電機所-陳治言-R06921084

Description：

一開始，先處理圖片檔的標頭資訊，包括 size，offset，width，height，bits 等資訊，之後再將圖片的色彩值讀入，因為這次範例圖片只有 8 位元色彩，所以每個畫素只需要 1byte 即可儲存。而這次的作業是先對圖片做噪音，分為 Gaussian 與 Salt&Pepper，並使用不同的 4 種強度。完成之後再對圖片進行除噪操作，使用 4 種不同的 filter，分別 Box filter、Median filter、OpeningClosing filter 與 ClosingOpening filter，將噪音去除完成之後，再將標頭與畫素資料一起寫出並存，並且產生 24 張噪音去除圖片與 4 張噪音化的原始圖片，並計算 SNR 值，即可完成這次的作業。

Algorithm：

Gaussian 噪音，是使用亂數種子，透過公式去計算放大倍率，並將原始畫素加上強度乘以倍率，來得到新的畫素成為噪音。

Salt&Pepper 噪音，是透過亂數種子來產生機率，並與強度做比較，來決定該畫素要填入白色或黑色，抑或是保持原樣，來得到新的畫素成為噪音。

Box filter，將該畫素與周圍畫素取其算術平均數，成為該點的新畫素。

Median filter，將該畫素與周圍畫素取其中位數，成為該點的新畫素。

OpeningClosing filter，依序使用 Erosion，Dilation，Dilation，Erosion 來對圖片做 4 次處理。

ClosingOpening filter，依序使用 Dilation，Erosion，Erosion，Dilation 來對圖片做 4 次處理。

PrincipalCode：

◎Gaussian_Noise

```
for(i=0; i<bmpInfo.biHeight; ++i)
    for(j=0; j<bmpInfo.biWidth; ++j)
        n = sqrt(-2 * log((double)rand() / RAND_MAX )) * cos(2 * M_PI * ((double)rand() /
        RAND_MAX ));
        BMPoutput_data[i][j].color = BMPdata[i][j].color + amplitude * n;
```

◎Salt_Pepper_Noise

```
for(i=0; i<bmpInfo.biHeight; ++i)
    for(j=0; j<bmpInfo.biWidth; ++j)
        n = (double)rand() / RAND_MAX;
        if(fabs(n) < probability)
            BMPoutput_data[i][j].color = 0;
        else if(1 < fabs(n) + probability)
            BMPoutput_data[i][j].color = 255;
        else
            BMPoutput_data[i][j].color = BMPdata[i][j].color;
```

◎Box

```
//將圖片外框填入 0 畫素
for(i=0; i<bmpInfo.biHeight; ++i)
    for(j=0; j<bmpInfo.biWidth; ++j)
        tmp[i+border][j+border] = BMPoutput_data[i][j].color;

//與周圍畫素取算數平均數
for(i=border; i<bmpInfo.biHeight+border; ++i)
    for(j=border; j<bmpInfo.biWidth+border; ++j)
        total = 0;
        for(k=0-border; k<=border; ++k)
            for(l=0-border; l<=border; ++l)
                total += tmp[i+k][j+l];
        BMPoutput2_data[i-border][j-border].color = total / (filter * filter);
```

◎Median

```
//將圖片外框填入 0 畫素
for(i=0; i<bmpInfo.biHeight; ++i)
    for(j=0; j<bmpInfo.biWidth; ++j)
        tmp[i+border][j+border] = BMPoutput_data[i][j].color;

//與周圍畫素取中位數
for(i=border; i<bmpInfo.biHeight+border; ++i)
    for(j=border; j<bmpInfo.biWidth+border; ++j)
        total = 0;
        for(k=0-border; k<=border; ++k)
            for(l=0-border; l<=border; ++l)
                sorted_data[total++] = tmp[i+k][j+l];
```

```
        total = filter * filter;
        nth_element(sorted_data, sorted_data + (total >> 1), sorted_data + total);
        BMPoutput2_data[i-border][j-border].color = sorted_data[total>>1];
```

◎Opening_Closing

```
erosion(BMPoutput_data, BMPoutput2_data);
dilation(BMPoutput2_data, BMPoutput3_data);
dilation(BMPoutput3_data, BMPoutput2_data);
erosion(BMPoutput2_data, BMPoutput3_data);
```

◎Closing_Opening

```
dilation(BMPoutput3_data, BMPoutput2_data);
erosion(BMPoutput2_data, BMPoutput3_data);
erosion(BMPoutput_data, BMPoutput2_data);
dilation(BMPoutput2_data, BMPoutput3_data);
```

◎SNR

```
for(i=0; i<bmpInfo.biHeight; ++i)
    for(j=0; j<bmpInfo.biWidth; ++j)
        u += BMPdata[i][j].color;
        uN += compareimg[i][j].color - BMPdata[i][j].color;
u /= n;
uN /= n;

for(i=0; i<bmpInfo.biHeight; ++i)
    for(j=0; j<bmpInfo.biWidth; ++j)
        VS += ((double)BMPdata[i][j].color - u) * ((double)BMPdata[i][j].color - u);
        VN += ((double)compareimg[i][j].color - (double)BMPdata[i][j].color - uN) *
        ((double)compareimg[i][j].color - (double)BMPdata[i][j].color - uN);

VS /= n;
VN /= n;
return 20 * log10(sqrt(VS / VN));
```

