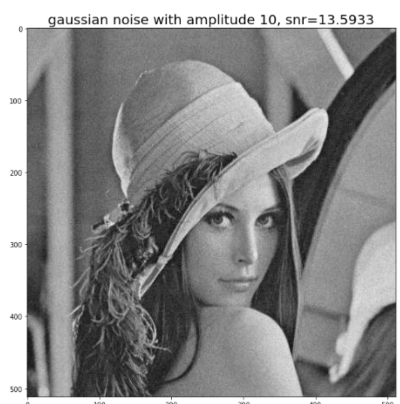
Computer Vision HW8

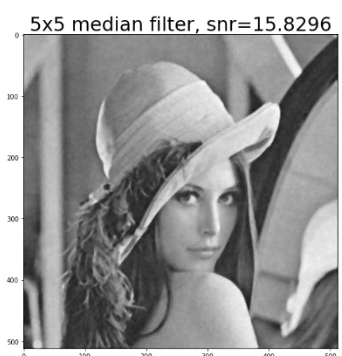
R08922a27 資工系 人工智慧碩士班 李吉昌

I use python 3.7 to implement all image processing requirements. Reading .bmp file by **PIL**, and then processing through **NumPy** array.

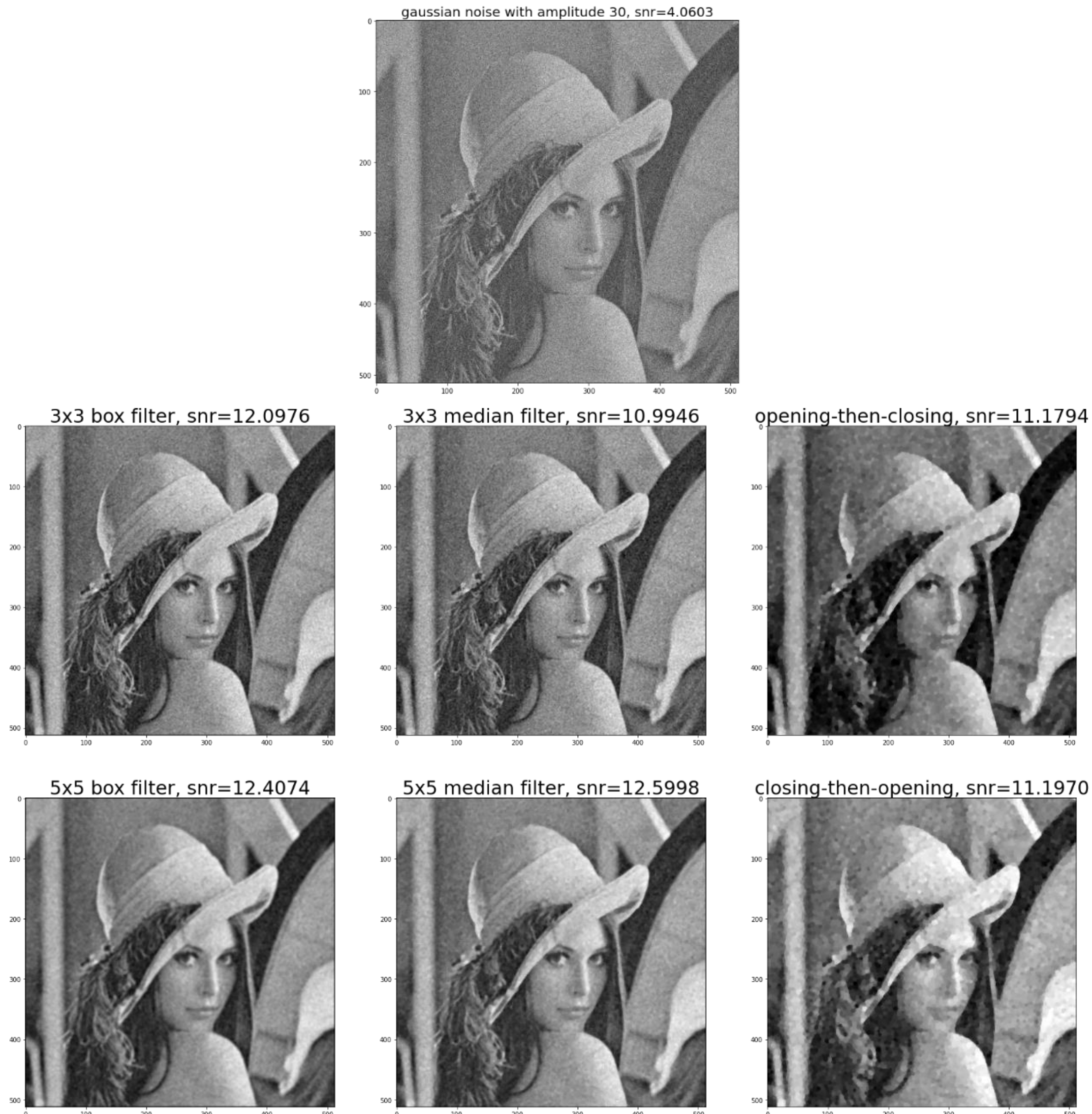
- ## 1. Results

Gaussian noise with amplitude of 10



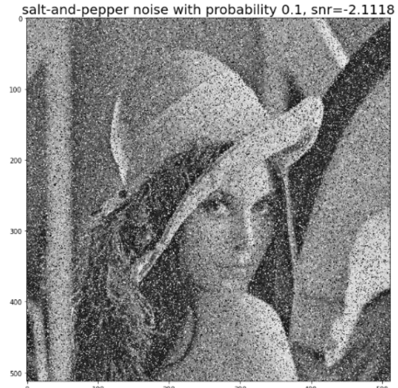


Gaussian noise with amplitude of 30



Salt-and-Pepper noise with probability 0.1

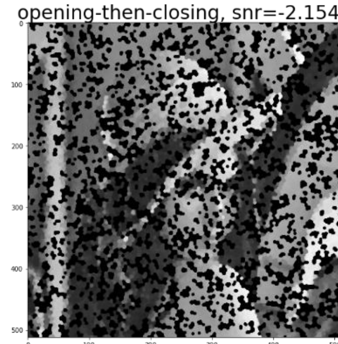
salt-and-pepper noise with probability 0.1, snr=-2.1118



3x3 box filter, snr=6.2063

3x3 median filter, snr=14.0706

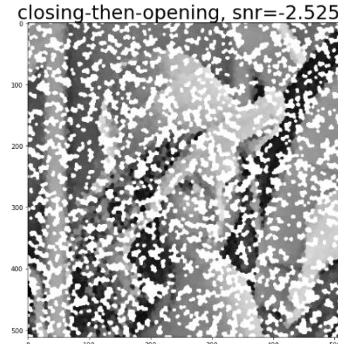
opening-then-closing, snr=-2.1546



5x5 box filter, snr=8.1617

5x5 median filter, snr=14.2072

closing-then-opening, snr=-2.5256



Salt-and-Pepper noise with probability 0.05

- # 2. Code fragment

```
def gaussian_noise_transform(img, amp):
    return img + amp * np.random.normal(0, 1, img.shape)

def salt_and_pepper_noise_transform(img, thr):
    prob_map = np.random.uniform(0, 1, img.shape)
    salt_idx = prob_map > 1 - thr
    pepper_idx = prob_map < thr
    img_sp = img.copy()
    img_sp[salt_idx] = 255
    img_sp[pepper_idx] = 0
    return img_sp
```

```
def padding(img, filter_size):
    img_pad = np.zeros((img.shape[0] + filter_size // 2 * 2, img.shape[1] +
filter_size // 2 * 2), np.int)
