Computer Vision HW8 Report
蕭恩慈 / B07902095

The task is to write a program which generates the following:

a. Gaussian noisy image with amplitude of 10
b. Gaussian noisy image with amplitude of 30
c. Salt-and-pepper noisy image with probability of 0.1
d. Salt-and-pepper noisy image with probability of 0.05

Then, continuing the task, the program also needs to apply the following filters to the above generated images:

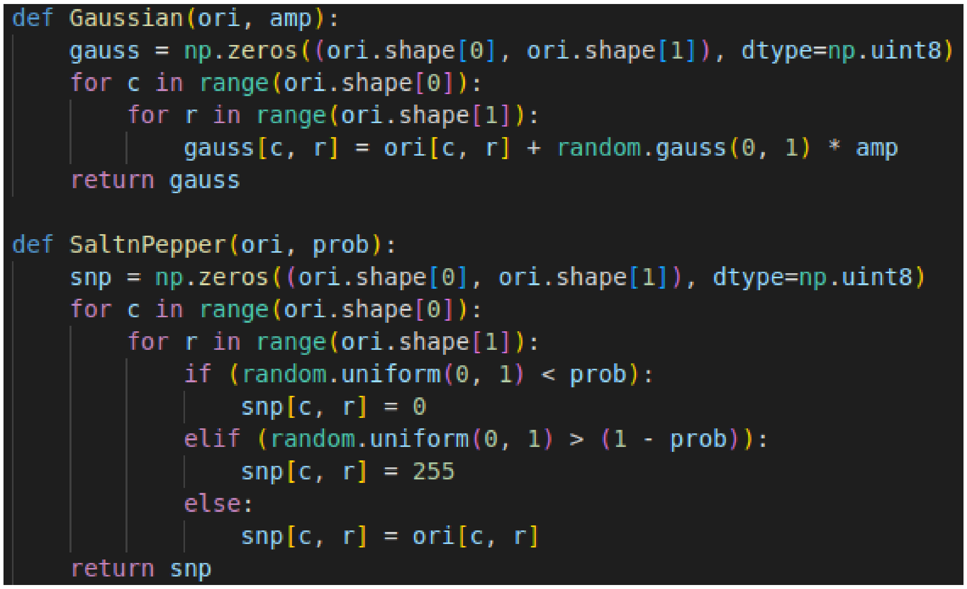
e. 3x3 Box filter
f. 5x5 Box filter
g. 3x3 Median filter
h. 5x5 Median filter
i. Opening-then-closing filter (Using octagonal kernel and value of 0)
j. Closing-then-opening filter (Using octagonal kernel and value of 0)

Lastly, the program has to calculate the signal-to-ratio(SNR) to all the above generated images and filtered images.

To complete the task, the program language that is used is Python, with numpy library for execution, math and random library for the calculation, and cv2 for opening and saving the image. For easier inspection, I assigned the filtered image results to the directory based on the generated image, while putting the generated image and the SNR text file in the main directory.

