HW 8: Noise Removal

Source Code

All questions are written in Python code, please refer to the file “main.py”.
All images will be stored in the folder “res” (automatically create a new folder).
In accordance with the **FAQ** of course website:

- All parts of the question are written from scratch, except for plotting images

Answer

1. Gaussian Noise with Amplitude

```
def GaussianNoise(source, mean, stdev, amplitude):
    return source + (amplitude * np.random.normal(mean, stdev,
source.shape))
```

2. Salt-and-Pepper Noise with Probability

```
def SaltPepperNoise(source, threshold):
    result = source.copy()
    random_map = np.random.uniform(low=0, high=1,
size=source.shape)
    for i in range(source.shape[0]):
        for j in range(source.shape[1]):
            if (random_map[i][j] < threshold):
                result[i][j] = 0
            elif (random_map[i][j] > 1-threshold):
                result[i][j] = 255
    result = result.astype(int)
    return result
```

3. Box Filter

```
def BoxFilter(source, boxsize):
    result = source.copy()
    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            boxpixel = []
            for x in range(boxsize):
                for y in range(boxsize):
                    xdest = i + (x - (boxsize//2))
                    ydest = j + (y - (boxsize//2))
                    if ((0 <= xdest < source.shape[0]) and (0 <=
ydest < source.shape[1])):
```

```
                        boxpixel.append(source[xdest][ydest])
            result[i][j] = int(sum(boxpixel) / len(boxpixel))
    return result
```

4. Median Filter

```
def MedianFilter(source, boxsize):
    result = source.copy()
    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            boxpixel = []
            for x in range(boxsize):
                for y in range(boxsize):
                    xdest = i + (x - (boxsize//2))
                    ydest = j + (y - (boxsize//2))
                    if ((0 <= xdest < source.shape[0]) and (0 <=
ydest < source.shape[1])):
                        boxpixel.append(source[xdest][ydest])
            boxpixel.sort()
            medianpixel = boxpixel[len(boxpixel)//2]
            result[i][j] = int(medianpixel)
    return result
```

5. Opening-then-closing and Closing-then opening Filter

```
def grayDilation(img, kernel):
    temp = img.copy()
    ycenter = int(kernel.shape[0] / 2)
    xcenter = int(kernel.shape[1] / 2)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            pixel = 0
            for x in range(kernel.shape[0]):
                for y in range(kernel.shape[1]):
                    if kernel[x][y] == 1:
                        xdest = i + x - ycenter
                        ydest = j + y - xcenter
                        if (0 <= xdest < img.shape[0]) and (0 <=
ydest < img.shape[0]):
                            pixel = max(pixel, img[xdest][ydest])
            temp[i][j] = pixel
    return temp

def grayErosion(img, kernel):
    temp = img.copy()
    ycenter = int(kernel.shape[0] / 2)
    xcenter = int(kernel.shape[1] / 2)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            pixel = 255
```

```
            for x in range(kernel.shape[0]):
                for y in range(kernel.shape[1]):
                    if kernel[x][y] == 1:
                        xdest = i + x - ycenter
                        ydest = j + y - xcenter
                        if (0 <= xdest < img.shape[0]) and (0 <=
ydest < img.shape[0]):
                            pixel = min(pixel, img[xdest][ydest])
            temp[i][j] = pixel
    return temp

def opening(source, kernel):
    return grayDilation(grayErosion(source, kernel), kernel)

def closing(source, kernel):
    return grayErosion(grayDilation(source, kernel), kernel)

def OpentoCloseFunc(source, kernel):
    return closing(opening(source, kernel), kernel)

def ClosetoOpenFunc(source, kernel):
    return opening(closing(source, kernel), kernel)
```

6. SNR

```
def SNR(original, noise):
    import math
    height, width = original.shape
    # Normalize
    ori_normalized = original / 255
    noise_normalized = noise / 255
    # VS
    us = np.sum(ori_normalized) / (height * width)
    temp1 = 0
    temp2 = 0
    for i in range(height):
        for j in range(width):
            temp1 += (ori_normalized[i][j] - us)**2
            temp2 += (noise_normalized[i][j] -
ori_normalized[i][j])
    VS = temp1 / (height * width)
    # VN
    unoise = temp2 / (height * width)
    temp = 0
    for i in range(height):
        for j in range(width):
            temp += (noise_normalized[i][j] -
ori_normalized[i][j] - unoise)**2
    VN = temp / (height * width)
    result = 20 * math.log10(math.sqrt(VS) / math.sqrt(VN))
```

```
    return result
```

Gaussian (10 & 30 respectively)

SNR of gaussian_10: 13.580451435301375
SNR of gaussian_30: 4.0431840504478735

Salt-and-Pepper (0.05 & 0.10 respectively)

SNR of saltpepper_05: 0.9183728229017984
SNR of saltpepper_10: -2.0792294087949004

Box Filter 3x3 (Gaussian 10 & Gaussian 30 respectively)

SNR of box_3_gaussian10: 17.727336859947375
SNR of box_3_gaussian30: 12.538914176760418

