Assignment 1 – Smoothing techniques

-Ashrith Chandramouli (aschandr@syr.edu)

Brief instructions :

The results for this assignment are generated from the python script – smoothing.py

The script does not require any parameters to run and can be run using the command

“python smoothing.py” from the command prompt in the directory of the script. The script is developed on Python 3.7.2 and I have included the following libraries which are necessary to run the script

```
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage
from math import sqrt,pi
```

Once the script is run, the results are displayed in a window. The result of each smoothing technique is shown in its own window. Closing an opened window will initiate the script to show the result of next smoothing technique.

All the output images have also been included in the zip file for reference.

The following pages of this report will explain my process and observations.

I have observed that the input image NoisyImage1 contains Gaussian noise which has a light tint to the image. NoisyImage2 contains salt and pepper/impulsive noise.

Task a) Average smoothing by using mean filter of 2 different kernel sizes

For this task, I've made use of two functions, one which uses a convolution function written by me and another which uses the convolve function provided by scipy.ndimage library. The functions are as shown below

```
# Manual mean filter to show the working of convolution
# Takes in the size of the kernel and input image as parameters
def mean_filter_manual(size,img):
    correction = size//2
    kernel = np.divide(np.ones((size,size)),size*size)
    res = np.copy(img)

    for i in range(correction,len(img)-correction):
        for j in range(correction,len(img[0])-correction):
            sum = 0

            subImg = img[i-correction:i+correction+1,j-
correction:j+correction+1]

            for x in range(size):
                for y in range(size):
                    sum += kernel[x][y]*subImg[x][y]
            res[i-1][j-1] = int(sum)

    return res
```

Mean filtering using a convolution process written by me

```
# Mean filter using the convolution function from ndimage which is faster
# Takes in the size of the kernel and input image as parameters
def mean_filter_auto(size,img):
    kernel = np.divide(np.ones((size,size)),size*size)
    res = ndimage.convolve(img,kernel)
    return res
```

Mean filtering using scipy.ndimage.convolve

To obtain the output images, I have made use of the library function for faster processing. The gist of convolution can be understood through the convolution code written by me. I have made use of 2 kernel sizes - 3 and 5 for this task. The resulting images after performing the filtering can be seen as shown below

Gaussian noise image

Impulsive noise image

Mean filter with size 3

Mean filter with size 3

Gaussian noise image

Impulsive noise image

Mean filter with size 5

Mean filter with size 5

Observations : Mean filter is fairly successful in removing the gaussian noise as observed by less tinting of the image. The salt and pepper noise is just attenuated and diffused and does not remove the noise very well. From the images above, we can observe that as the kernel size increases, the noise is filtered better but there is also a loss in the quality of the image due to the blurring effect. In the examples I've taken, using a filter with kernel size 5 is better since it better removes salt and pepper noise.

Task b) Gaussian smoothing by applying two different standard deviation values

For this task, I've made use of 3 functions – first to apply the gaussian function, second to create a gaussian kernel and third to perform gaussian smoothing. The functions are as shown

```
# Gaussian function
# Takes in the step, mean and standard deviation as parameters
def gaussian(x, mu, sig):
    return 1./(sqrt(2.*pi)*sig)*np.exp(-np.power((x - mu)/sig, 2.)/2)
```

Gaussian function

```
# Function to create a gaussian kernel
# Size of the kernel is determined by the standard deviation value - Size = 5
* standard deviation
# Takes in the standard deviation as parameter
def gaussian_kernel(sigma):
    size = int(5 * sigma)
    size = size+1 if size%2 == 0 else size
    limit = size//2
    #The range we pick values from must be equal to 5 times sigma
    gaus_range = [x for x in range(-limit,limit+1)]
    kernel = [gaussian(x,0,sigma) for x in gaus_range]
    normalized_kernel = [x/kernel[limit] for x in kernel]
    print(normalized_kernel)
    total = sum(normalized_kernel)
    gauss_kernel = [x/total for x in normalized_kernel]
    print(gauss_kernel)
    return np.array(gauss_kernel).reshape(1,size)
```

Gaussian kernel function

```
# Applies a gaussian filter to an image and returns the filtered image
# Takes in standard deviation and input image as parameters
def gaussian_filter(sigma,img):
    kernel = gaussian_kernel(sigma)
    print(kernel)
    res = ndimage.convolve(img,kernel)
    kernel = np.transpose(kernel)
    res = ndimage.convolve(img,kernel)
    return res
```