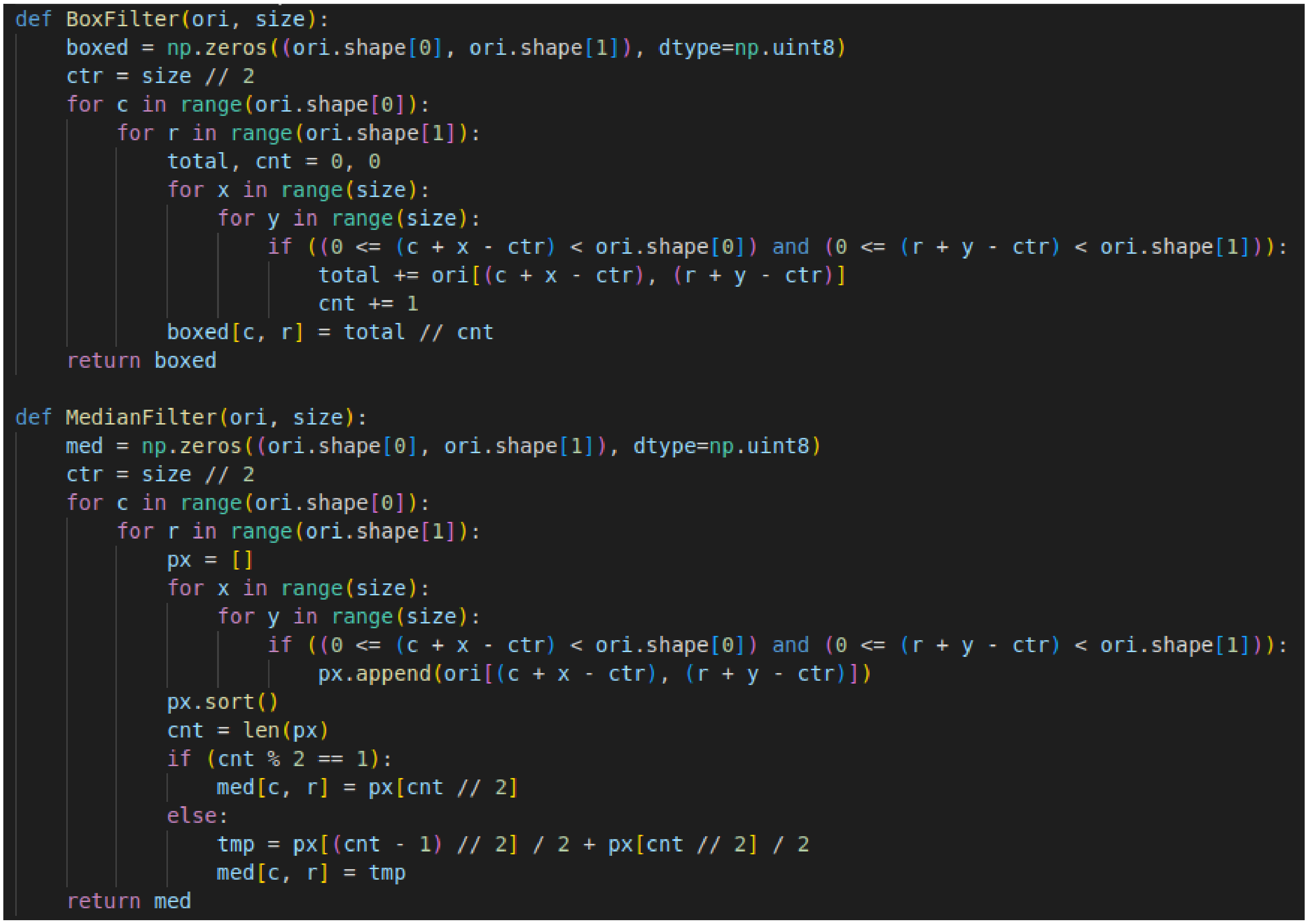
B07902095_HW8_ver1
|__________________ gauss-10 (for the results of filtered Gaussian noise image with amplitude of 10)
|__________________ gauss-30 (for the results of filtered Gaussian noise image with amplitude of 30 results)
|__________________ snp-01 (for the results of filtered salt-and-pepper noise image with probability of 0.1)
|__________________ snp-0.05 (for the results of filtered salt-and-pepper noise image with probability of 0.05)
|__________________ main.py
|__________________ SNR.txt (SNR of each images result are written here)
|__________________ lena-gauss-10.bmp (image result for Gaussian noise image with amplitude of 10)
|__________________ lena-gauss-30.bmp (image result for Gaussian noise image with amplitude of 30)
|__________________ lena-snp-01.bmp (image result for salt-and-pepper noise image with probability of 0.1)
|__________________ lena-snp-005.bmp (image result for salt-and-pepper noise image with probability of 0.05)
|__________________ lena.bmp (the original image)
|__________________ report.pdf

Code snippet for generating Gaussian noise image and salt-and-pepper noise image:



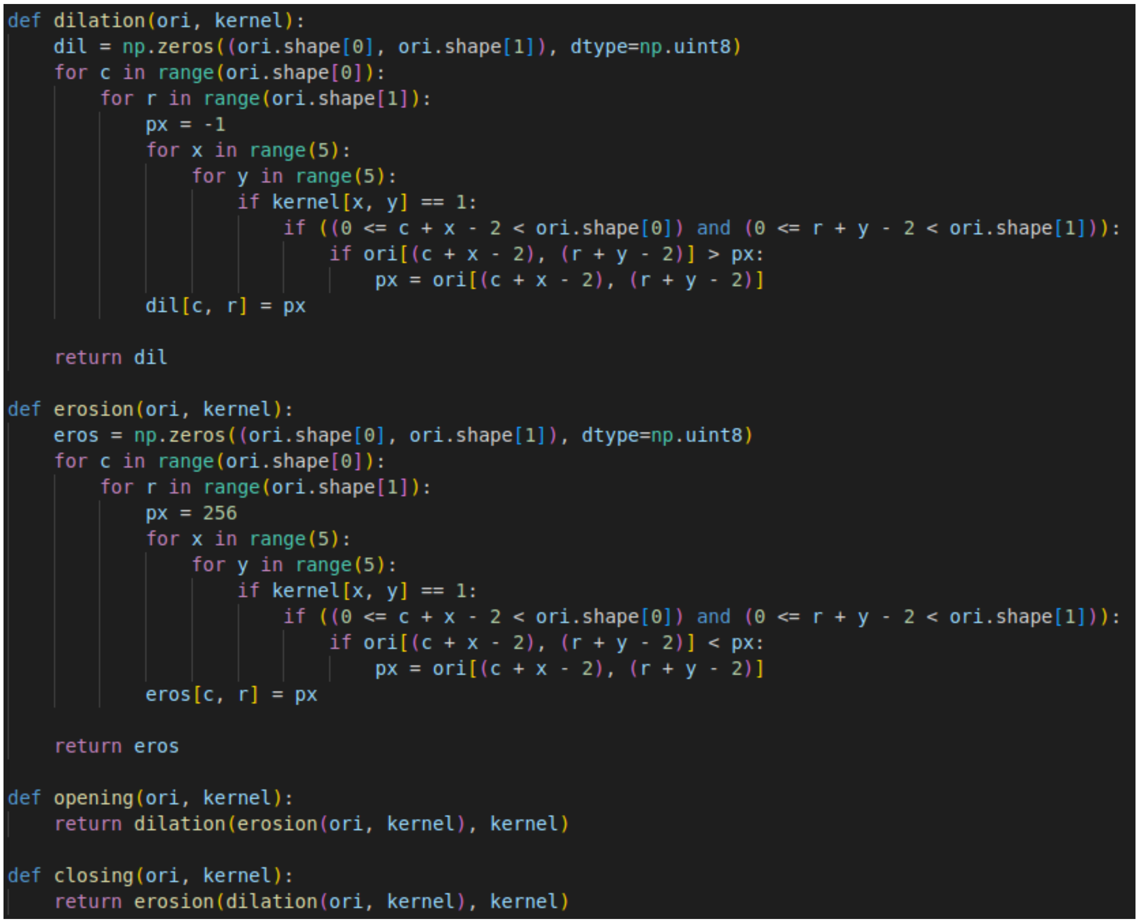
```
def Gaussian(ori, amp):
    gauss = np.zeros((ori.shape[0], ori.shape[1]), dtype=np.uint8)
    for c in range(ori.shape[0]):
        for r in range(ori.shape[1]):
            gauss[c, r] = ori[c, r] + random.gauss(0, 1) * amp
    return gauss

def SaltnPepper(ori, prob):
    snp = np.zeros((ori.shape[0], ori.shape[1]), dtype=np.uint8)
    for c in range(ori.shape[0]):
        for r in range(ori.shape[1]):
            if (random.uniform(0, 1) < prob):
                snp[c, r] = 0
            elif (random.uniform(0, 1) > (1 - prob)):
                snp[c, r] = 255
            else:
                snp[c, r] = ori[c, r]
    return snp
```

Code snippet for applying box filter and median filter:



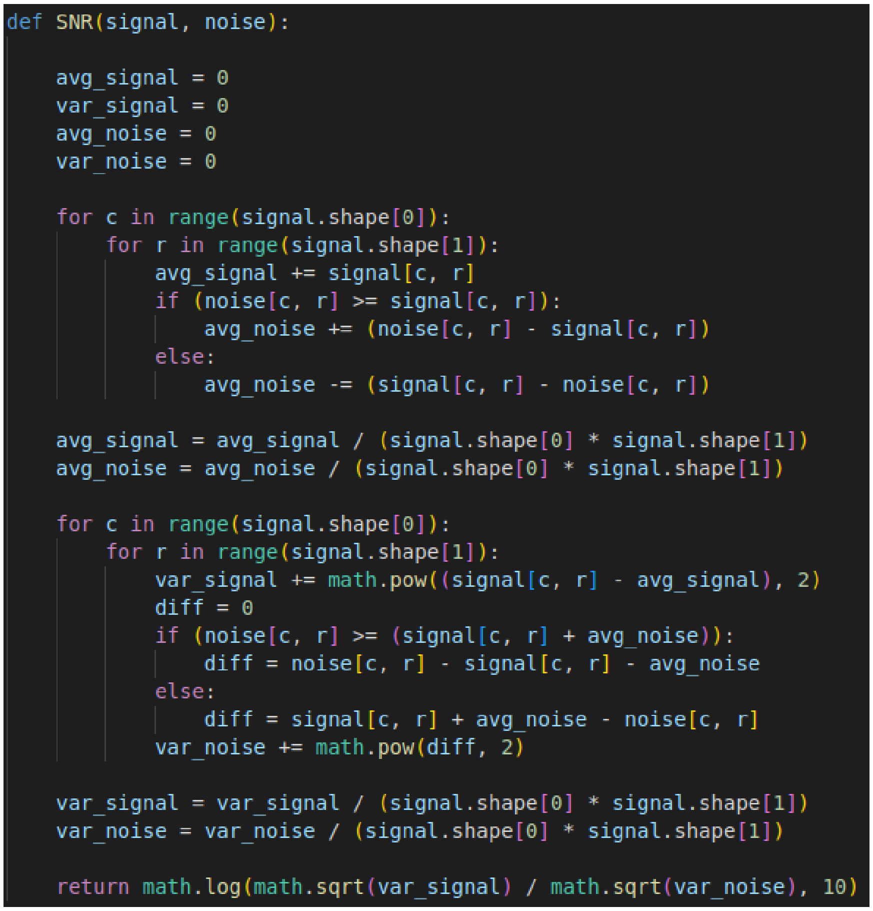
```
def BoxFilter(ori, size):
    boxed = np.zeros((ori.shape[0], ori.shape[1]), dtype=np.uint8)
    ctr = size // 2
    for c in range(ori.shape[0]):
        for r in range(ori.shape[1]):
            total, cnt = 0, 0
            for x in range(size):
                for y in range(size):
                    if ((0 <= (c + x - ctr) < ori.shape[0]) and (0 <= (r + y - ctr) < ori.shape[1])):
                        total += ori[(c + x - ctr), (r + y - ctr)]
                        cnt += 1
            boxed[c, r] = total // cnt
    return boxed

def MedianFilter(ori, size):
    med = np.zeros((ori.shape[0], ori.shape[1]), dtype=np.uint8)
    ctr = size // 2
    for c in range(ori.shape[0]):
        for r in range(ori.shape[1]):
            px = []
            for x in range(size):
                for y in range(size):
                    if ((0 <= (c + x - ctr) < ori.shape[0]) and (0 <= (r + y - ctr) < ori.shape[1])):
                        px.append(ori[(c + x - ctr), (r + y - ctr)])
            px.sort()
            cnt = len(px)
            if (cnt % 2 == 1):
                med[c, r] = px[cnt // 2]
            else:
                tmp = px[(cnt - 1) // 2] / 2 + px[cnt // 2] / 2
                med[c, r] = tmp
    return med
```

