared difference between a pixel intensity in the original image and the intensity of the corresponding q pixel in the filtered image, i.e., given 2 images A and B with N pixels each, $RMSD(A, B) := \sqrt{(1/N)\sum_p (A(p) - B(p))^2}$ where A(p) is the intensity of pixel p in image A .
Tune the parameters (standard-deviations for Gaussians over space and intensity) to minimize the RMSD between the filtered and the original image.

- Write a function *myBilateralFiltering.m* to implement this.
- Show the original, corrupted, and filtered versions side by side, using the same (gray) colormap.
- Show the mask for the spatial Gaussian, as an image.
- Report the optimal parameter values found, say σ^*_{space} and $\sigma^*_{intensity}$, along with the optimal $RMSD$.
- Report $RMSD$ values for filtered images obtained with
  1. $0.9*\sigma^*_{space}$ and $\sigma^*_{intensity}$
  2. $1.1*\sigma^*_{space}$ and $\sigma^*_{intensity}$
  3. σ^*_{space} and $0.9*\sigma^*_{intensity}$
  4. σ^*_{space} and $1.1*\sigma^*_{intensity}$, with all other parameter values unchanged.

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import zeros,zeros_like,exp,roll
from numpy.random import normal
import h5py

np.random.seed(0)
```

```python

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

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

def truncate(array):
    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 add_noise(image):
    out = image.copy()
    noise = normal(size=image.shape,scale=0.05*np.max(image))
    return truncate(out+noise)

def filter_bilateral(input_image, sigma_spatial, sigma_intensity):
    # make a simple Gaussian function taking the squared radius
    gaussian = lambda r2, sigma: exp(-0.5*r2/sigma**2 )

    # define the window width to be the 2 time the spatial std. dev. to
    # be sure that most of the spatial kernel is actually captured
    win_width = int(3*sigma_spatial +1)

    wgt_sum = zeros_like(input_image)
    result  = zeros_like(input_image)

    for shft_x in range(-win_width,win_width+1):
        for shft_y in range(-win_width,win_width+1):
            # compute the spatial contribution
            spatial = gaussian(shft_x**2+shft_y**2, sigma_spatial )

            # shift by the offsets to get image window
            window = roll(input_image, [shft_y, shft_x], axis=[0,1])

            # compute the intensity contribution
            combined_filter = spatial*gaussian( (window-input_image)**2, sigma_intensity )

            # result stores the mult. between combined filter and image window
            result += window*combined_filter
            wgt_sum += combined_filter

    # normalize the result and return
    plt.imsave("../images/GaussianMask_"+str(sigma_spatial)+"_"+
               str(sigma_intensity) + ".png" ,wgt_sum,cmap="gray")
    return result/wgt_sum

def plot_images(filename,ssp,sint,input_image,noisy_image,output_image,cmap="gray"):

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

    fig,axes = plt.subplots(1,3, constrained_layout=True)
    axes[0].imshow(input_image/np.max(input_image),cmap=cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image",fontsize=12)

    axes[1].imshow(noisy_image/np.max(noisy_image),cmap=cmap)
    axes[1].axis("on")
```

```
    axes[1].set_title("Noisy Image",fontsize=12)

    im = axes[2].imshow(output_image/np.max(output_image), cmap=cmap)
    axes[2].axis("on")
    axes[2].set_title("Bilateral Filtered Image",fontsize=12)

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

def RMSD(A,B):
    r,c = A.shape
    A = A/np.max(A)
    B = B/np.max(B)
    total = np.sum(np.square(A-B))
    return np.sqrt(total/(r*c))

def myBilateralFiltering(filename,sigma_spatial,sigma_intensity):
    name = filename.split("/")[-1].split(".")[0]
    if not filename.endswith(".mat"):
        image = read_file(filename)
    else:
        image = cv2.imread(filename,0)

    noisy_image = addnoise(image)