Box Filter 3x3 (Salt-and-Pepper 0.05 & Salt-and-Pepper 0.10 respectively)

SNR of box_3_saltpepper05: 9.42579989381387
SNR of box_3_saltpepper10: 6.382717374015606

Box Filter 5x5 (Gaussian 10 & Gaussian 30 respectively)

SNR of box_5_gaussian10: 14.86174805694938
SNR of box_5_gaussian30: 13.315149839504967

Box Filter 5x5 (Salt-and-Pepper 0.05 & Salt-and-Pepper 0.10 respectively)

SNR of box_5_saltpepper05: 11.12830260215814
SNR of box_5_saltpepper10: 8.540053353128048

Median Filter 3x3 (Gaussian 10 & Gaussian 30 respectively)


SNR of median_3_gaussian10: 17.62998629899334
SNR of median_3_gaussian30: 11.074165496630622

Median Filter 3x3 (Salt-and-Pepper 0.05 & Salt-and-Pepper 0.10 respectively)


SNR of median_3_saltpepper05: 19.11894788420701
SNR of median_3_saltpepper10: 15.208334764430155

Median Filter 5x5 (Gaussian 10 & Gaussian 30 respectively)

SNR of median_5_gaussian10: 16.016571024104763
SNR of median_5_gaussian30: 12.898755414474405

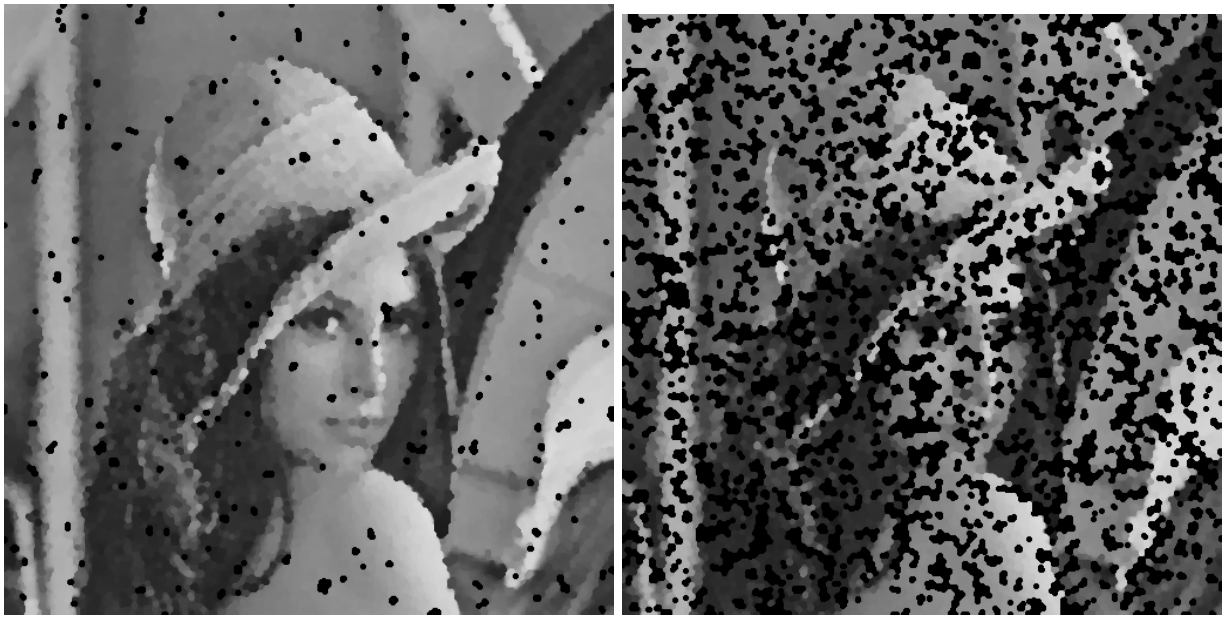
Median Filter 5x5 (Salt-and-Pepper 0.05 & Salt-and-Pepper 0.10 respectively)

SNR of median_5_saltpepper05: 16.324705535510283
SNR of median_5_saltpepper10: 15.712610094036418

Opening-then-closing (Gaussian 10 & Gaussian 30 respectively)

SNR of opentoclose_gaussian10: 13.24380950626756
SNR of opentoclose_gaussian30: 11.161011106496144

Opening-then-closing (Salt-and-Pepper 0.05 & Salt-and-Pepper 0.10 respectively)



SNR of opentoclose_saltpepper05: 5.221519513579595
SNR of opentoclose_saltpepper10: -2.129574221364153

Closing-then-opening (Gaussian 10 & Gaussian 30 respectively)

SNR of closetoopen_gaussian10: 13.58871637708621
SNR of closetoopen_gaussian30: 11.174166816499644

Closing-then-opening (Salt-and-Pepper 0.05 & Salt-and-Pepper 0.10 respectively)

SNR of closetoopen_saltpepper05: 5.7890540624734275
SNR of closetoopen_saltpepper10: -2.6272087689072072