

# HW 8

| b05902019 資工三 蔡青邑



Using "lena.bmp" as input image.

## Python Packages I used

- `skimage.io`: for basic image i/o.
- `numpy`: for convenience of array manipulation.
- `tqdm`: for showing the progress of the executing of the code.
- `math`: for `log10` and square calculation.

## Some Other Functions I Build

- `blank_image(height, width)`: returning an all-black image of the given input height and width.
- `kernel_value(x, y)`: return the kernel value of position (x, y), which, in

our case, are always 0.

## Noise

I use the `random.normal` function from NumPy to help me get the distribution from normal distribution, and `random.uniform` to get me the uniform distribution. Below is actually the code I learned from our slide with a little bit modification.

- `img_Gaul10` : gaussian noise with amplitude of 10
- `img_Gaul30` : gaussian noise with amplitude of 30
- `img_Salt005` : salt-and-pepper noise with probability 0.05
- `img_Salt01` : salt-and-pepper noise with probability 0.1

```
# start noising
import numpy as np
grayImg = lena.copy()
img_Gaul10 = np.zeros(grayImg.shape, dtype=int)
img_Gaul30 = np.zeros(grayImg.shape, dtype=int)
img_Salt005 = np.zeros(grayImg.shape, dtype=int)
img_Salt01 = np.zeros(grayImg.shape, dtype=int)

for i in range(height):
    for j in range(width):
        img_Gaul10[i, j] = grayImg[i, j] + 10 * np.random.normal(0.0,
1.0, None)
        img_Gaul30[i, j] = grayImg[i, j] + 30 * np.random.normal(0.0,
1.0, None)
        if(np.random.uniform(0., 1., None) < 0.05):
            img_Salt005[i, j] = 0
        elif(np.random.uniform(0., 1., None) > 1 - 0.05):
            img_Salt005[i, j] = 255
        else:
            img_Salt005[i, j] = grayImg[i, j]
        if(np.random.uniform(0., 1., None) < 0.1):
            img_Salt01[i, j] = 0
        elif(np.random.uniform(0., 1., None) > 1 - 0.1):
            img_Salt01[i, j] = 255
        else:
            img_Salt01[i, j] = grayImg[i, j]
img_Gaul10 = np.clip(img_Gaul10, 0, 255)
img_Gaul30 = np.clip(img_Gaul30, 0, 255)
```

## Box Filter

I use for loop to calculate the sum and leave the points near the edge of images unchanged.

```
def box_filter(img, filterxx):
    temp = img.copy()
    for i in tqdm(range(height)):
        for j in range(width):
            filter_applied = [i, j] + filterxx
            # print(filter_applied)
            SUM = 0
            count = 0
            out_of_range = False
            for x, y in filter_applied:
                if((x in range(0, height)) and (y in range(0,
width))):
                    SUM += temp[x][y]
                    count += 1
                else:
                    out_of_range = True
                    break
            if not out_of_range:
                temp[i][j] = SUM / count
    return temp
```

## Median Filter

Simarlarily, I make an empty list and append all value in the filter into it, and then I calculate the median by my self-written funciton. (For the pixels near the edge of the image, I leave them unchanged)

```
def median(x):
    x = sorted(x)
    listlength = len(x)
    num = round(listlength / 2)
    middlenum = x[num]
    return middlenum

# median filter
def median_filter(img, filterxx):
    temp = img.copy()
    for i in tqdm(range(height)):
        for j in range(width):
            filter_applied = [i, j] + filterxx
            # print(filter_applied)
            SUM = []
            out_of_range = False
            for x, y in filter_applied:
```

```

        if((x in range(0, height)) and (y in range(0,
width))):
            SUM.append(temp[x][y])
        else:
            out_of_range = True
            break
    if not out_of_range:
        temp[i][j] = int(median(SUM))
return temp

```

## Dilation, Erosion, Opening, and Closing, open\_then\_closing, closing\_then\_opening

I wrote a function for each of them, below is how I implement.

