2021/12/13 陳嘉政 r10922171 HW9

1. Roberts operator(Threshold: 30)

主要程式碼:

(1)函式輸入分別為:照 hw8 方法擴張一次的圖、原圖及閥值。先宣告一個 result 矩陣，將原圖複製到 result，初始化兩個 mask 係數矩陣，將擴張圖 row by row，col by col，從(1,1)開始到(rows-1, col-1)，每個點作為原點圈出 2x2 矩形，分別與兩個 mask 求值(matrixSum 函式)，求出的兩個值平方後相加，然後加總再開根號看有沒有>=閥值，若有，則 result 圖中點(i - 1, j – 1)視為邊(value = 0)，否則 value = 255。

```
Mat Roberts_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[2];
    mask[0] = (Mat_<double>(2, 2) << //r1
        -1, 0,
        0, 1);
    mask[1] = (Mat_<double>(2, 2) << //r2
        0, -1,
        1, 0);

    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            Mat kernel = padded(cv::Rect(j, i, 2, 2));
            double value = 0;
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                double temp = matrixSum(kernel, mask[n]);
                value += pow(temp, 2);
            }
            if (sqrt(value) >= threshold)
                result.at<uint8_t>(i - 1, j - 1) = 0;
            else
                result.at<uint8_t>(i - 1, j - 1) = 255;
        }
    }
    return result;
}
```

matrixSum 的作用是將圈出的矩形中每個點與 mask 對應的係數相乘再加總。

```
double matrixSum(Mat kernel, Mat mask)
{
    double total = 0;
    for (int i = 0; i < kernel.rows; i++) {
        for (int j = 0; j < kernel.cols; j++) {
            total += kernel.at<uint8_t>(i, j) * mask.at<double>(i, j);
        }
    }
    return total;
}
```

結果:

此閥值產生的結果比閥值為 12 產生的結果的邊還要細，可以推得閥值越高邊會越細

2. Prewitt operator(Threshold: 24)

主要程式碼:

除了點(I , j)圈出的矩形原點為(j – 1, i – 1)，大小為 3x3，以及 mask 矩陣的大小及係數改變，其他做法都跟 1.相似。

```
Mat Prewitt_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[2];
    mask[0] = (Mat_<double>(3, 3) << //p1
        -1, -1, -1,
         0,  0,  0,
         1,  1,  1);
    mask[1] = (Mat_<double>(3, 3) << //p2
        -1, 0, 1,
        -1, 0, 1,
        -1, 0, 1);

    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            Mat kernel = padded(cv::Rect(j - 1, i - 1, 3, 3));
            double value = 0;
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                double temp = matrixSum(kernel, mask[n]);
                value += pow(temp, 2);
            }
            if (sqrt(value) >= threshold)
                result.at<uint8_t>(i - 1, j - 1) = 0;
            else
                result.at<uint8_t>(i - 1, j - 1) = 255;
        }
    }
    return result;
}
```

結果:

3. Sobel operator(Threshold: 38)

主要程式碼:

除了點(I , j)圈出的矩形原點為(j – 1, i – 1)，大小為 3x3，以及 mask 矩陣的大小及係數改變，其他做法都跟 1.相似

```
Mat Sobel_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[2];
    mask[0] = (Mat_<double>(3, 3) << //s1
        -1, -2, -1,
         0,  0,  0,
         1,  2,  1);
    mask[1] = (Mat_<double>(3, 3) << //s2
        -1, 0, 1,
        -2, 0, 2,
        -1, 0, 1);

    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            Mat kernel = padded(cv::Rect(j - 1, i - 1, 3, 3));
            double value = 0;
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                double temp = matrixSum(kernel, mask[n]);
                value += pow(temp, 2);
            }
            if (sqrt(value) >= threshold)
                result.at<uint8_t>(i - 1, j - 1) = 0;
            else
                result.at<uint8_t>(i - 1, j - 1) = 255;
        }
    }
    return result;
}
```

結果:

4. Frei and Chen operator(Threshold: 30)

主要程式碼:

除了點(I , j)圈出的矩形原點為(j – 1, i – 1)，大小為 3x3，以及 mask 矩陣的大小及係數改變，其他做法都跟 1.相似

```
Mat Frei_and_Chen_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[2];
    mask[0] = (Mat_<double>(3, 3) << //f1
        -1, -sqrt(2), -1,
        0,         0,  0,
        1,   sqrt(2),  1);
    mask[1] = (Mat_<double>(3, 3) << //f2
             -1, 0,       1,
        -sqrt(2), 0, sqrt(2),
             -1, 0,       1);

    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            Mat kernel = padded(cv::Rect(j - 1, i - 1, 3, 3));
            double value = 0;
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                double temp = matrixSum(kernel, mask[n]);
                value += pow(temp, 2);
            }
            if (sqrt(value) >= threshold)
                result.at<uint8_t>(i - 1, j - 1) = 0;
            else
                result.at<uint8_t>(i - 1, j - 1) = 255;
        }
    }
    return result;
}
```

結果:

5. Kirsch compass operator(Threshold: 135)

主要程式碼:

Mask 係數矩陣有 8 個，宣告長度為 8 的 value 陣列，分別將每個係數矩陣與擴張圖圈出的 3x3 矩形計算出的 matrixSum 存入 value 陣列。呼叫 findMax 函式將 array 由小到大排列並回傳最大值。若此值>=閥值，則 result 中此點 value = 0，否則 value = 255。

```
Mat Kirsch_Compass_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[8];
    mask[0] = (Mat_<double>(3, 3) << //k0
         -3, -3, 5,
         -3,  0, 5,
         -3, -3, 5);
    mask[1] = (Mat_<double>(3, 3) << //k1
         -3,  5,  5,
         -3,  0,  5,
         -3, -3, -3);
    mask[2] = (Mat_<double>(3, 3) << //k2
          5,  5,  5,
         -3,  0, -3,
         -3, -3, -3);
    mask[3] = (Mat_<double>(3, 3) << //k3
          5,  5, -3,
          5,  0, -3,
         -3, -3, -3);
    mask[4] = (Mat_<double>(3, 3) << //k4
          5, -3, -3,
          5,  0, -3,
          5, -3, -3);
    mask[5] = (Mat_<double>(3, 3) << //k5
         -3, -3, -3,
          5,  0, -3,
          5,  5, -3);
```

```
    mask[6] = (Mat_<double>(3, 3) << //k6
        -3, -3, -3,
        -3,  0, -3,
         5,  5,  5);
    mask[7] = (Mat_<double>(3, 3) << //k7
        -3, -3, -3,
        -3,  0,  5,
        -3,  5,  5);

    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            Mat kernel = padded(cv::Rect(j - 1, i - 1, 3, 3));
            double value[8] = { 0 };
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                value[n] = matrixSum(kernel, mask[n]);
            }
            if (findMax(value, 8) >= threshold)
                result.at<uint8_t>(i - 1, j - 1) = 0;
            else
                result.at<uint8_t>(i - 1, j - 1) = 255;
        }
    }
    return result;
}
```

findMax 函式就是在做 bubble sort 並回傳最後一個元素(最大值)

```
double findMax(double array[], int arraysize)
{
    for (int i = arraysize - 2; i >= 0; i--) {
        for (int j = 0; j <= i; j++) {
            if (array[j] > array[j + 1]) {
                double temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    return array[arraysize - 1];
}
```

結果:

6. Robinson compass operator(Threshold: 43)

主要程式碼:

做法與 5.相同，差在 mask 矩陣係數改變

```
Mat Robinson_compass_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[8];
    mask[0] = (Mat_<double>(3, 3) << //r0
        -1, 0, 1,
        -2, 0, 2,
        -1, 0, 1);
    mask[1] = (Mat_<double>(3, 3) << //r1
         0, 1, 2,
        -1, 0, 1,
        -2, -1, 0);
    mask[2] = (Mat_<double>(3, 3) << //r2
         1, 2, 1,
         0, 0, 0,
        -1, -2, -1);
    mask[3] = (Mat_<double>(3, 3) << //r3
         2, 1, 0,
         1, 0, -1,
         0, -1, -2);
    mask[4] = -mask[0]; //r4
    mask[5] = -mask[1]; //r5
    mask[6] = -mask[2]; //r6
    mask[7] = -mask[3]; //r7

    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            Mat kernel = padded(cv::Rect(j - 1, i - 1, 3, 3));
            double value[8] = { 0 };
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                value[n] = matrixSum(kernel, mask[n]);
            }
            if (findMax(value, 8) >= threshold)
                result.at<uint8_t>(i - 1, j - 1) = 0;
            else
                result.at<uint8_t>(i - 1, j - 1) = 255;
        }
    }
    return result;
}
```

結果:

7. Nevatia Babu operator(Threshold: 12500)

主要程式碼:

將原圖擴張 2 次作為擴張圖，mask 矩陣有 6 個，大小為 5x5，宣告長度為 6 的 value 陣列。Row by row, col by col，從擴張圖中(2,2)開始到(rows – 2, cols – 2)，每個點(i , j)都以(j – 2, i – 2)為原點，圈出 5x5 大小的矩形，將矩形分別與每個 mask 算出 matrixSum，存入 value 陣列，之後呼叫 findMax，回傳 value 陣列最大值，將此值與閥值比較，若為 >=，原圖點(i – 2, j – 2) 設為 0，否則設為 255。

```
Mat Nevatia_Babu_operator(Mat padded, Mat orig, int threshold)
{
    Mat result;
    orig.copyTo(result);

    Mat mask[6];
    mask[0] = (Mat_<double>(5, 5) << //0度
         100,  100,  100,  100,  100,
         100,  100,  100,  100,  100,
           0,    0,    0,    0,    0,
        -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100);
    mask[1] = (Mat_<double>(5, 5) << //30度
         100,  100,  100,  100,  100,
         100,  100,  100,   78,  -32,
         100,   92,    0,  -92, -100,
          32,  -78, -100, -100, -100,
        -100, -100, -100, -100, -100);
    mask[2] = (Mat_<double>(5, 5) << //60度
        100, 100,  100,   32, -100,
        100, 100,   92,  -78, -100,
        100, 100,    0, -100, -100,
        100,  78,  -92, -100, -100,
        100, -32, -100, -100, -100);
    mask[3] = (Mat_<double>(5, 5) << //-90度
        -100, -100, 0, 100, 100,
        -100, -100, 0, 100, 100,
        -100, -100, 0, 100, 100,
        -100, -100, 0, 100, 100,
        -100, -100, 0, 100, 100);
    mask[4] = (Mat_<double>(5, 5) << //-60度
        -100,   32,  100, 100, 100,
        -100,  -78,   92, 100, 100,
        -100, -100,    0, 100, 100,
        -100, -100,  -92,  78, 100,
        -100, -100, -100, -32, 100);
```

```
    mask[5] = (Mat_<double>(5, 5) << //-30度
        100,  100,  100,  100,  100,
        -32,   78,  100,  100,  100,
       -100,  -92,    0,   92,  100,
       -100, -100, -100,  -78,   32,
       -100, -100, -100, -100, -100);

    for (int i = 2; i < padded.rows - 2; i++) {
        for (int j = 2; j < padded.cols - 2; j++) {
            Mat kernel = padded(cv::Rect(j - 2, i - 2, 5, 5));
            double value[6] = { 0 };
            for (int n = 0; n < sizeof(mask) / sizeof(*mask); n++) {
                value[n] = matrixSum(kernel, mask[n]);
            }
            if (findMax(value, 6) >= threshold)
                result.at<uint8_t>(i - 2, j - 2) = 0;
            else
                result.at<uint8_t>(i - 2, j - 2) = 255;
        }
    }
    return result;
}
```

結果: