# Computer Vision hw8

B05902050

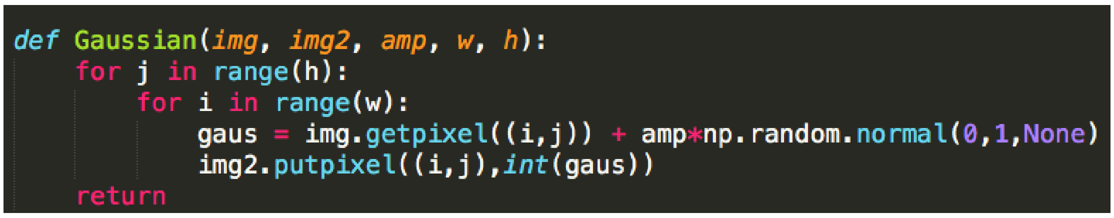
黃子源

I use PIL to complete the homework. In my program, I use function getpixel() and putpixel() to get the value of every pixel.

Gaussian:

the new image is the original pixel plus amplitude*gaussian random variable, I use numpy.random.normal to calculate N(0,1) to done this work.

Principal code fragment:

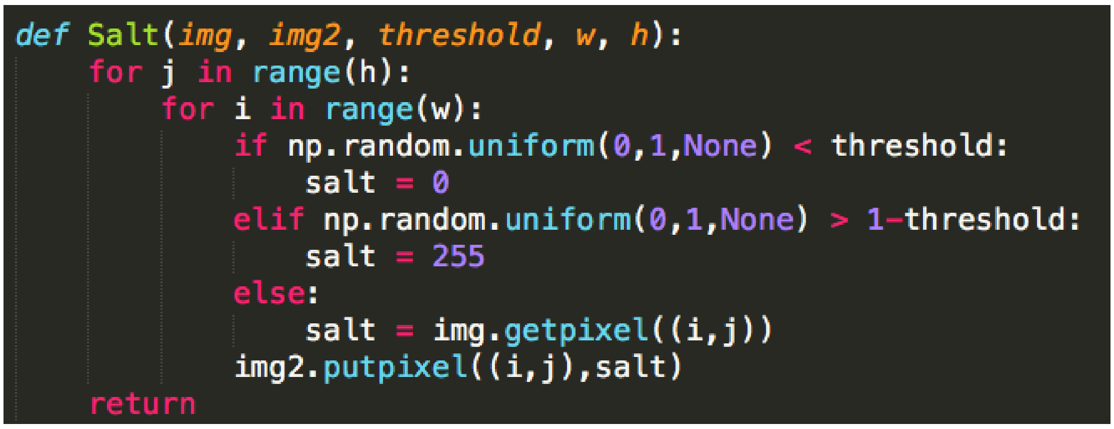


```python
def Gaussian(img, img2, amp, w, h):
    for j in range(h):
        for i in range(w):
            gaus = img.getpixel((i,j)) + amp*np.random.normal(0,1,None)
            img2.putpixel((i,j),int(gaus))
    return
```

Salt and Pepper:

The new image is decide from whether uniform(0,1) is smaller than threshold, I use numpy.random.uniform to calculate uniform(0,1) to done this work.

Principal code fragment:

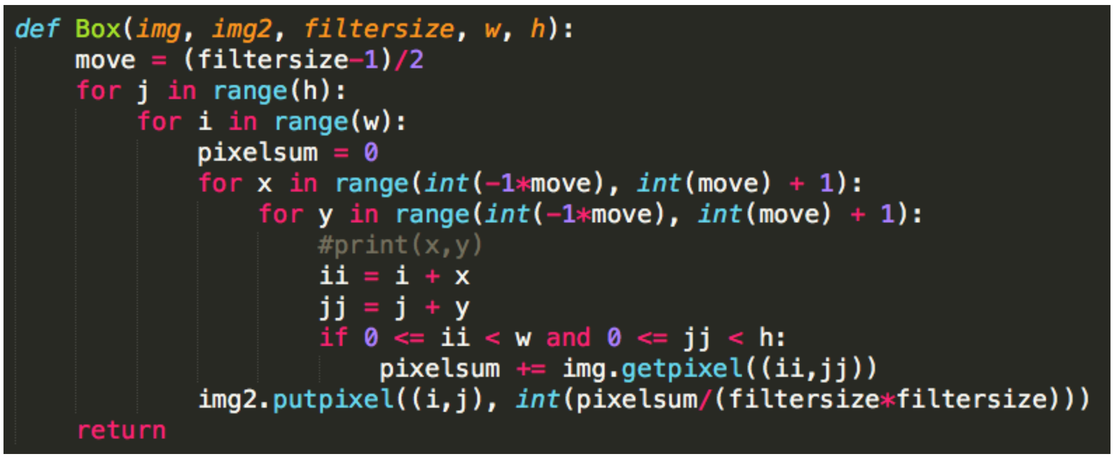


```python
def Salt(img, img2, threshold, w, h):
    for j in range(h):
        for i in range(w):
            if np.random.uniform(0,1,None) < threshold:
                salt = 0
            elif np.random.uniform(0,1,None) > 1-threshold:
                salt = 255
            else:
                salt = img.getpixel((i,j))
            img2.putpixel((i,j),salt)
    return
```