### Dilation

- my function: `dilation(img, kernel)`
- $(f \oplus k)(x, y) = \max\{f(x - i, y - j) + k(i, j) | (i, j) \in K, (x - i, y - j) \in f\}$

### Erosion

- my function: `erosion(img, kernel)`
- $(f \ominus k)(x, y) = \min\{f(x + i, y + j) - k(i, j) | (i, j) \in K, (x + i, y + j) \in f\}$

### Opening

- my function: `opening(img, kernel)`
- $B \circ K = (B \ominus K) \oplus K$

Simply apply the formula:

```

def opening(img, kernel):
    temp = erosion(img, kernel)
    temp = dilation(temp, kernel)
    return temp

```

### Closing

- my function: `closing(img, kernel)`
- $B \bullet K = (B \oplus K) \ominus K$

Simply apply the formula:

```

def closing(img, kernel):
    temp = dilation(img, kernel)
    temp = erosion(temp, kernel)
    return temp

```

## open\_then\_closing

```

def opening_then_closing(img, kernel):
    temp = opening(img, kernel)
    temp = closing(img, kernel)
    return temp

```

## closing\_then\_opening

```

def closing_then_opening(img, kernel):
    temp = closing(img, kernel)
    temp = opening(img, kernel)
    return temp

```

# How I calculate SNR

name: (SNR,  $\mu_N$ , VN)

```

gaussian10: (13.629265164173457, -0.4727897644042969, 99.29737056550915)
gaussian30: (4.062263030794185, -0.6105003356933594, 898.7480512052648)
salt_and_pepper_noise005: (1.0559193956714161, -0.0734710693359375, 1795.8593709774832)
salt_and_pepper_noise01: (-1.849405246608125, -0.4713935852050781, 3505.9425029125414)
box_filter3x3_gaussian10: (16.402219149906305, -0.7838706970214844, 52.43754486866905)
box_filter3x3_gaussian30: (12.0949638939512, -0.9214439392089844, 141.3734561734928)
box_filter3x3_salt_and_pepper005: (9.379928232058706, -0.3886604309082031, 264.1626675871424)
box_filter3x3_salt_and_pepper01: (6.517863296406162, -0.7821693420410156, 510.5966194773899)
box_filter5x5_gaussian10: (13.604809293028804, -1.0313720703125, 99.85810850560665)
box_filter5x5_gaussian30: (12.386430656584853, -1.168304443359375, 132.19687808142316)
box_filter5x5_salt_and_pepper005: (10.672159149037803, -0.6364898681640625, 196.17711789115222)
box_filter5x5_salt_and_pepper01: (8.431149678541395, -1.0309257507324219, 328.6618510618309)
median_filter3x3_gaussian10: (17.58516964925235, -0.5956306457519531, 39.93441251906894)
median_filter3x3_gaussian30: (10.999809302070354, -0.8469009399414062, 181.92120942126394)
median_filter3x3_salt_and_pepper005: (18.589494071450034, -0.1982574462890625, 31.68946137983752)
median_filter3x3_salt_and_pepper01: (14.94310047901201, -0.3736686706542969, 73.37603105732104)
median_filter5x5_salt_and_pepper005: (15.946698623648171, -0.36748504638671875, 58.23638418459109)
median_filter5x5_salt_and_pepper01: (14.190944806868103, -0.6039199829101562, 87.25087361109426)
median_filter5x5_gaussian10: (15.833171412747788, -0.7803993225097656, 59.77879046613305)
median_filter5x5_gaussian30: (12.588413695115634, -1.0368728637695312, 126.18940577271876)
open_close_gaussian10 (13.216340004283573, -12.325607299804688, 109.20191750936083)
open_close_gaussian30 (11.177740203715285, -35.022682189941406, 174.61849644452957)
open_close_sap005 (5.426266235011431, -11.865680694580078, 656.5044001851943)
open_close_sap01 (-2.104435489761353, -47.96576690673828, 3717.986544784652)
close_open_gaussian10 (13.637574521507092, 10.76889419555664, 99.10756651724834)
close_open_gaussian30 (11.132352772817162, 33.335113525390625, 176.45297602397477)
close_open_sap005 (6.040581197025387, 10.61874771118164, 569.9098262871522)
close_open_sap01 (-1.8099504536696402, 41.27030563354492, 3474.235952064078)

```

```

from skimage import io
import numpy as np
import os
import math

```

```
lena = io.imread('lena.bmp')
namelist = os.listdir(".")
mu = np.sum(lena) / lena.size
VS = np.sum(np.power(lena - mu, 2)) / lena.size
print('mu =', mu, ', VS =', VS)
for i in namelist:
    if(not '.png' in i):
        continue
    temp = io.imread(i)
    mu_N = np.sum(temp - lena) / temp.size
    VN = np.sum(np.power(temp - lena - mu_N, 2)) / temp.size
    # print(temp - lena - mu_N)
    SNR = 20 * math.log10(math.sqrt(VS) / math.sqrt(VN))
```

# Images

lena\_Salto01.png



lena\_Gaul30.png



lena\_Salto05.png



closing\_then\_opening\_on\_Salto05.png



box\_filter3X3\_on\_Gaul10.png



median\_filter3X3\_on\_Salto05.png



median\_filter3X3\_on\_Gaul10.png



box\_filter5X5\_on\_Salto05.png



opening\_then\_closing\_on\_Gaul10.png



box\_filter5X5\_on\_Salto1.png



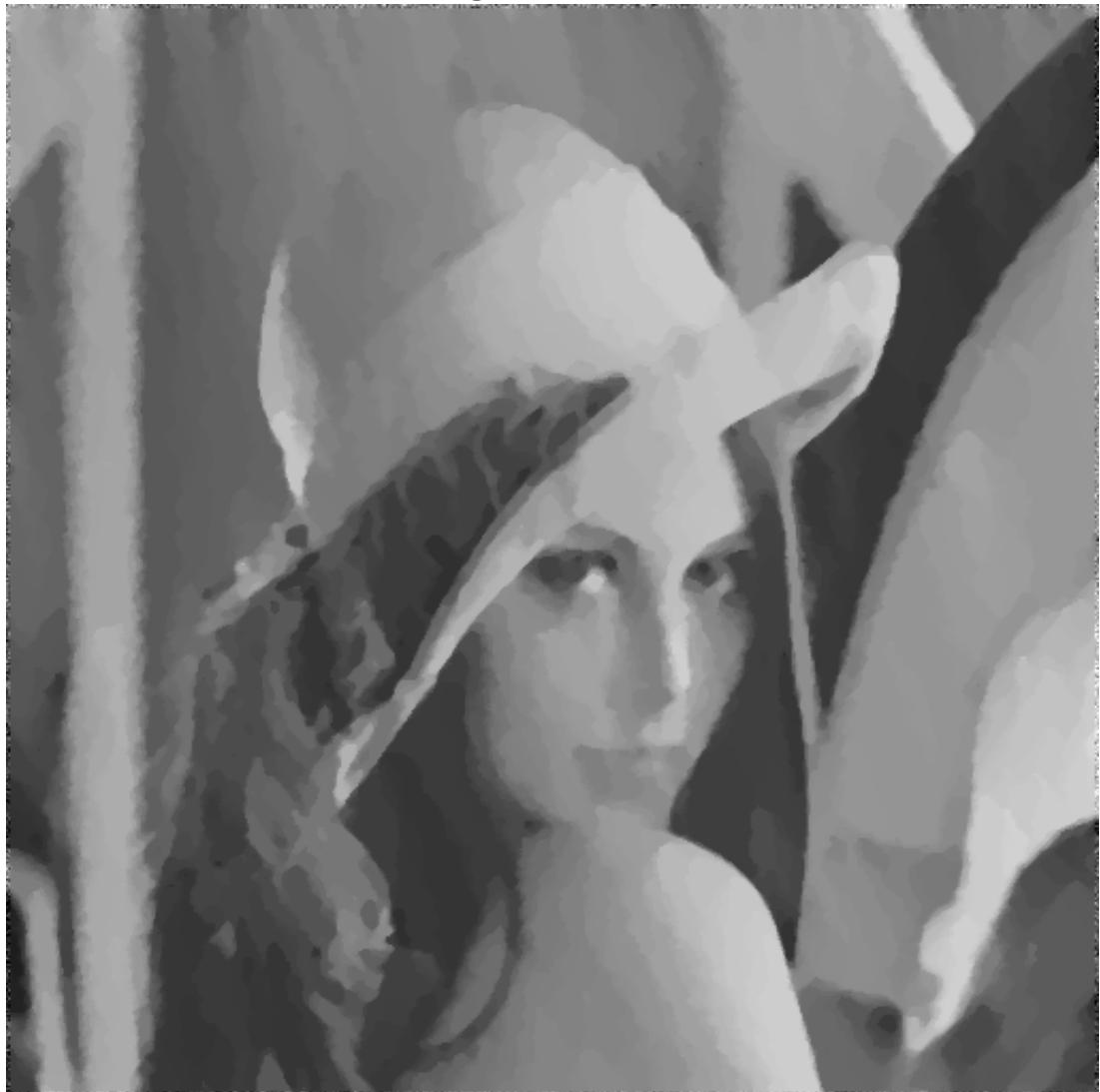
box\_filter5X5\_on\_Gaul30.png



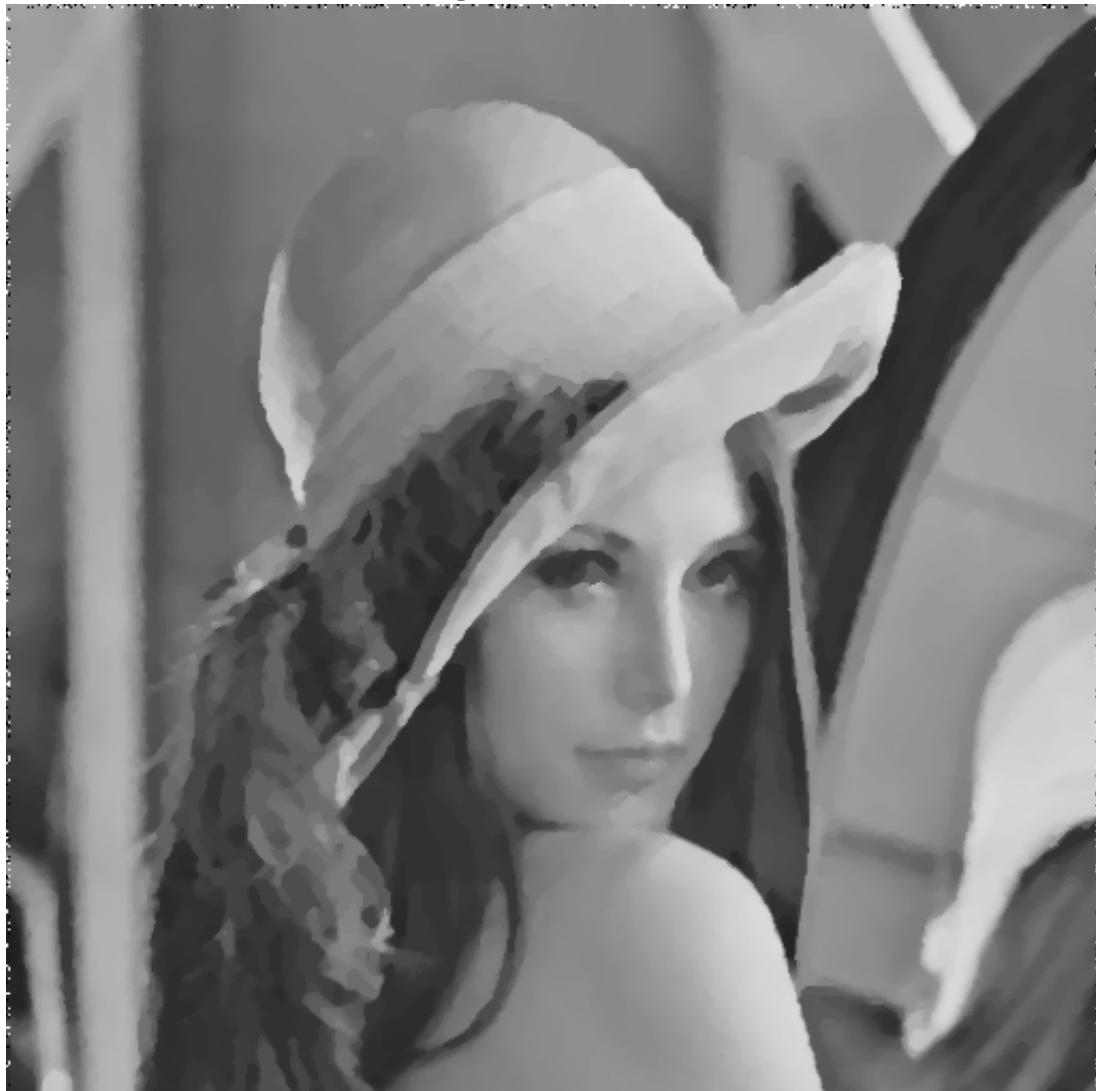
closing\_then\_opening\_on\_Gaul30.png



median\_filter5X5\_on\_Gaul30.png



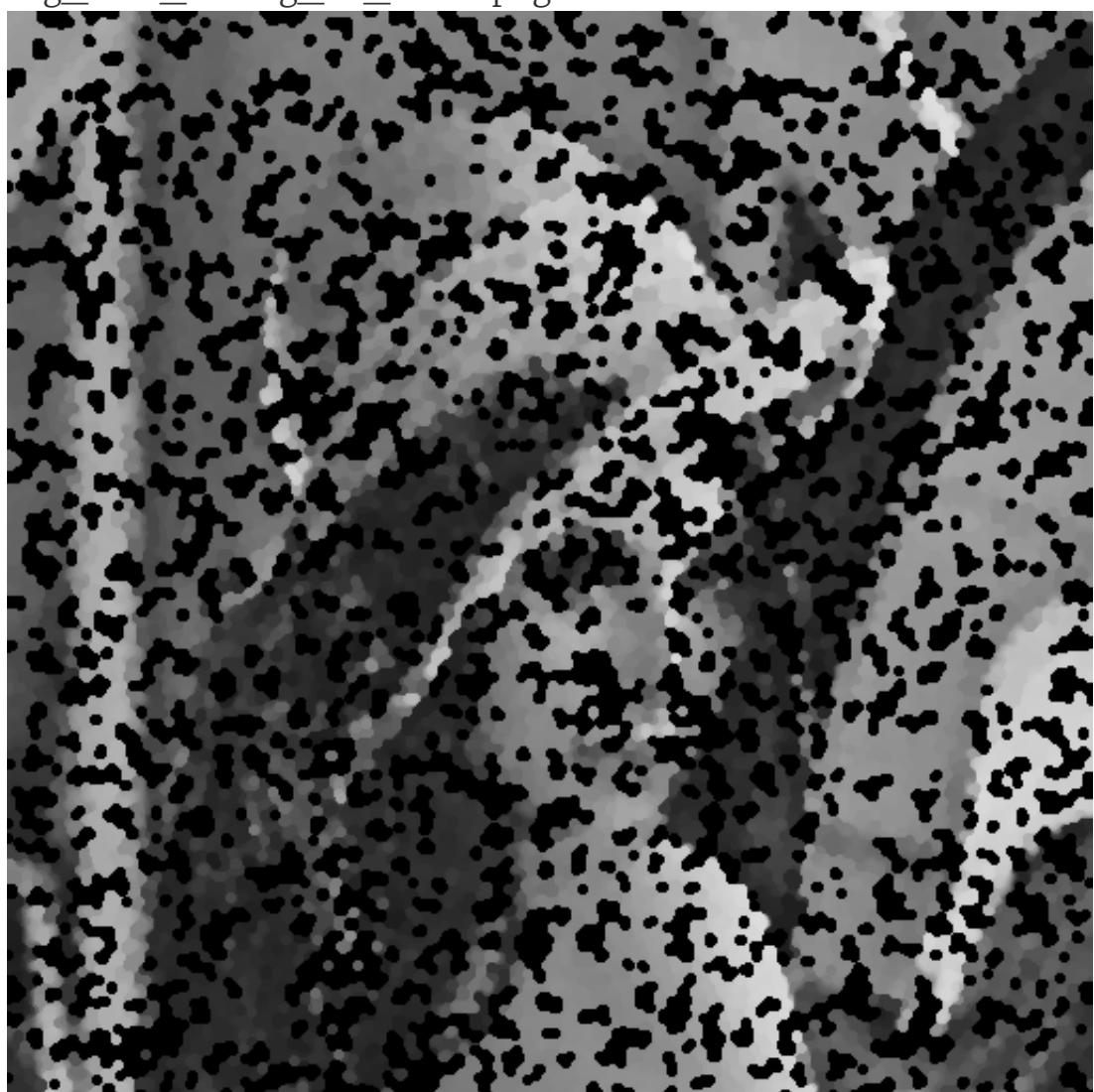
median\_filter5X5\_on\_Salto1.png



closing\_then\_opening\_on\_Salto1.png



opening\_then\_closing\_on\_Salto1.png



opening\_then\_closing\_on\_Gaul30.png



opening\_then\_closing\_on\_Salto05.png



box\_filter5X5\_on\_Gaul10.png



box\_filter3X3\_on\_Salto05.png



median\_filter5X5\_on\_Gaul10.png



median\_filter5X5\_on\_Salto05.png



closing\_then\_opening\_on\_Gaul10.png



lena\_Gaul10.png



box\_filter3X3\_on\_Gaul30.png



box\_filter3X3\_on\_Salto1.png



median\_filter3X3\_on\_Salto1.png



median\_filter3X3\_on\_Gaul30.png

