CV HW8

資工四 陳光裕 B07902072

程式架構:

Gaussian noise 跟 Salt and pepper noise

```python
def gaussian_noise(img, amptitude):
    new_img = np.copy(img)
    width, height = img.shape[:2]
    for i in range(width):
        for j in range(height):
            gauss_val = img[i,j] + amptitude*np.random.normal(0, 1)
            if gauss_val < 0:
                new_img[i,j] = 0
            elif gauss_val > 255:
                new_img[i,j] = 255
            else:
                new_img[i,j] = int(gauss_val)
    return new_img

def salt_and_pepper(img, threshold):
    width, height = img.shape[:2]
    new_img = np.copy(img)
    for i in range(width):
        for j in range(height):
            rand_val = random.uniform(0, 1)
            if rand_val < threshold:
                new_img[i,j] = 0
            elif rand_val > 1-threshold:
                new_img[i,j] = 255
            else:
                new_img[i,j] = img[i,j]
    return new_img
```

Opening 跟 Closing 在作業五實做過就不附程式碼

我自己做了 Padding，根據 filter 大小不同去做邊緣的擴張

```python
def padding(img, expand):
    width, height = img.shape[:2]
    new_img = np.zeros((width+2, height+2))
    for i in range(width):
        for j in range(height):
            new_img[i+1,j+1] = img[i,j]
    new_img[0,0] = img[0,0]
    new_img[width+1, 0] = img[width-1, 0]
    new_img[width+1, width+1] = img[width-1, width-1]
    new_img[0, width+1] = img[0, width-1]
    for i in range(1, width+2):
        new_img[0, i] = new_img[1, i]
        new_img[width+1, i] = new_img[width, i]
        new_img[i, 0] = new_img[i, 1]
        new_img[i, width+1] = new_img[i, width]
    if expand == 3:
        return new_img
    elif expand == 5:
        return padding(new_img, 3)
```

Box filter 跟 Median filter

```python
def box_filter(img, filter_size):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    if filter_size == 3:
        filter_kernel = filter_3
        padding_img = padding(img, 3)
        for i in range(1, width+1):
            for j in range(1, height+1):
                # cal mean
                mean = 0.0
                N = 9
                for x, y in filter_kernel:
                    mean += padding_img[i+x, j+y]
                mean /= N
                new_img[i-1,j-1] = int(mean)
    if filter_size == 5:
        filter_kernel = filter_5
        padding_img = padding(img, 5)
        for i in range(2, width+2):
            for j in range(2, height+2):
                # cal mean
                mean = 0.0
                N = 25
                for x, y in filter_kernel:
                    mean += padding_img[i+x, j+y]
                mean /= N
                new_img[i-2,j-2] = mean
    return new_img
```

```python
def median_filter(img, filter_size):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    if filter_size == 3:
        filter_kernel = filter_3
        padding_img = padding(img, 3)
        for i in range(1, width+1):
            for j in range(1, height+1):
                # cal median
                pixel_list=[]
                for x, y in filter_kernel:
                        pixel_list.append(padding_img[i+x, j+y])
                pixel_list.sort()
                idx = int(len(pixel_list)/2)
                new_img[i-1,j-1] = pixel_list[idx]
    if filter_size == 5:
        filter_kernel = filter_5
        padding_img = padding(img, 5)
        for i in range(2, width+2):
            for j in range(2, height+2):
                # cal median
                pixel_list=[]
                for x, y in filter_kernel:
                        pixel_list.append(padding_img[i+x, j+y])
                pixel_list.sort()
                idx = int(len(pixel_list)/2)
                new_img[i-2,j-2] = pixel_list[idx]
    return new_img
```

