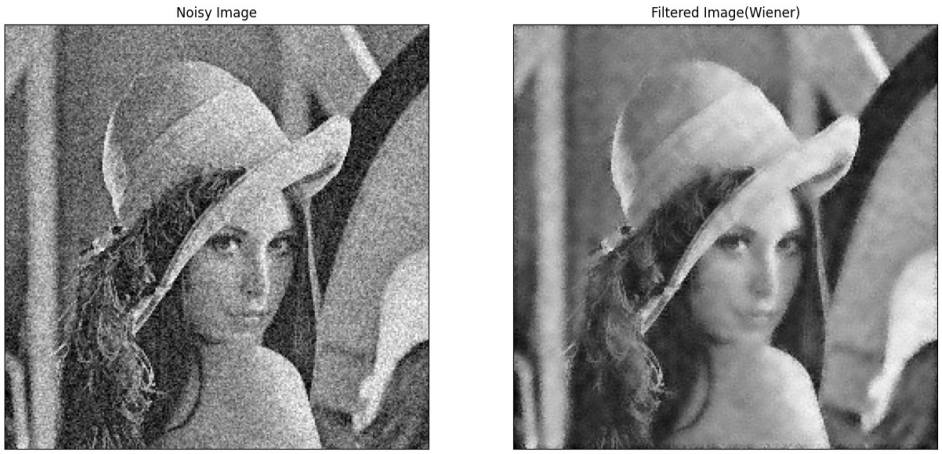
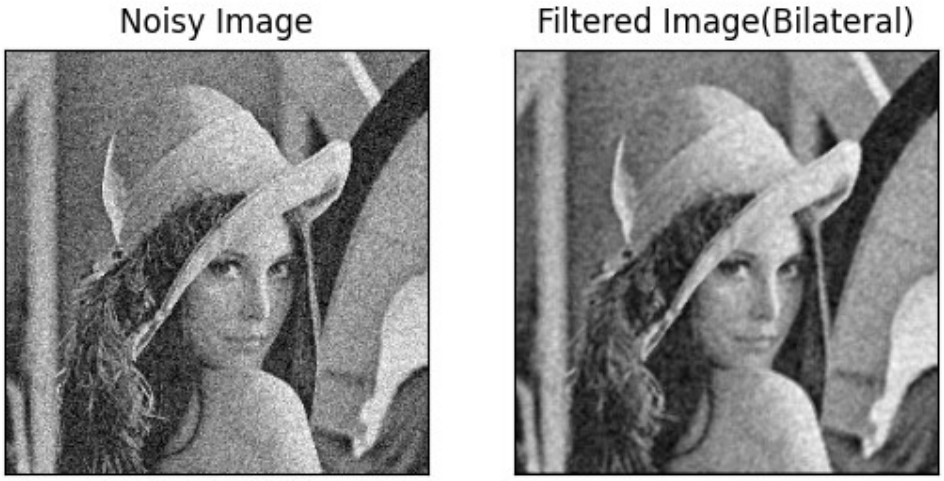
Assignment 4
2017113547 이정근



WienerFilter.py

```
import cv2
import numpy as np
from numpy import fft
import matplotlib.pyplot as plt


def add_gaussian(image, sigma):
    noise = np.random.normal(0, sigma, image.shape)

    # Add the Gaussian noise to the image
    output = np.clip(image + noise, 0, 255)
    output = output.astype('uint8')

    return output


in_image_lena = cv2.imread('lena_gray.png', 0)

noisy_image = add_gaussian(in_image_lena, 0.1*255)

height, width = in_image_lena.shape
recovered_image = np.zeros((height, width))
```

```
padded_image = np.pad(noisy_image, ((3, 3), (3, 3)), 'constant')
height, width = padded_image.shape

vn = np.var(noisy_image)

for i in range(3, height-3):
    for j in range(3, width-3):
        m = np.mean(padded_image[i-3:i+4, j-3:j+4])
        vg = np.var(padded_image[i-3:i+4, j-3:j+4])
        recovered_image[i-3, j-3] = m + vg/(vg+vn)*(padded_image[i][j]-m)

plt.subplot(121), plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(recovered_image, cmap='gray')
plt.title('Filtered Image(Wiener)'), plt.xticks([]), plt.yticks([])
plt.show()
```



BilateralFilter.py

```
import cv2
import numpy as np
from numpy import fft
import matplotlib.pyplot as plt

def add_gaussian(image, sigma):
    noise = np.random.normal(0, sigma, image.shape)

    # Add the Gaussian noise to the image
    output = np.clip(image + noise, 0, 255)
    #output = output.astype('uint8')

    return output


in_image_lena = cv2.imread('lena_gray.png', 0)
```

```
noisy_image = add_gaussian(in_image_lena, 0.1*255)

height, width = in_image_lena.shape
recovered_image = np.zeros((height, width))

padded_image = np.pad(noisy_image, ((1, 1), (1, 1)), 'constant')
height, width = padded_image.shape


def pre_weight(x, y, k, l, sigmaX, sigmaY, sigmaR, g):
    return  np.exp(-(x-k)**2/(2*(sigmaX**2))-(y-l)**2/(2*(sigmaY**2))-(g[x,y]-g[k,l])**2/(2*(sigmaR**2)))


def sum_pre_weight(x, y, sigmaX, sigmaY, sigmaR, g):
    sum =0
    for k in range(x-1, x+2):
        for l in range(y-1, y+2):
            sum += pre_weight(x, y, k, l, sigmaX, sigmaY, sigmaR, g)
    return sum


def weight(x, y, k, l, sigmaX, sigmaY, sigmaR, g):
    return pre_weight(x, y, k, l, sigmaX, sigmaY, sigmaR, g)/sum_pre_weight(x, y, sigmaX, sigmaY,
sigmaR, g)


sigmaX = 100
sigmaY = 100
sigmaR = 100
for i in range(1, height-1):
    for j in range(1, width-1):
        recovered_image[i - 1, j - 1] = ₩
            np.sum(np.multiply(padded_image[i-1:i+2, j-1:j+2],
                               np.array([[weight(i, j, k, l, sigmaX, sigmaY, sigmaR, padded_image)
                                          for k in range(i-1, i+2)] for l in range(j-1, j+2)])))

plt.subplot(121), plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(recovered_image, cmap='gray')
```

```
plt.title('Filtered Image(Bilateral)'), plt.xticks([]), plt.yticks([])
plt.show()
```