Box filter:

I do this part by calculating the average value in the box. If the box is over the image range, I will put value 0 in the out-of-range pixel.

Principal code fragment:

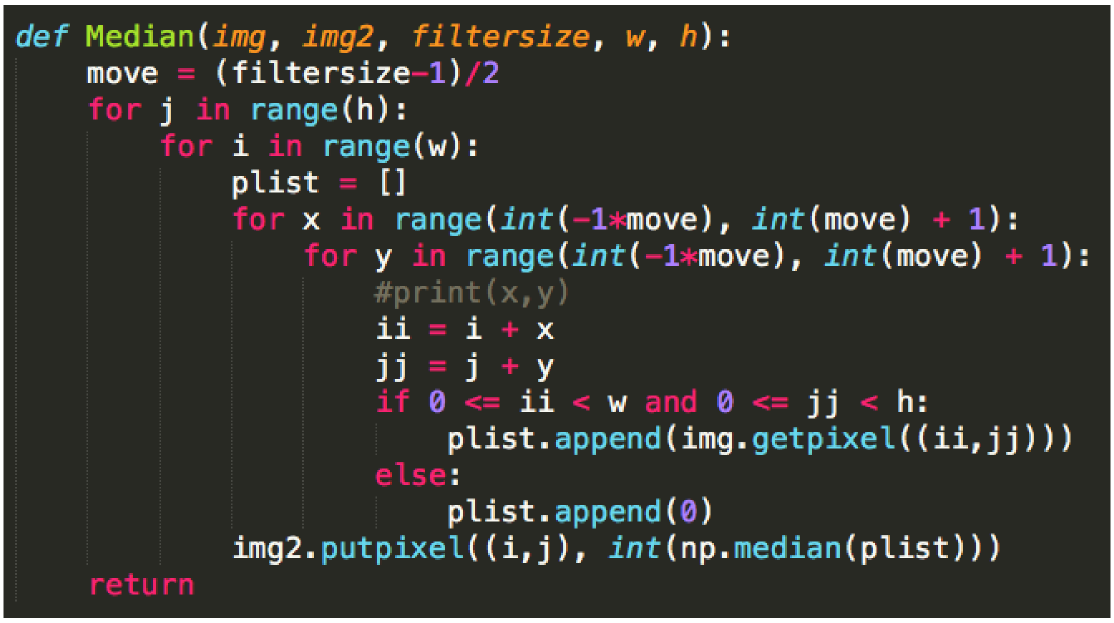


```python
def Box(img, img2, filtersize, w, h):
    move = (filtersize-1)/2
    for j in range(h):
        for i in range(w):
            pixelsum = 0
            for x in range(int(-1*move), int(move) + 1):
                for y in range(int(-1*move), int(move) + 1):
                    #print(x,y)
                    ii = i + x
                    jj = j + y
                    if 0 <= ii < w and 0 <= jj < h:
                        pixelsum += img.getpixel((ii,jj))
            img2.putpixel((i,j), int(pixelsum/(filtersize*filtersize)))
    return
```

Median filter:

I do this part by calculating the median value in the box. If the box is over the image range, I will put value 0 in the out-of-range pixel.