Gaussian smoothing/filter function

The gaussian function provides the necessary values for the kernel. The kernel function builds a 1D kernel according to the SEPAR_FILTER algorithm. This kernel is then used for convolution in the gaussian filter function. I have made use of 2 standard deviation values – 1.4 and 1.8. The resulting images are as shown below

Gaussian noise image

Impulsive noise image

Gaussian filter with sigma 1.4

Gaussian filter with sigma 1.4

Gaussian noise image

Impulsive noise image

Gaussian filter with sigma 1.8

Gaussian filter with sigma 1.8

Observations : Gaussian filter is successful in removing the gaussian noise but does not perform well for impulsive noise. Increasing the standard deviation results in better removal of gaussian noise but there isn't much improvement for impulsive noise. The quality of image is well retained for gaussian smoothing. In the examples I have taken, a standard deviation of 1.8 is better than 1.4 since the gaussian noise is filtered more.

Task c) Median filtering by using two different kernel sizes

For this task, I've implemented two functions. One function uses a manual convolution written by me and the other function uses the scipy.ndimage.convolve function as shown

```
# Applies a median filter with manual convolution and returns the filtered ima
ge
# Takes in the size of kernel and input image as parameters
def median_filter_manual(size,img):
    correction = size//2
    kernel = np.zeros((size,size))
    res = np.copy(img)
    for i in range(correction,len(img)-correction):
        for j in range(correction,len(img[0])-correction):
            kernel = img[i-correction:i+correction+1,j-
correction:j+correction+1]
            res[i,j] = np.median(kernel)

    return res
```

Median filter with manual convolution

```
# Applies a median filter using ndimage library which is much faster and retur
ns the filtered image
# Takes in the size of the kernel and input image as parameters
def median_filter_auto(sz,img):
    res = ndimage.median_filter(img,size = sz)
    return res
```

Median filter using scipy.ndimage.convolve

I have used two kernel sizes – 3 and 5 for the median filter. The manual convolution is slower than the convolve function provided by the scipy library but gives a better representation of how the convolution actually occurs. The results of running the filter are as shown below

Gaussian noise image

Impulsive noise image

Median filter with size 3

Median filter with size 3

Gaussian noise image

Impulsive noise image

Median filter with size 5

Median filter with size 5

Observations : The median filter is an excellent filter for removing impulsive/salt and pepper noise. This is due to the nature of choosing the median in the kernel where the extremities are filtered out. The gaussian noise is also removed fairly well by this smoothing. As we increase the kernel size, we can observe that the salt and pepper noise is almost completely removed but the increase in kernel size decreases the sharpness of the image. In the examples I've chosen, a kernel size of 5 is better since it almost completely removes the salt and pepper noise while also removing some gaussian noise better than kernel size 3.

Conclusions :

Comparing all three smoothing techniques, we can observe that the gaussian smoothing performs best for removing gaussian noise but worst for removing impulsive noise. For removing salt and pepper noise, the median smoothing is a clear winner and is almost equal or a little worse than average smoothing. Averaging falls in between for both gaussian and impulsive noise. The quality of image is best retained in gaussian smoothing followed by median smoothing and averaging smoothing at the end. Averaging blurs out the image and as we increase the size of the kernel, we lose much of the data. Gaussian smoothing best retains the original image followed by median smoothing which loses sharpness and averaging comes last.

Gaussian smoothing performs best for removing gaussian noise but comes close with averaging. Gaussian smoothing is the winner here since it retains the original image much better than averaging due to the nature of the gaussian kernel. The gaussian kernel gives a higher weightage to the pixel that is being convolved compared to averaging where every pixel has equal weightage during convolution. This also means that noise can get diffused more on an averaging filter compared to a gaussian filter. A median filter will not be able to remove noise since the gaussian noise may fall in the median range

Median smoothing performs the best for impulsive noise due to the fact that impulsive noise will most definitely not fall into the median range during convolution. Most impulsive noise have values nearing 0 or 255 which falls in the extremities while calculating median. Due to this property, a median smoothing is the best choice for removing impulsive noise. Averaging only diffuses the noise into nearby pixels as does a gaussian filter.