Computer Vision Homework #8

Student-ID: r10944020; Name: 林顥倫; Department: GINM

[Results]

- **SNR** 計算執行結果

```
(base) ➜  hw8 python3 hw8.py
SNR of image amp_10_img is 13.587635
SNR of image amp_30_img is 4.166017
SNR of image salt_and_pepper_005 is 0.930780
SNR of image salt_and_pepper_010 is -2.084784
SNR of image gaussian_noise_amp_10_boxFilter_3.bmp is 11.126948
SNR of image gaussian_noise_amp_10_boxFilter_5.bmp is 11.239321
SNR of image gaussian_noise_amp_30_boxFilter_3.bmp is 9.423687
SNR of image gaussian_noise_amp_30_boxFilter_5.bmp is 10.513988
SNR of image salt_and_pepper_010_boxFilter_3.bmp is 5.498357
SNR of image salt_and_pepper_010_boxFilter_5.bmp is 7.543221
SNR of image salt_and_pepper_005_boxFilter_3.bmp is 7.748931
SNR of image salt_and_pepper_005_boxFilter_5.bmp is 9.352107
SNR of image gaussian_noise_amp_10_medianFilter_3.bmp is 11.382461
SNR of image gaussian_noise_amp_10_medianFilter_5.bmp is 11.858098
SNR of image gaussian_noise_amp_30_medianFilter_3.bmp is 8.925113
SNR of image gaussian_noise_amp_30_medianFilter_5.bmp is 10.722641
SNR of image salt_and_pepper_010_medianFilter_3.bmp is 10.570159
SNR of image salt_and_pepper_010_medianFilter_5.bmp is 11.747935
SNR of image salt_and_pepper_005_medianFilter_3.bmp is 11.555443
SNR of image salt_and_pepper_005_medianFilter_5.bmp is 11.885192
SNR of image gaussian_noise_amp_10_closing-then-opening.bmp is 13.571137
SNR of image gaussian_noise_amp_10_opening-then-closing.bmp is 13.248241
SNR of image gaussian_noise_amp_30_closing-then-opening.bmp is 11.143020
SNR of image gaussian_noise_amp_30_opening-then-closing.bmp is 11.136029
SNR of image salt_and_pepper_010_closing-then-opening.bmp is -2.521927
SNR of image salt_and_pepper_010_opening-then-closing.bmp is -2.105385
SNR of image salt_and_pepper_005_closing-then-opening.bmp is 5.211238
SNR of image salt_and_pepper_005_opening-then-closing.bmp is 6.058366
```

- 圖片生成結果

(a) Generate noisy images with gaussian noise (amplitude of 10[左] and 30[右])


(b) Generate noisy images with salt-and-pepper noise (probability 0.1[左] and 0.05[右])


(c) Use the 3x3, 5x5 box filter on images generated by (a)(b)

[3x3 - Gaussian amp = 10]

[3x3 - Gaussian amp = 30]

[3x3 - Salt and Pepper prob = 0.1]

[3x3 - Salt and Pepper prob = 0.05]

[5x5 - Gaussian amp = 10]

[5x5 - Gaussian amp = 30]

[5x5 - Salt and Pepper prob = 0.1]

[5x5 - Salt and Pepper prob = 0.05]

(d) Use 3x3, 5x5 median filter on images generated by (a)(b)

[3x3 - Gaussian amp = 10]

[3x3 - Gaussian amp = 30]

[3x3 - Salt and Pepper prob = 0.1]

[3x3 - Salt and Pepper prob = 0.05]

[5x5 - Gaussian amp = 10]

[5x5 - Gaussian amp = 30]

[5x5 - Salt and Pepper prob = 0.1]