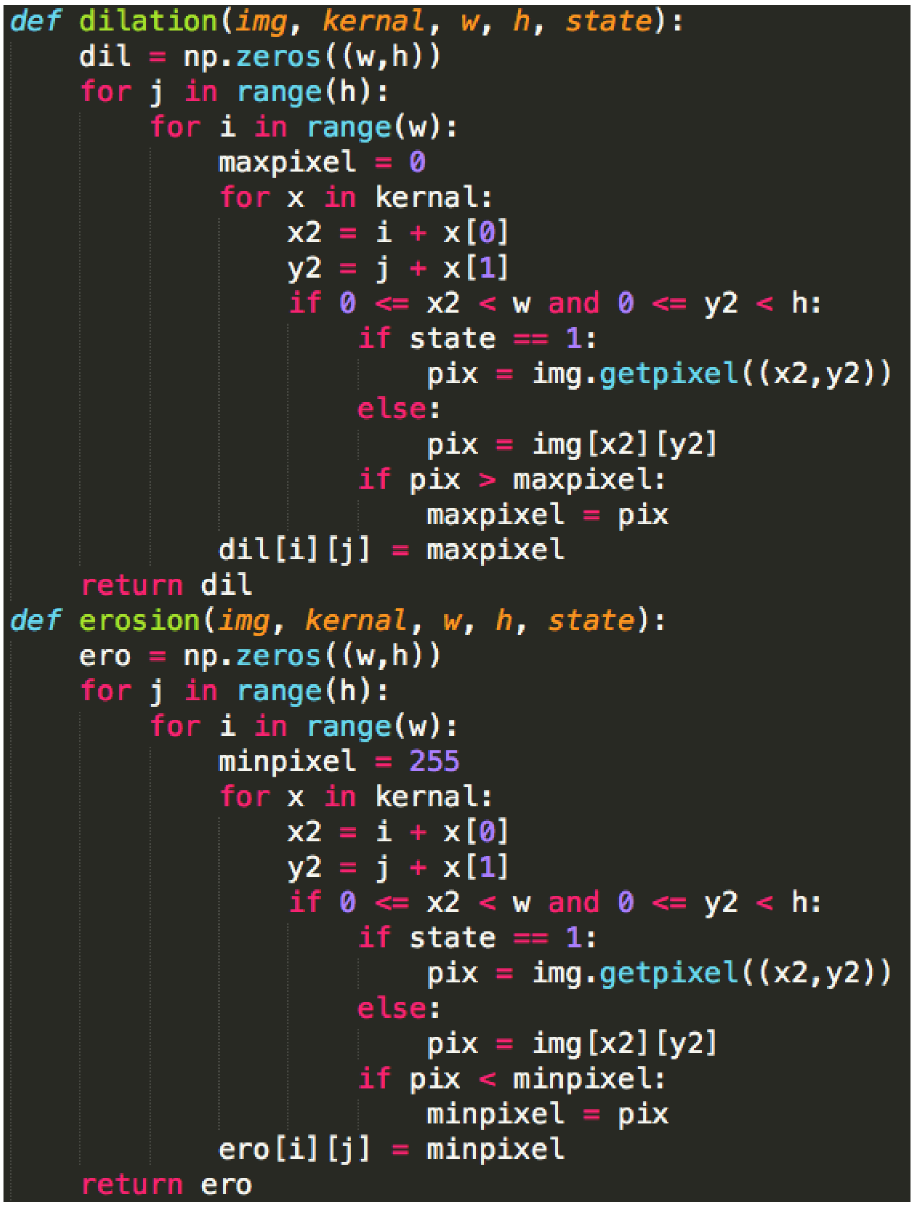
Principal code fragment:



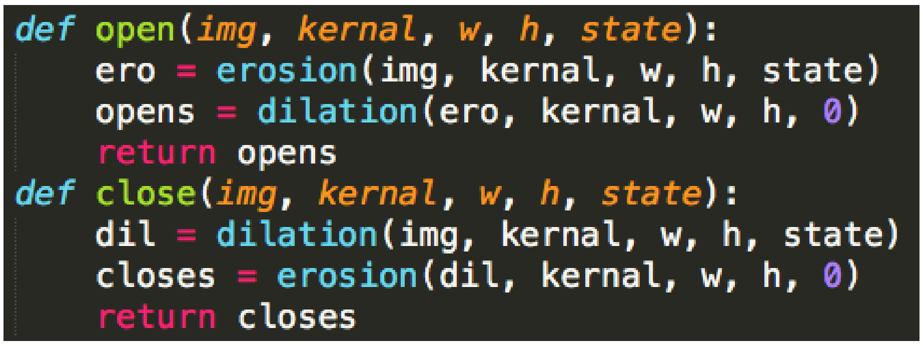
```python
def Median(img, img2, filtersize, w, h):
    move = (filtersize-1)/2
    for j in range(h):
        for i in range(w):
            plist = []
            for x in range(int(-1*move), int(move) + 1):
                for y in range(int(-1*move), int(move) + 1):
                    #print(x,y)
                    ii = i + x
                    jj = j + y
                    if 0 <= ii < w and 0 <= jj < h:
                        plist.append(img.getpixel((ii,jj)))
                    else:
                        plist.append(0)
            img2.putpixel((i,j), int(np.median(plist)))
    return
```