Parameters：

編譯程式碼 g++ -o lena lena.cpp

執行程式./lena lena.bmp

lena.bmp 是我們的 Input Image

ResultingImages：

gaussian_noise_10

gaussian_noise_30

salt_pepper_noise_0.05

salt_pepper_noise_0.1

gaussian_noise_10_box3x3

gaussian_noise_10_box5x5

gaussian_noise_10_median3x3

gaussian_noise_10_median5x5

gaussian_noise_10_opening_closing

gaussian_noise_10_closing_opening

gaussian_noise_30_box3x3

gaussian_noise_30_box5x5

aussian_noise_30_median3x3

gaussian_noise_30_median5x5

gaussian_noise_10_opening_closing

gaussian_noise_10_closing_opening

salt_pepper_noise_0.05_box3x3

salt_pepper_noise_0.05_box5x5

salt_pepper_noise_0.05_median3x3

salt_pepper_noise_0.05_median5x5

salt_pepper_noise_0.05_opening_closing

salt_pepper_noise_0.05_closing_opening

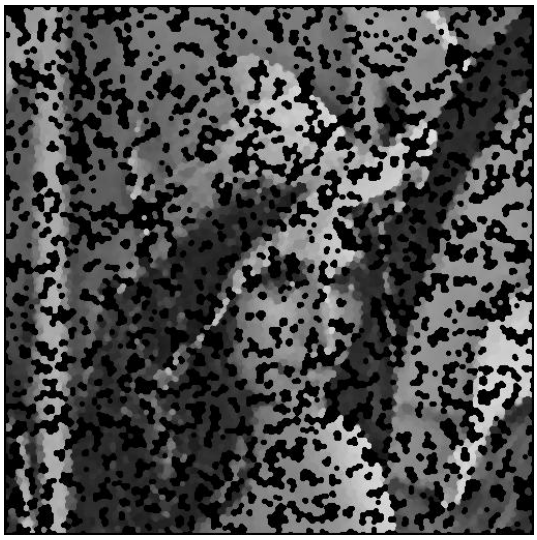
salt_pepper_noise_0.1_box3x3

salt_pepper_noise_0.1_box5x5

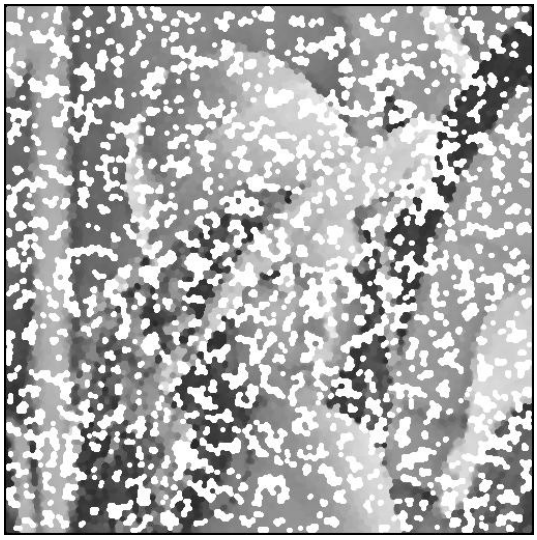
salt_pepper_noise_0.1_median3x3

salt_pepper_noise_0.1_median5x5

salt_pepper_noise_0.1_opening_closing



salt_pepper_noise_0.1_closing_opening



gaussian_noise_10：SNR = 13.5423
gaussian_noise_10_box3x3：SNR = 16.3708
gaussian_noise_10_box5x5：SNR = 13.6609
gaussian_noise_10_median3x3：SNR = 17.5706
gaussian_noise_10_median5x5：SNR = 15.8511
gaussian_noise_10_opening_closing：SNR = 8.6121
gaussian_noise_10_closing_opening：SNR = 7.6509

gaussian_noise_30：SNR = 2.1780
gaussian_noise_30_box3x3：SNR = 9.6511
gaussian_noise_30_box5x5：SNR = 10.3105
gaussian_noise_30_median3x3：SNR = 10.6531
gaussian_noise_30_median5x5：SNR = 12.1972
gaussian_noise_30_opening_closing：SNR = 6.4853
gaussian_noise_30_closing_opening：SNR = 4.5742

salt_pepper_noise_0.05：SNR = 0.8402
salt_pepper_noise_0.05_box3x3：SNR = 9.1876
salt_pepper_noise_0.05_box5x5：SNR = 10.5374
salt_pepper_noise_0.05_median3x3：SNR = 18.5808
salt_pepper_noise_0.05_median5x5：SNR = 15.9095
salt_pepper_noise_0.05_opening_closing：SNR = 4.4299
salt_pepper_noise_0.05_closing_opening：SNR = 3.5768

salt_pepper_noise_0.1：SNR = -2.1065
salt_pepper_noise_0.1_box3x3：SNR = 6.2475
salt_pepper_noise_0.1_box5x5：SNR = 8.1919
salt_pepper_noise_0.1_median3x3：SNR = 14.4747
salt_pepper_noise_0.1_median5x5：SNR = 14.3446
salt_pepper_noise_0.1_opening_closing：SNR = -2.3627
salt_pepper_noise_0.1_closing_opening：SNR = -2.9652