    for i in np.arange(filter_size // 2, img.shape[0] + filter_size // 2):
        for j in np.arange(filter_size // 2, img.shape[1] + filter_size //
2):
            img_pad[i, j] = img[i - filter_size // 2, j - filter_size // 2]
    return img_pad

def box_filter(img, filter_size):
    img_mean = np.zeros(img.shape)
    img_pad = padding(img, filter_size)
    for i in range(img_mean.shape[0]):
        for j in range(img_mean.shape[1]):
            img_mean[i, j] = img_pad[i: i + filter_size, j: j +
filter_size].mean()
    return img_mean

def median_filter(img, filter_size):
    img_med = np.zeros(img.shape)
    img_pad = padding(img, filter_size)
    for i in range(img_med.shape[0]):
        for j in range(img_med.shape[1]):
            img_med[i, j] = np.median(img_pad[i: i + filter_size, j: j +
filter_size])
    return img_med

def snr(img_org, img_pro):
    noise = img_pro - img_org
    return np.log10(img_org.var()/noise.var()) * 10
```

- ## 3. Brief Description

All the assigned operators' implementation details follow the course's lecture slides. The **np.random** was used to simulate all the assigned noise distribution. The filters were implemented by using looping to determine each processed pixel's value. Please note that the boundary pixel's values of the opening and closing operation's outputs are different, so that the SNR values would be a little different from the reference.