R07922141
張緣彩
11/24/2018

Computer Vision HW8 Report

- **Implementation**

  Signal-to-Ratio (SNR) function is same as the equations described in website.

```
def getSNR(image, noise_image):
    total = float(512**2)

    # calculate mean
    m1, m2 = 0.0, 0.0
    for i in range(512):
        for j in range(512):
            m1 += float(image[i][j])
            m2 += float(noise_image[i][j]) - float(image[i][j])
    m1 /= total
    m2 /= total

    # calculate variance
    v1, v2 = 0.0, 0.0
    for i in range(512):
        for j in range(512):
            v1 += (float(image[i][j]) - m1)**2
            v2 += (float(noise_image[i][j]) - float(image[i][j]) - m2)**2
    v1 /= total
    v2 /= total

    return 20 * math.log(math.sqrt(v1)/math.sqrt(v2), 10)
```

  Gaussian noise generate function is same as the equation described in the lecture note. Given Img[x, y] = Img[x, y] + amp*gaussian distribution(0, 1) with mean of 0 and standard deviation of 1.

```
def GaussianNoise(image, amplitude):
    noise_lena = copy.deepcopy(image)

    for i in range(512):
        for j in range(512):
            noise = image[i][j] + amplitude*random.gauss(0,1)
            noise_lena[i][j] = max(0, min(255, int(noise)))

    return noise_lena
```

  Salt and Pepper noise generate function is also same as the equation described in the lecture note. Given a probability, sample a value from uniform distribution of (0, 1) then if the value is smaller than the probability or larger than 1 - probability. The pixel will be given 0 or 255 respectively, otherwise it is original value.

```
def SaltAndPepperNoise(image, probability):
    noise_lena = copy.deepcopy(image)

    for i in range(512):
        for j in range(512):
            r = random.uniform(0, 1)
            if r < probability:
                noise_lena[i][j] = 0
            elif r > 1 - probability:
                noise_lena[i][j] = 255

    return noise_lena
```

Box filter function is same as the equation described in the lecture note. Given a kernel, the value of the pixel is actually the mean of its neighbors. I use a list to save its neighbor's value then calculate its mean and assign back to it.

```
def BoxFilter(image, kernel):
    filtered_lena = copy.deepcopy(image)

    if kernel == (3, 3):
        k = get_neighbour((1, 1), kernel33)
    else:
        k = get_neighbour((2, 2), kernel55)

    for i in range(512):
        for j in range(512):
            lst = []
            for t in k:
                if (i+t[0] >= 0 and i+t[0] <= 511) and (j+t[1] >= 0  and j+t[1] <= 511):
                    lst.append(image[i+t[0]][j+t[1]])

            filtered_lena[i][j] = int(sum(lst)/len(lst))

    return filtered_lena
```

Median filter function is same as box filter (described in the lecture note as well). Then only different is we use the median of its neighbor instead of mean. Given a kernel, the value of the pixel is actually the median of its neighbors. I use a list to save its neighbor's value then find the median from the list and assign back to it.

```
def MedianFilter(image, kernel):
    filtered_lena = copy.deepcopy(image)

    if kernel == (3, 3):
        k = get_neighbour((1, 1), kernel33)
    else:
        k = get_neighbour((2, 2), kernel55)

    for i in range(512):
        for j in range(512):
            lst = []
            for t in k:
                if (i+t[0] >= 0 and i+t[0] <= 511) and (j+t[1] >= 0  and j+t[1] <= 511):
                    lst.append(image[i+t[0]][j+t[1]])

            # median
            med = int(len(lst)/2)
            lst.sort()
            if len(lst) % 2 == 0:
                tmp = int((int(lst[med-1]) + int(lst[med]))/2)
            else:
                tmp = lst[med]

            filtered_lena[i][j] = tmp

    return filtered_lena
```

- **Result**

SNR

	Noise	Box 3x3	Box 5x5	Med 3x3	Med 5x5	O then C	C then O
G 10	13.608	17.743	14.863	17.666	16.008	13.273	13.590
G 30	4.177	12.602	13.306	11.098	12.900	11.191	11.201
SAP 0.05	0.951	9.497	11.196	19.170	16.360	5.853	5.377
SAP 0.1	-2.097	6.318	8.505	14.848	15.787	-2.191	-2.457

IMAGE

Gaussian noise - 10, 30


Salt and Pepper noise - 0.05, 0.1


Gaussian 10 - BOX filter - 3x3 5x5

Gaussian 30 - BOX filter - 3x3 5x5

Salt and Pepper 0.05 - BOX filter - 3x3, 5x5

Salt and Pepper 0.1 - BOX filter - 3x3, 5x5

Gaussian 10 - MEDIAN filter - 3x3 5x5

Gaussian 30 - MEDIAN filter - 3x3 5x5

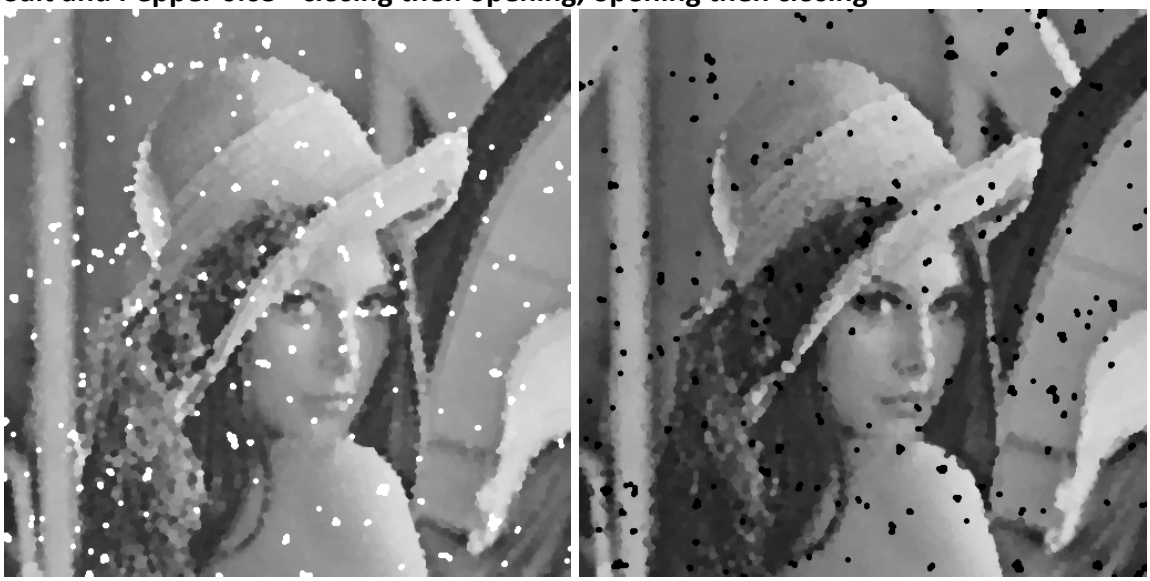
Salt and Pepper 0.05 - MEDIAN filter - 3x3, 5x5

Salt and Pepper 0.1 - MEDIAN filter - 3x3, 5x5

Gaussian 10 – closing then opening, opening then closing

Gaussian 30 - closing then opening, opening then closing

Salt and Pepper 0.05 - closing then opening, opening then closing



Salt and Pepper 0.1 - closing then opening, opening then closing