Opening and Closing:

The method doing opening and closing are taught in the gray scale morphology homework: opening is do dilation after erosion and closing is do erosion after dilation. So I just simply modify some part of the code and this part is done. Principal code fragment:



```python
def dilation(img, kernal, w, h, state):
    dil = np.zeros((w,h))
    for j in range(h):
        for i in range(w):
            maxpixel = 0
            for x in kernal:
                x2 = i + x[0]
                y2 = j + x[1]
                if 0 <= x2 < w and 0 <= y2 < h:
                    if state == 1:
                        pix = img.getpixel((x2,y2))
                    else:
                        pix = img[x2][y2]
                    if pix > maxpixel:
                        maxpixel = pix
            dil[i][j] = maxpixel
    return dil
def erosion(img, kernal, w, h, state):
    ero = np.zeros((w,h))
    for j in range(h):
        for i in range(w):
            minpixel = 255
            for x in kernal:
                x2 = i + x[0]
                y2 = j + x[1]
                if 0 <= x2 < w and 0 <= y2 < h:
                    if state == 1:
                        pix = img.getpixel((x2,y2))
                    else:
                        pix = img[x2][y2]
                    if pix < minpixel:
                        minpixel = pix
            ero[i][j] = minpixel
    return ero
```



```python
def open(img, kernal, w, h, state):
    ero = erosion(img, kernal, w, h, state)
    opens = dilation(ero, kernal, w, h, 0)
    return opens
def close(img, kernal, w, h, state):
    dil = dilation(img, kernal, w, h, state)
    closes = erosion(dil, kernal, w, h, 0)
    return closes
```

SNR:

I do this part by the formula downward, and I use math.log and math.sqrt to do the log and square calculation.

$$VS = \frac{\sum_{\forall n} (I(i,j) - \mu)^2}{\|n\|}$$

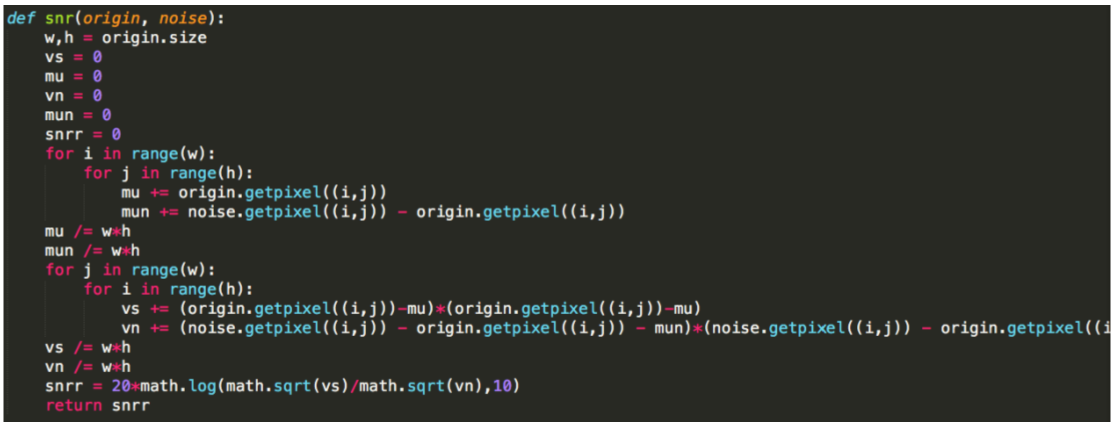
$$\mu = \frac{\sum_{\forall n} I(i,j)}{\|n\|}$$