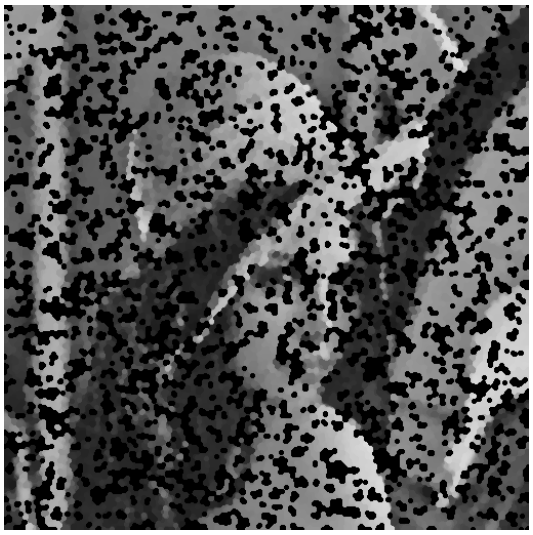
[5x5 - Salt and Pepper prob = 0.05]

(e) Use both opening-then-closing and closing-then-opening filter (using the octogonal 3-5-5-5-3 kernel, value = 0) on images generated by (a)(b)

[opening-then-closing - Gaussian amp = 10]

[opening-then-closing - Gaussian amp = 30]

[opening-then-closing - Salt and Pepper prob = 0.1]

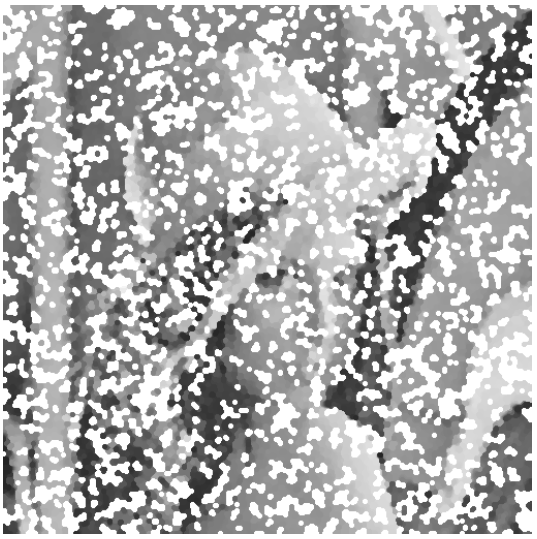


[opening-then-closing - Salt and Pepper prob = 0.05]

[closing-then-opening - Gaussian amp = 10]

[closing-then-opening - Gaussian amp = 30]

[closing-then-opening - Salt and Pepper prob = 0.1] [closing-then-opening - Salt and Pepper prob = 0.05]



[Code Fragment & Explanation]

Part-1 SNR Computation 根據投影片的公式計算, 先除以255將數值normalize到0-1

```
60 def computeSNR(img, img_res, img_res_name):
61     signal, noise = img / 255, img_res / 255
62     signal_mean, noise_mean, signal_var, noise_var = np.mean(signal), 0, 0, 0
63     imageW, imageH = img.shape
64     for x in range(imageW):
65         for y in range(imageH):
66             noise_mean += (noise[x][y] - signal[x][y])
67     noise_mean = noise_mean / ( imageW * imageH )
68     for x in range(imageW):
69         for y in range(imageH):
70             signal_var += math.pow(signal[x][y] - signal_mean, 2)
71             noise_var += math.pow((noise[x][y] - signal[x][y]) - noise_mean, 2)
72     signal_var = signal_var / ( imageW * imageH )
73     noise_var = noise_var / ( imageW * imageH )
74     SNR_val = 20 * math.log( math.sqrt(signal_var) / math.sqrt(noise_var), 10)
75     print("SNR of image %s is %f" % (img_res_name, SNR_val))
```

Part-2 (a)(b) Gaussian Noise & Salt-and-pepper noise 根據投影片的公式計算

```
42 def generate_gaussian_noise_img(img, mu, sigma, amp):
43     img_res = img + amp * np.random.normal(mu, sigma, img.shape)
44     for x in range(img.shape[0]):
45         for y in range(img.shape[1]):
46             if img_res[x][y] > 255: img_res[x][y] = 255
47             if img_res[x][y] < 0: img_res[x][y] = 0
48     return img_res
49
50 def generate_salt_and_pepper(img, prob):
51     distribution_map = np.random.uniform(0, 1, img.shape)
52     imageW, imageH = img.shape[0], img.shape[1]
53     img_res = img.copy() #np.zeros((imageW, imageH), dtype='int32')
54     for x in range(imageW):
55         for y in range(imageH):
56             if distribution_map[x][y] < prob: img_res[x][y] = 0
57             elif distribution_map[x][y] > 1 - prob: img_res[x][y] = 255
58     return img_res
```

Part-3 (c)(d) Box Filter & Median Filter 根據投影片的公式計算

```python
def boxFilter(img, kernel_size):
    imageW, imageH = img.shape
    img_res = np.zeros((imageW, imageH), dtype='int32')
    normalization_term = kernel_size * kernel_size
    kernel = []
    for x in range(-kernel_size//2, kernel_size//2):
        for y in range(-kernel_size//2, kernel_size//2):
            kernel.append([x,y])
    for x in range(imageW):
        for y in range(imageH):
            for [dx, dy] in kernel:
                if x+dx >= 0 and x+dx < imageW and y+dy >= 0 and y+dy < imageH:
                    img_res[x][y] += img[x+dx][y+dy]
            img_res[x][y] = img_res[x][y] / normalization_term
    return img_res

def medianFilter(img, kernel_size):
    imageW, imageH = img.shape
    img_res = np.zeros((imageW, imageH), dtype='int32')
    kernel = []
    for x in range(-kernel_size//2, kernel_size//2):
        for y in range(-kernel_size//2, kernel_size//2):
            kernel.append([x,y])
    for x in range(imageW):
        for y in range(imageH):
            pixel_vals = []
            for [dx, dy] in kernel:
                if x+dx >= 0 and x+dx < imageW and y+dy >= 0 and y+dy < imageH:
                    pixel_vals.append(img[x+dx][y+dy])
            img_res[x][y] = np.median(pixel_vals)
    return img_res
```

Part-4 (e) Opening and Closing 根據投影片的公式計算, 這部分程式碼根據先前作業修改

```python
def validPixel(x, bound):
    return (x >= 0 and x <= bound)

def Dilation(img, kernel):
    imageW, imageH = img.shape
    new_img = np.zeros((imageW, imageH), dtype='int32')
    for x in range(imageW):
        for y in range(imageH):
            localMaximum = 0
            for [ex, ey] in kernel:
                dest_x, dest_y = x + ex, y + ey
                if ( validPixel(dest_x, imageW-1) and validPixel(dest_y, imageH-1) ):
                    localMaximum = max( localMaximum, img[dest_x, dest_y])
            new_img[x, y] = localMaximum
    return new_img

def Erosion(img, kernel):
    imageW, imageH = img.shape
    new_img = np.zeros((imageW, imageH), dtype='int32')
    for x in range(imageW):
        for y in range(imageH):
            localMinimum = 255
            for [ex, ey] in kernel:
                dest_x, dest_y = x + ex, y + ey
                if ( validPixel(dest_x, imageW-1) and validPixel(dest_y, imageH-1) ): # valid pixel
                    localMinimum = min( localMinimum, img[dest_x, dest_y]) # erosion
            new_img[x, y] = localMinimum
    return new_img

def Opening(img, kernel):
    return Dilation(Erosion(img, kernel), kernel)

def Closing(img, kernel):
    return Erosion(Dilation(img, kernel), kernel)
```