    bilateral = filter_bilateral(noisy_image,sigma_spatial,sigma_intensity)
    plot_images(filename,sigma_spatial,sigma_intensity,image,noisy_image,bilateral)
    print("RMSD of {} image for Sigma_spatial: {} and Sigma_intensity: {} is
          {}".format(name,sigma_spatial,sigma_intensity,RMSD(image,bilateral)))
```

RMSD Calculations :

- 2/data/barbara.mat
  The optimum values of parameters found are :
  - $\sigma^*_{space} = 2.5$
  - $\sigma^*_{intensity} = 25$
  - RMSD of barbara image for σ^*_{space}= 2.5 and $\sigma^*_{intensity}$= 25 is 0.033.

  RMSD calculations for the other values specified:

  1. RMSD of barbara image for Sigma Spatial(0.9*σ^*_{space}): 2.25 and Sigma intensity($\sigma^*_{intensity}$): 25 is 0.034.
  2. RMSD of barbara image for Sigma spatial(1.1*σ^*_{space}): 2.75 and Sigma intensity ($\sigma^*_{intensity}$): 25 is 0.034.
  3. RMSD of barbara image for Sigma spatial(σ^*_{space})): 2.5 and Sigma intensity ($\sigma^*_{intensity}$)): 22.5 is 0.055.
  4. RMSD of barbara image for Sigma spatial(σ^*_{space}): 2.5 and Sigma intensity (1.1*$\sigma^*_{intensity}$): 27.5 is 0.034.

- 2/data/grass.png
  The optimum values of parameters found are :
  - $\sigma^*_{space} = 1.35$
  - $\sigma^*_{intensity} = 25$
  - RMSD of grass image for σ^*_{space}: 1.5 and $\sigma^*_{intensity}$: 25 is 0.032.

  RMSD calculations for the other values specified:

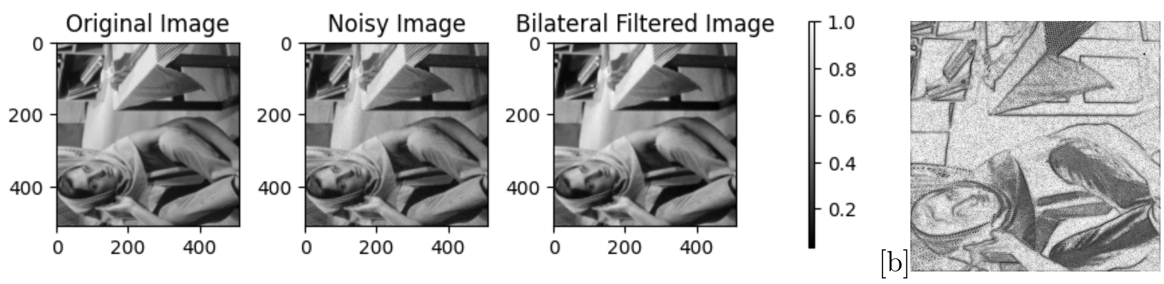
  1. RMSD of grass image for Sigma spatial 0.9*(σ^*_{space}): 1.35 and Sigma intensity($\sigma^*_{intensity}$): 25 is 0.036
  2. RMSD of grass image for Sigma spatial 1.1*(σ^*_{space}): 1.65 and Sigma intensity($\sigma^*_{intensity}$): 25 is 0.033