計算 Signal-to-Ratio(SNR):

```python
def mu_s(img):
    width, height = img.shape[:2]
    N = width * height
    mean = 0.0
    for i in range(width):
        for j in range(height):
            mean += img[i,j]
    return mean/N

def VS(img):
    width, height = img.shape[:2]
    N = width * height
    vs = 0.0
    mean = mu_s(img)
    for i in range(width):
        for j in range(height):
            vs += (img[i,j] - mean) ** 2
    return vs/N

def mu_noise(img, noise_img):
    width, height = img.shape[:2]
    N = width * height
    mn = 0.0
    for i in range(width):
        for j in range(height):
            mn += (noise_img[i,j] - img[i,j])
    return mn/N
```

```python
def VN(img, noise_img):
    width, height = img.shape[:2]
    N = width * height
    vn = 0.0
    mn = mu_noise(img, noise_img)
    for i in range(width):
        for j in range(height):
            vn += ((noise_img[i,j] - img[i,j]) - mn) ** 2
    return vn/N

def normalize(img):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    for i in range(width):
        for j in range(height):
            new_img[i, j] = img[i, j]/255*1
    return new_img

def SNR(img, compare_img):
    nor_img = normalize(img)
    nor_cmp = normalize(compare_img)
    return 20 * math.log((math.sqrt(VS(nor_img)) / math.sqrt(VN(nor_img, nor_cmp))), 10)
```

Noisy image with gaussian noise, amplitude=10:

3x3 Box filter	5x5 Box filter
	
3x3 Median filter	5x5 Median filter
	
Opening then closing	Closing then opening

Noisy image with gaussian noise, amplitude=30

3x3 Box filter

5x5 Box filter

3x3 Median filter

5x5 Median filter

Opening then closing

Closing then opening

Noisy image with salt-and-pepper noise, probability=0.1

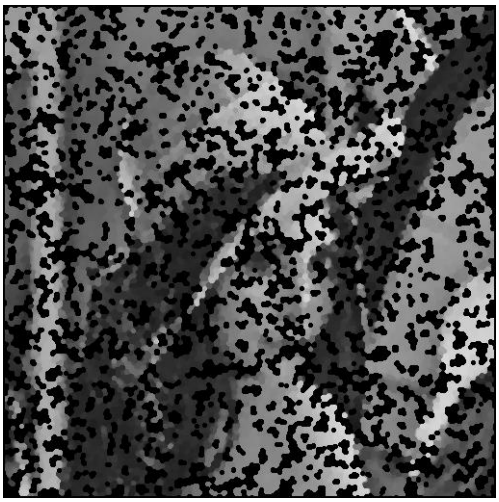
3x3 Box filter	5x5 Box filter

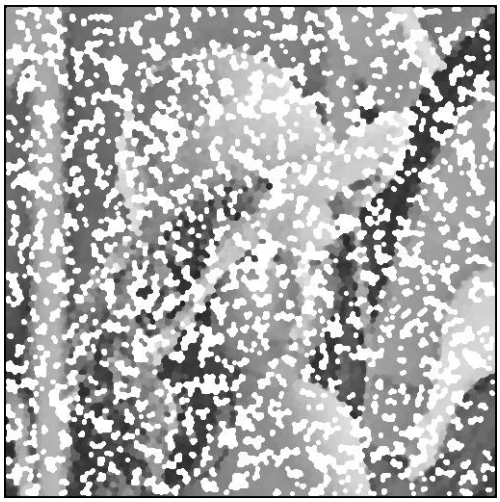
3x3 Median filter

5x5 Median filter

Opening then closing

Closing then opening





Noisy image with salt-and-pepper noise, probability=0.05

3x3 Box filter

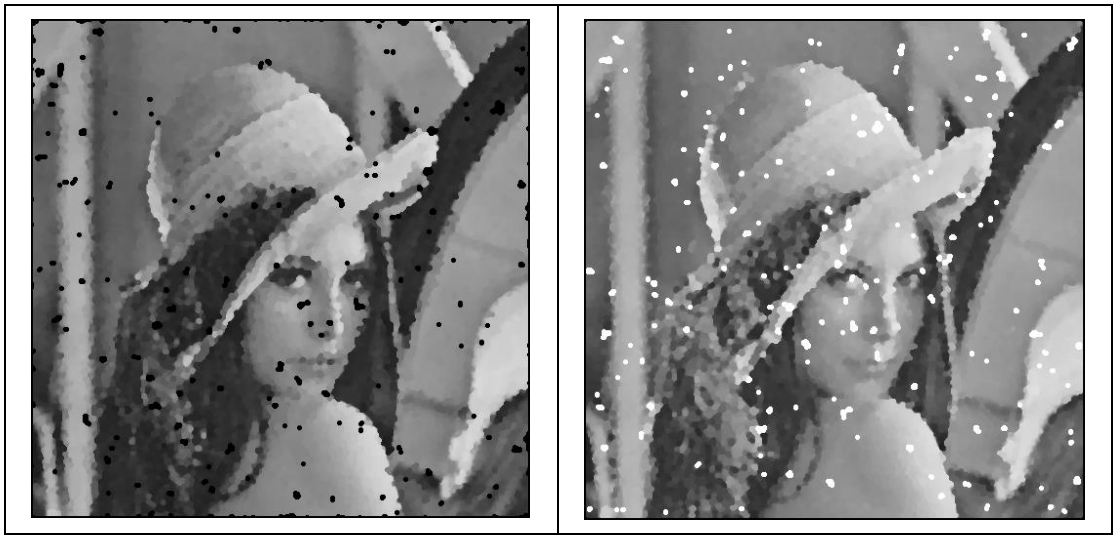
5x5 Box filter

3x3 Median filter

5x5 Median filter

Opening then closing

Closing then opening



Signal-to-Ratio(SNR):

Gaussian 10	Gaussian 30	Salt pepper 0.1	Salt pepper 0.05
13.569	4.167	-2.091	0.922

SNR	3x3 Box	5x5 Box	3x3 Median	5x5 Median	Open close	Close open
Gaussian 10	17.744	14.872	17.642	16.003	8.624	7.645
Gaussian 30	12.593	13.267	11.059	12.839	8.630	6.082
Salt pepper 0.1	6.346	8.509	14.907	15.720	-2.349	-2.925
Salt pepper 0.05	9.444	11.148	19.099	16.383	4.693	3.925

```
Gaussian 10= 13.569170680880756
Gaussian 30= 4.167279018743168
Salt and pepper 0.1= -2.0918700233977456
Salt and pepper 0.05= 0.9222741709391878
Gau 10 3x3 Box= 17.7443905549444406
Gau 30 3x3 Box= 12.593881497004356
Gau 10 5x5 Box= 14.87226071204067
Gau 30 5x5 Box= 13.267543955782262
SaltPepper 0.1 3x3 Box= 6.346462202083587
SaltPepper 0.05 3x3 Box= 9.44468894776518
SaltPepper 0.1 5x5 Box= 8.509037550159853
SaltPepper 0.05 5x5 Box= 11.14849647084187
Gau 10 3x3 Median= 17.64267217176249
Gau 30 3x3 Median= 11.059047357652148
Gau 10 5x5 Median= 16.003234809225553
Gau 30 5x5 Median= 12.83998123367606
SaltPepper 0.1 3x3 Median= 14.907835232274342
SaltPepper 0.05 3x3 Median= 19.099718015998047
SaltPepper 0.1 5x5 Median= 15.7201355474141241
SaltPepper 0.05 5x5 Median= 16.383495440541036
Gau 10 OtC= 8.6248363359726649
Gau 30 OtC= 8.630671444928291
Gau 10 CtO= 7.64590914869411
Gau 30 CtO= 6.08273594635592
SaltPepper 0.1 OtC= -2.3497496157497584
SaltPepper 0.05 OtC= 4.693437940193018
SaltPepper 0.1 CtO= -2.9254562357692753
SaltPepper 0.05 CtO= 3.925100320356459
```