Code snippet for opening-then-closing and closing-then-opening:



```
def dilation(ori, kernel):
    dil = np.zeros((ori.shape[0], ori.shape[1]), dtype=np.uint8)
    for c in range(ori.shape[0]):
        for r in range(ori.shape[1]):
            px = -1
            for x in range(5):
                for y in range(5):
                    if kernel[x, y] == 1:
                        if ((0 <= c + x - 2 < ori.shape[0]) and (0 <= r + y - 2 < ori.shape[1])):
                            if ori[(c + x - 2), (r + y - 2)] > px:
                                px = ori[(c + x - 2), (r + y - 2)]
            dil[c, r] = px

    return dil

def erosion(ori, kernel):
    eros = np.zeros((ori.shape[0], ori.shape[1]), dtype=np.uint8)
    for c in range(ori.shape[0]):
        for r in range(ori.shape[1]):
            px = 256
            for x in range(5):
                for y in range(5):
                    if kernel[x, y] == 1:
                        if ((0 <= c + x - 2 < ori.shape[0]) and (0 <= r + y - 2 < ori.shape[1])):
                            if ori[(c + x - 2), (r + y - 2)] < px:
                                px = ori[(c + x - 2), (r + y - 2)]
            eros[c, r] = px

    return eros

def opening(ori, kernel):
    return dilation(erosion(ori, kernel), kernel)

def closing(ori, kernel):
    return erosion(dilation(ori, kernel), kernel)
```

Code snippet for generating SNR:



```python
def SNR(signal, noise):

    avg_signal = 0
    var_signal = 0
    avg_noise = 0
    var_noise = 0

    for c in range(signal.shape[0]):
        for r in range(signal.shape[1]):
            avg_signal += signal[c, r]
            if (noise[c, r] >= signal[c, r]):
                avg_noise += (noise[c, r] - signal[c, r])
            else:
                avg_noise -= (signal[c, r] - noise[c, r])

    avg_signal = avg_signal / (signal.shape[0] * signal.shape[1])
    avg_noise = avg_noise / (signal.shape[0] * signal.shape[1])

    for c in range(signal.shape[0]):
        for r in range(signal.shape[1]):
            var_signal += math.pow((signal[c, r] - avg_signal), 2)
            diff = 0
            if (noise[c, r] >= (signal[c, r] + avg_noise)):
                diff = noise[c, r] - signal[c, r] - avg_noise
            else:
                diff = signal[c, r] + avg_noise - noise[c, r]
            var_noise += math.pow(diff, 2)

    var_signal = var_signal / (signal.shape[0] * signal.shape[1])
    var_noise = var_noise / (signal.shape[0] * signal.shape[1])

    return math.log(math.sqrt(var_signal) / math.sqrt(var_noise), 10)
```