Part-5 Main Function

```python
if __name__ == '__main__':
    img = cv2.imread("./lena.bmp", cv2.IMREAD_GRAYSCALE)

    # (a) Generate noisy images with gaussian noise(amplitude of 10 and 30)
    amp_10_img = generate_gaussian_noise_img(img, 0, 1, 10)
    cv2.imwrite('gaussian_noise_amp_10.png', amp_10_img)
    amp_30_img = generate_gaussian_noise_img(img, 0, 1, 30)
    cv2.imwrite('gaussian_noise_amp_30.png', amp_30_img)
    computeSNR(img, amp_10_img, "amp_10_img")
    computeSNR(img, amp_30_img, "amp_30_img")

    # (b) Generate noisy images with salt-and-pepper noise( probability 0.1 and 0.05)
    salt_and_pepper_010 = generate_salt_and_pepper(img, 0.10)
    cv2.imwrite('salt_and_pepper_010.png', salt_and_pepper_010)
    salt_and_pepper_005 = generate_salt_and_pepper(img, 0.05)
    cv2.imwrite('salt_and_pepper_005.png', salt_and_pepper_005)
    computeSNR(img, salt_and_pepper_005, "salt_and_pepper_005")
    computeSNR(img, salt_and_pepper_010, "salt_and_pepper_010")

    # (c) Use the 3x3, 5x5 box filter on images generated by (a)(b)
    imgs_name = ["gaussian_noise_amp_10", "gaussian_noise_amp_30", "salt_and_pepper_010", "salt_and_pepper_005"]
    imgs = [amp_10_img, amp_30_img, salt_and_pepper_010, salt_and_pepper_005]
    kernel_sizes = [3,5]
    for (current_img_name, current_img) in zip(imgs_name, imgs):
        for kernel_size in kernel_sizes:
            img_res_name = current_img_name + "_boxFilter_" + str(kernel_size) + ".png"
            img_res = boxFilter(current_img, kernel_size)
            computeSNR(img, img_res, img_res_name)
            cv2.imwrite(img_res_name, img_res)

    # (d) Use 3x3, 5x5 median filter on images generated by (a)(b)
    imgs_name = ["gaussian_noise_amp_10", "gaussian_noise_amp_30", "salt_and_pepper_010", "salt_and_pepper_005"]
    imgs = [amp_10_img, amp_30_img, salt_and_pepper_010, salt_and_pepper_005]
    kernel_sizes = [3,5]
    for (current_img_name, current_img) in zip(imgs_name, imgs):
        for kernel_size in kernel_sizes:
            img_res_name = current_img_name + "_medianFilter_" + str(kernel_size) + ".png"
            img_res = medianFilter(current_img, kernel_size)
            computeSNR(img, img_res, img_res_name)
            cv2.imwrite(img_res_name, img_res)

    # (e) Use both opening-then-closing and closing-then opening filter
    # (using the octogonal 3-5-5-5-3 kernel, value = 0) on images generated by (a)(b)
    # kernel is a 3-5-5-5-3 octagon, where the orgin is at the center
    kernel = [    [-2, -1], [-2, 0], [-2, 1],
        [-1, -2], [-1, -1], [-1, 0], [-1, 1], [-1, 2],
        [0, -2],  [0, -1],  [0, 0],  [0, 1],  [0, 2],
        [1, -2],  [1, -1],  [1, 0],  [1, 1],  [1, 2],
                  [2, -1],  [2, 0],  [2, 1]]
    imgs_name = ["gaussian_noise_amp_10", "gaussian_noise_amp_30", "salt_and_pepper_010", "salt_and_pepper_005"]
    imgs = [amp_10_img, amp_30_img, salt_and_pepper_010, salt_and_pepper_005]
    funcs = ["closing-then-opening", "opening-then-closing"]
    for (current_img_name, current_img) in zip(imgs_name, imgs):
        for func in funcs:
            img_res_name = current_img_name + "_" + func + ".png"
            if func == "closing-then-opening":
                img_res = Opening(Closing(current_img, kernel), kernel)
            elif func == "opening-then-closing":
                img_res = Closing(Opening(current_img, kernel), kernel)
            computeSNR(img, img_res, img_res_name)
            cv2.imwrite(img_res_name, img_res)
```