3. RMSD of grass image for Sigma spatial (σ^*_{space}): 1.5 and Sigma intensity 0.9*($\sigma^*_{intensity}$): 22.5 is 0.034.
4. RMSD of grass image for Sigma spatial (σ^*_{space}): 2.5 and Sigma intensity 1.1*($\sigma^*_{intensity}$): 27.5 is 0.041.

- 2/data/honeyCombReal.png
  The optimum values of parameters found are :
  - $\sigma^*_{space} = 1.5$
  - $\sigma^*_{intensity} = 25$
  - RMSD of honeyCombReal image for σ^*_{space}: 1.5 and $\sigma^*_{intensity}$: 25 is 0.031.

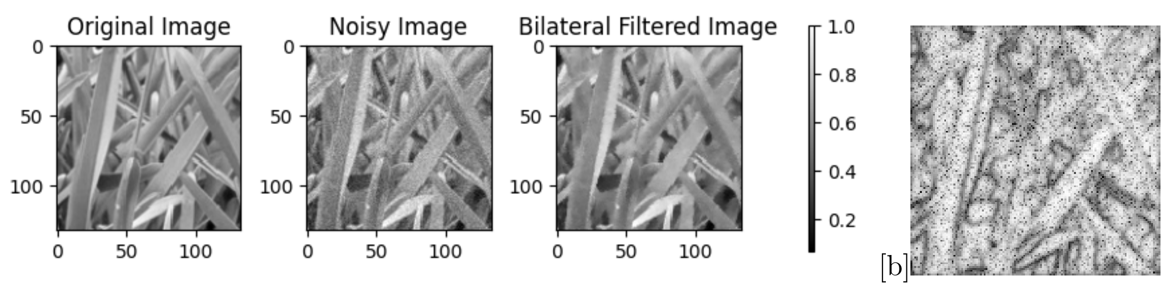
  RMSD calculations for the other values specified:

  1. RMSD of honeyCombReal image for Sigma spatial 0.9*(σ^*_{space}) : 1.35 and Sigma intensity ($\sigma^*_{intensity}$: 25 is 0.033.
  2. RMSD of honeyCombReal image for Sigma spatial 1.1*(σ^*_{space}) : 1.65 and Sigma intensity ($\sigma^*_{intensity}$) : 25 is 0.032.
  3. RMSD of honeyCombReal image for Sigma spatial (σ^*_{space}: 1.5 and Sigma intensity 0.9*(σ^*_{space}): 22.5 is 0.032.
  4. RMSD of honeyCombReal image for Sigma spatial (σ^*_{space}): 2.5 and Sigma intensity 1.1*(σ^*_{space}): 27.5 is 0.034.



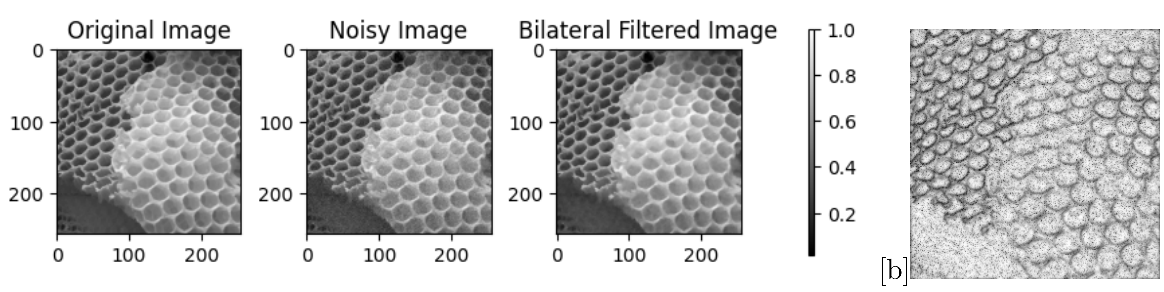
[a] [b]

Figure 1: (a) barbara image Bilateral Filter with σ^*_{space}=2.5 and $\sigma^*_{intensity} = 25$ (b) barbara image Bilateral Filter Gaussian Mask



[a] [b]

Figure 2: (a) grass image Bilateral Filter with σ^*_{space}=1.5 and $\sigma^*_{intensity} = 25$ (b) grass image Bilateral Filter Gaussian Mask



[a] [b]

Figure 3: (a) honeyCombReal image Bilateral Filter with σ^*_{space}=1.5 and $\sigma^*_{intensity} = 25$ (b) honeyCombReal image Bilateral Filter Gaussian Mask

All images for the other spatial gaussian standard deviations and intensity gaussian standard deviations are present in the *images* folder in the submission zip.

myMainScript.py

```python
from myBilateralFiltering import myBilateralFiltering
from time import time

start = time()
files = ["../data/barbara.mat","../data/grass.png","../data/honeyCombReal.png"]

for file_name in files:
    if "barbara" in file_name:
        combinations = [(2.25,25),(2.5,22.5),(2.5,25),(2.5,27.5),(2.75,25)]
    elif "grass" in file_name:
        combinations = [(1.35,25),(1.5,22.5),(1.65,25),(1.5,27.5),(1.5,25)]
    elif "honey" in file_name:
        combinations = [(1.35,25),(1.5,22.5),(1.65,25),(1.5,27.5),(1.5,25)]
    for sigma_s,sigma_int in combinations:
        myBilateralFiltering(file_name,sigma_s,sigma_int)

end = time()
print("Total time taken : ",(end-start)/60,"minutes")
```

To test the **cartoonization** concept learnt in class, we took an image of Tom Cruise, the famous Hollywood actor and iterated the bilateral filtering continuously for 4times. The results seem to be quite convincing. The following image would make the things clear.

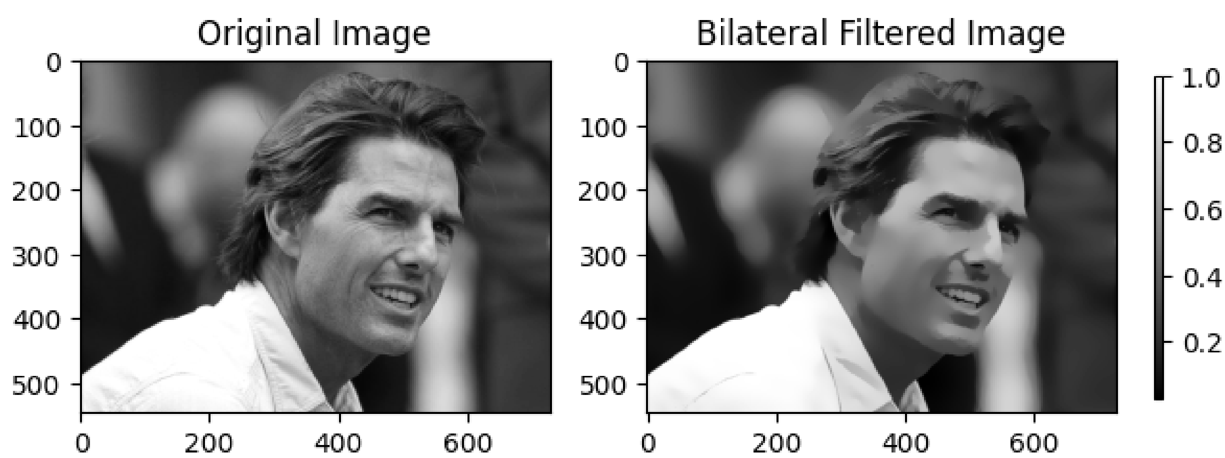


Figure 4: Tom Cruise Cartoonify comparison with σ^*_{space}=5 and $\sigma^*_{intensity} = 10$

Cartoonizing Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import zeros,zeros_like,exp,roll,array
from numpy.random import normal
import h5py
import cv2

np.random.seed(0)

def filter_bilateral(filename,input_image, sigma_spatial, sigma_intensity):
    """
    Performs standard bilateral filtering of an input image. If padding is desired,
    img_in should be padded prior to calling

    inputs:- input_image       (ndarray) input image
           - sigma_spatial      (float)   spatial gaussian standard deviation
           - sigma_intensity      (float)   value gaussian standard. deviation
    outputs:-result      (ndarray) output bilateral-filtered image
    """

    # make a simple Gaussian function taking the squared radius
```

```python
    gaussian = lambda r2, sigma: exp(-0.5*r2/sigma**2 )

    # define the window width to be the 2 time the spatial std. dev. to
    # be sure that most of the spatial kernel is actually captured
    win_width = int(3*sigma_spatial +1)

    wgt_sum = zeros_like(input_image)
    result  = zeros_like(input_image)

    for shft_x in range(-win_width,win_width+1):
        for shft_y in range(-win_width,win_width+1):
            # compute the spatial contribution
            spatial = gaussian(shft_x**2+shft_y**2, sigma_spatial )

            # shift by the offsets to get image window
            window = roll(input_image, [shft_y, shft_x], axis=[0,1])

            # compute the intensity contribution
            combined_filter = spatial*gaussian( (window-input_image)**2, sigma_intensity )

            # result stores the mult. between combined filter and image window
            result += window*combined_filter
            wgt_sum += combined_filter

    # normalize the result and return
    return result/wgt_sum

def plot_images(filename,ssp,sint,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",fontsize=12)

    im = axes[1].imshow(output_image/np.max(output_image), cmap=cmap)
    axes[1].axis("on")
    axes[1].set_title("Bilateral Filtered Image",fontsize=12)

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.35)
    plt.savefig("../images/"+name+str(ssp)+"_"+str(sint)+"Bilateral.png",
                bbox_inches="tight",pad=-1)
    plt.imsave("../images/"+name+str(ssp)+"_"+str(sint)+"BilateralOnly.png",
                output_image/np.max(output_image),cmap=cmap)
    plt.cla()


def myBilateralFiltering(filename,sigma_spatial,sigma_intensity):
    name = filename.split("/")[-1].split(".")[0]
    image = cv2.imread(filename)
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    output = filter_bilateral(filename,array(image,dtype="float32"),sigma_spatial,
                              sigma_intensity)
    for _ in range(3):
        output= filter_bilateral(filename,output,sigma_spatial,sigma_intensity)

    plot_images(filename,sigma_spatial,sigma_intensity,image,output)


if __name__=="__main__":
    filename = "../data/tom_cruise.jpg"
    myBilateralFiltering(filename,5,10)
```