$$VN = \frac{\sum_{\forall n} (I_N(i,j) - I(i,j) - \mu_N)^2}{\|n\|}$$

$$\mu_N = \frac{\sum_{\forall n} (I_N(i,j) - I(i,j))}{\|n\|}$$

$$SNR = 20 \log_{10} \frac{\sqrt{VS}}{\sqrt{VN}}$$

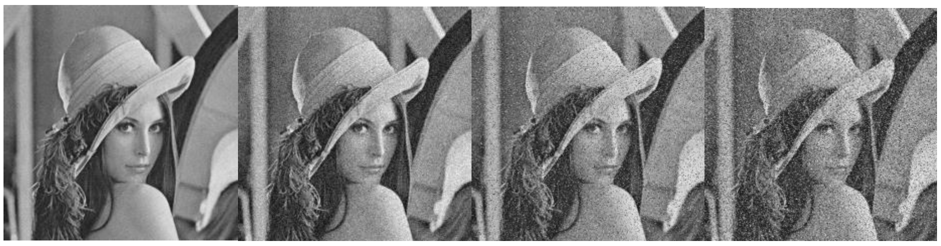
Principal code fragment:



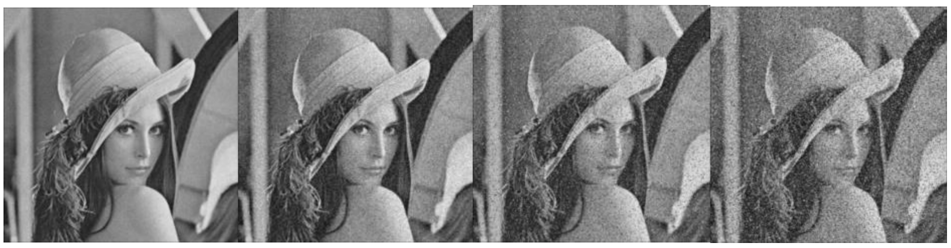
```python
def snr(origin, noise):
    w,h = origin.size
    vs = 0
    mu = 0
    vn = 0
    mun = 0
    snrr = 0
    for i in range(w):
        for j in range(h):
            mu += origin.getpixel((i,j))
            mun += noise.getpixel((i,j)) - origin.getpixel((i,j))
    mu /= w*h
    mun /= w*h
    for j in range(w):
        for i in range(h):
            vs += (origin.getpixel((i,j))-mu)*(origin.getpixel((i,j))-mu)
            vn += (noise.getpixel((i,j)) - origin.getpixel((i,j)) - mun)*(noise.getpixel((i,j)) - origin.getpixel((i
    vs /= w*h
    vn /= w*h
    snrr = 20*math.log(math.sqrt(vs)/math.sqrt(vn),10)
    return snrr
```

Results:

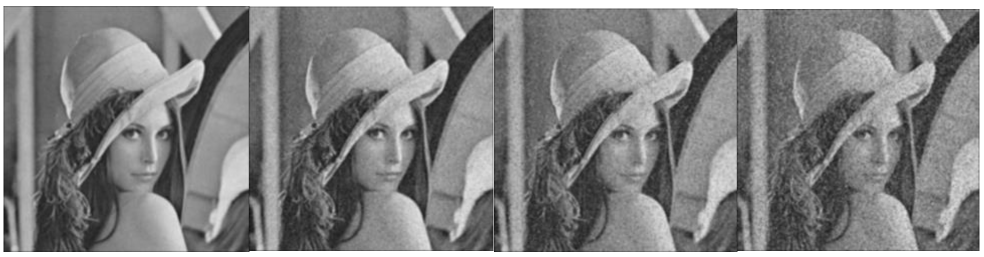
Noise (gaussian10, gaussian30, salt0.05, salt0.1):



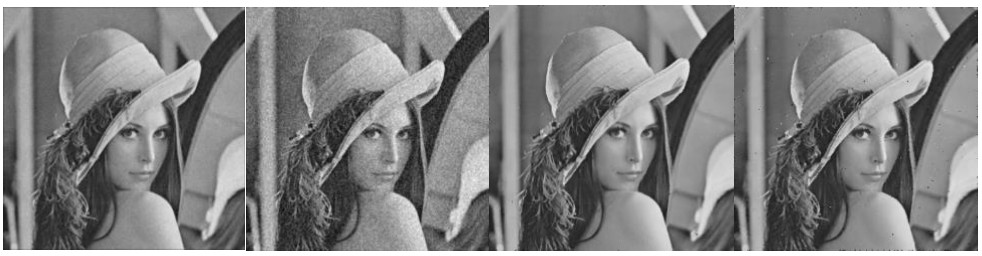
3x3 Box filter (gaussian10, gaussian30, salt0.05, salt0.1):



