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 convience 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.normal` 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, μ_N, VN)


```
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

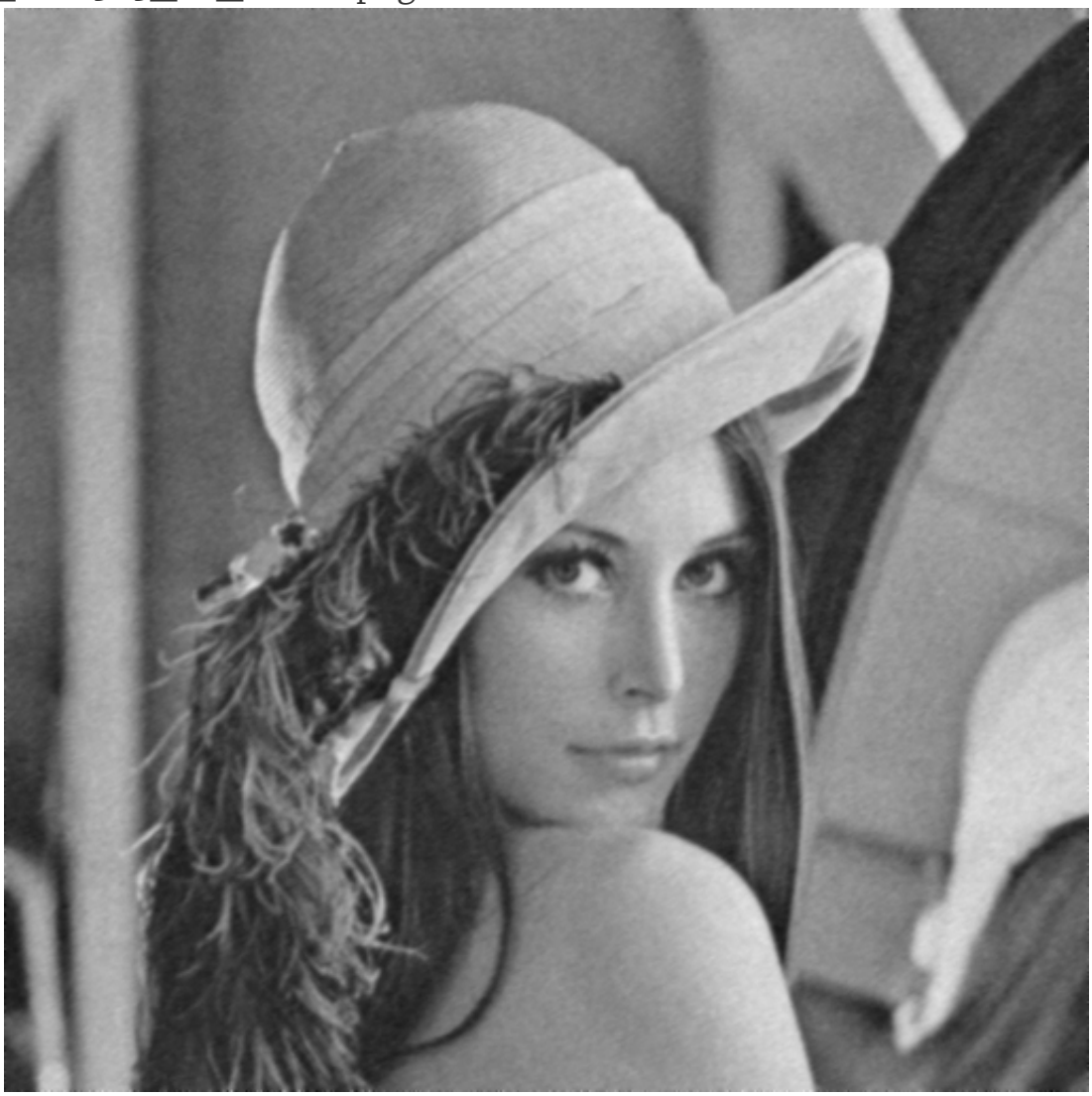
lena_Salt01.png

lena_Gaul30.png

lena_Salt005.png

closing_then_opening_on_Salt005.png

box_filter3X3_on_Gaul10.png



median_filter3X3_on_Salt005.png

median_filter3X3_on_Gaul10.png

box_filter5X5_on_Salt005.png

opening_then_closing_on_Gaul10.png

box_filter5X5_on_Salt01.png

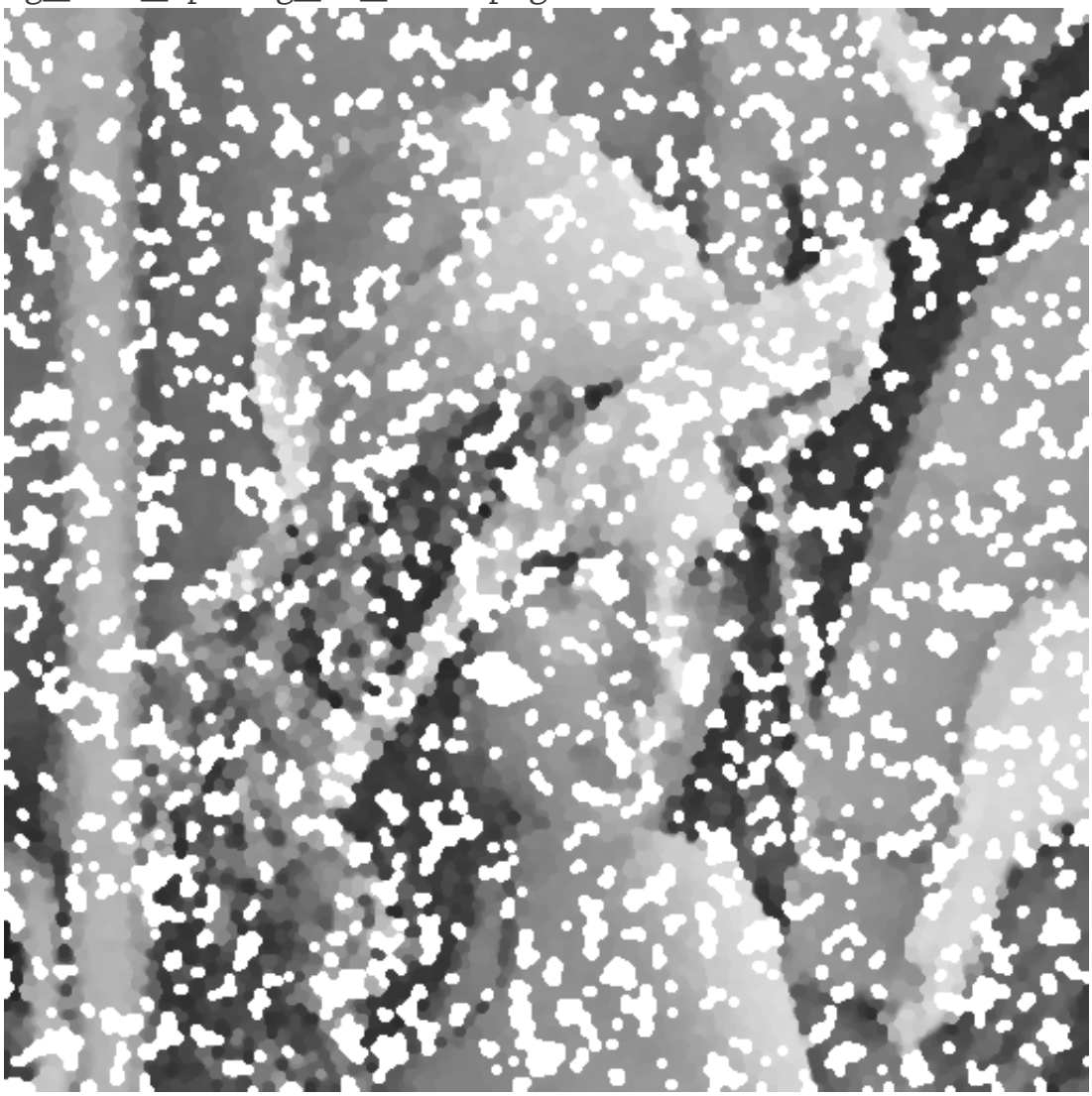
box_filter5X5_on_Gaul30.png

closing_then_opening_on_Gaul30.png

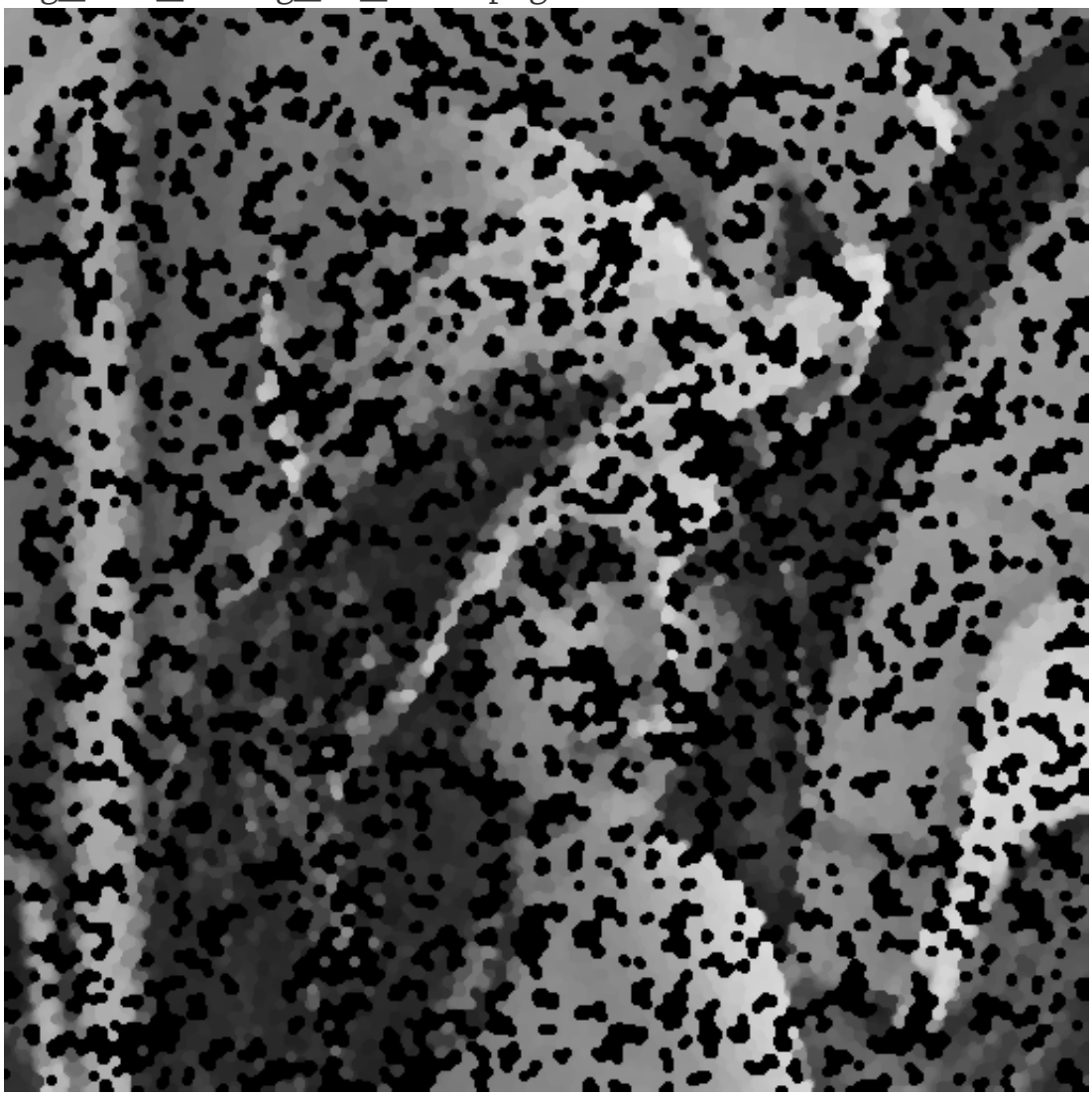
median_filter5X5_on_Gaul30.png

median_filter5X5_on_Salt01.png

closing_then_opening_on_Salt01.png



opening_then_closing_on_Salt01.png



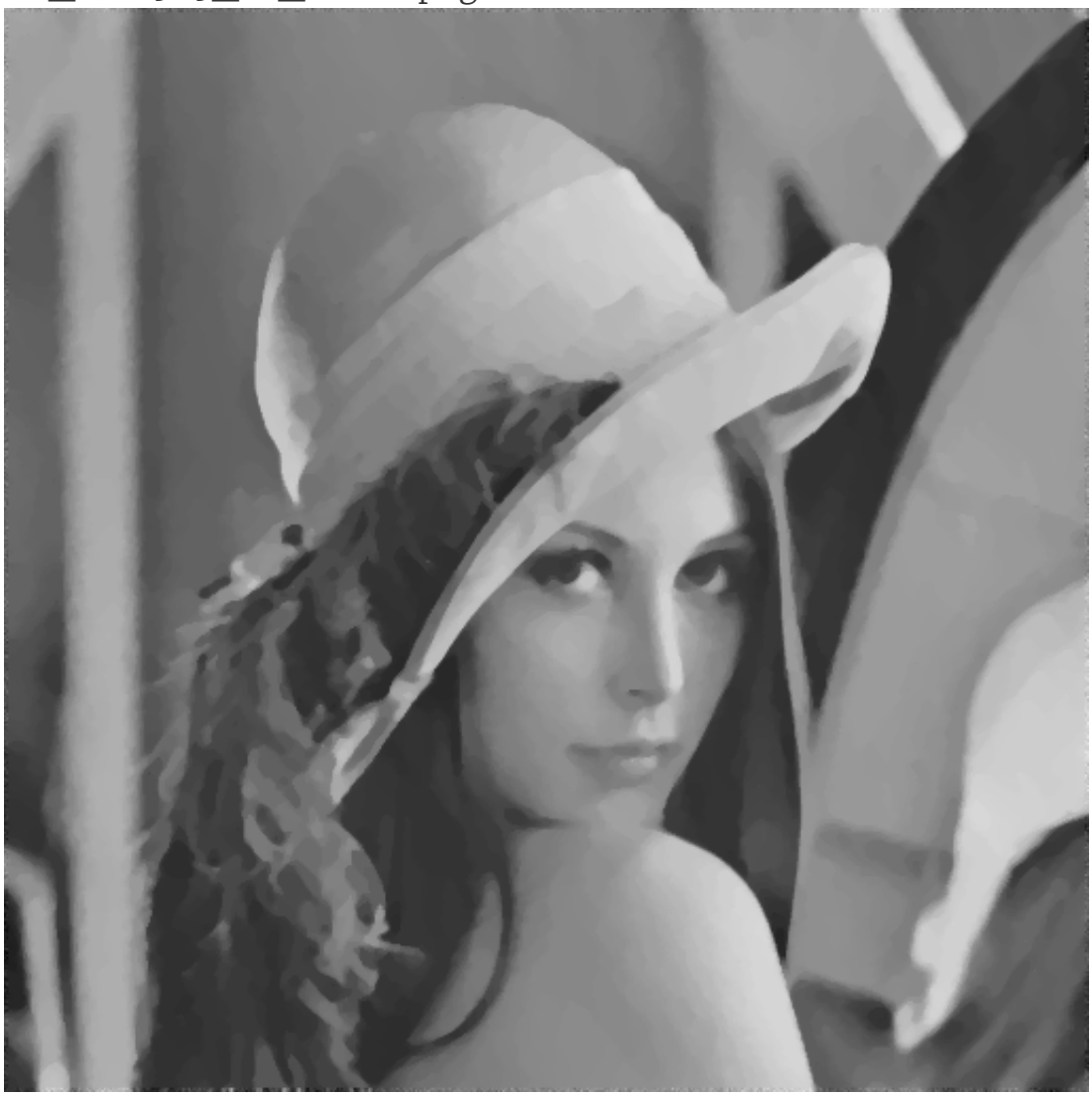
opening_then_closing_on_Gaul30.png

opening_then_closing_on_Salt005.png

box_filter5X5_on_Gaul10.png

box_filter3X3_on_Salt005.png

median_filter5X5_on_Gaul10.png



median_filter5X5_on_Salt005.png

closing_then_opening_on_Gaul10.png

lena_Gaul10.png

box_filter3X3_on_Gaul30.png

box_filter3X3_on_Salt01.png

median_filter3X3_on_Salt01.png

median_filter3X3_on_Gaul30.png