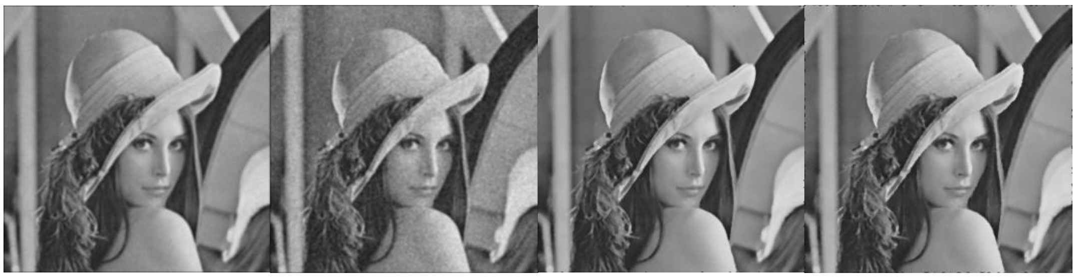
5x5 Box filter (gaussian10, gaussian30, salt0.05, salt0.1):



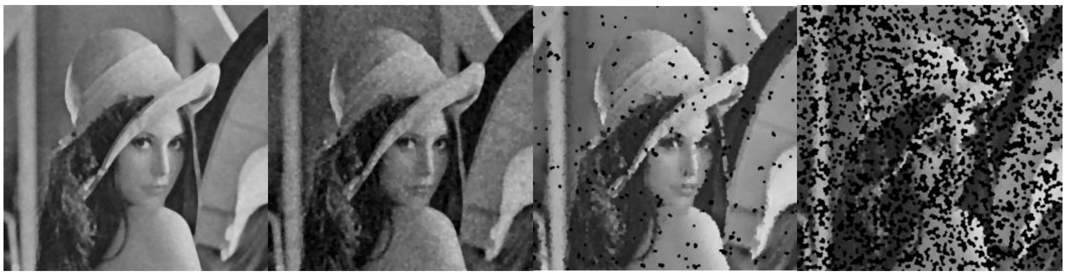
3x3 Median filter (gaussian10, gaussian30, salt0.05, salt0.1):



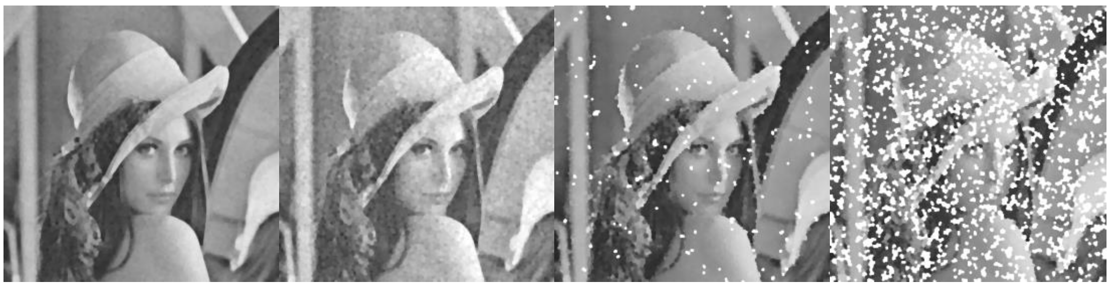
5x5 Median filter (gaussian10, gaussian30, salt0.05, salt0.1):



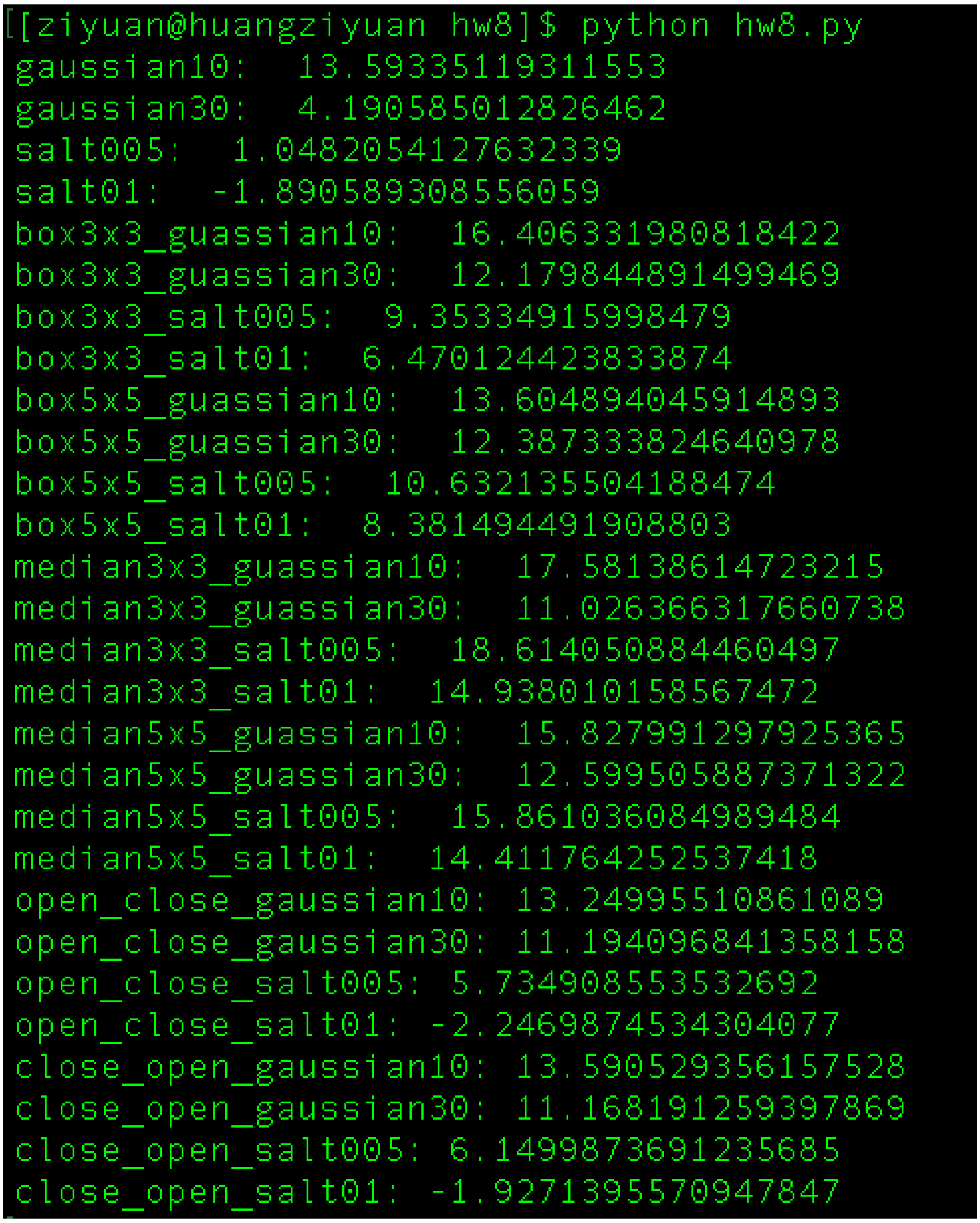
Opening and closing (gaussian10, gaussian30, salt0.05, salt0.1):



Closing and opening (gaussian10, gaussian30, salt0.05, salt0.1):



SNR:


```
[[ziyuan@huangziyuan hw8]$ python hw8.py
gaussian10:  13.59335119311553
gaussian30:  4.190585012826462
salt005:  1.0482054127632339
salt01:  -1.890589308556059
box3x3_guassian10:  16.406331980818422
box3x3_guassian30:  12.179844891499469
box3x3_salt005:  9.35334915998479
box3x3_salt01:  6.470124423833874
box5x5_guassian10:  13.604894045914893
box5x5_guassian30:  12.387333824640978
box5x5_salt005:  10.632135504188474
box5x5_salt01:  8.381494491908803
median3x3_guassian10:  17.58138614723215
median3x3_guassian30:  11.026366317660738
median3x3_salt005:  18.614050884460497
median3x3_salt01:  14.938010158567472
median5x5_guassian10:  15.827991297925365
median5x5_guassian30:  12.599505887371322
median5x5_salt005:  15.861036084989484
median5x5_salt01:  14.411764252537418
open_close_gaussian10: 13.24995510861089
open_close_gaussian30: 11.194096841358158
open_close_salt005: 5.734908553532692
open_close_salt01: -2.246987453430407
close_open_gaussian10: 13.590529356157528
close_open_gaussian30: 11.168191259397869
close_open_salt005: 6.149987369123568
close_open_salt01: -1.927139557094784
```