

# Matrix 大作业报告

计算机 2 班

2252941 杨瑞灵

完成日期：2023 年 10 月 17 日

# 目录

<b>1 设计思路与功能描述</b>	<b>3</b>
1.1 设计思路 . . . . .	3
1.2 功能描述 . . . . .	6
1.2.1 矩阵加法 . . . . .	7
1.2.2 矩阵数乘 . . . . .	7
1.2.3 矩阵转置 . . . . .	8
1.2.4 矩阵乘法 . . . . .	8
1.2.5 Hadamard 乘积 . . . . .	9
1.2.6 矩阵卷积 . . . . .	9
1.2.7 卷积应用 . . . . .	10
1.2.8 OTSU 算法 . . . . .	10
1.2.9 其他加分项 . . . . .	11
1.2.10 your picture . . . . .	12
<b>2 问题及解决方法</b>	<b>13</b>
2.1 问题一：图片斑驳??? . . . . .	13
<b>3 心得体会</b>	<b>16</b>
3.1 10月10日 . . . . .	16
<b>4 源代码</b>	<b>17</b>
4.1 Matrix.h . . . . .	17
4.2 main.app . . . . .	17
4.3 menu.app . . . . .	21
4.4 normal_operate.app . . . . .	22
4.5 opencv_operate.app . . . . .	35

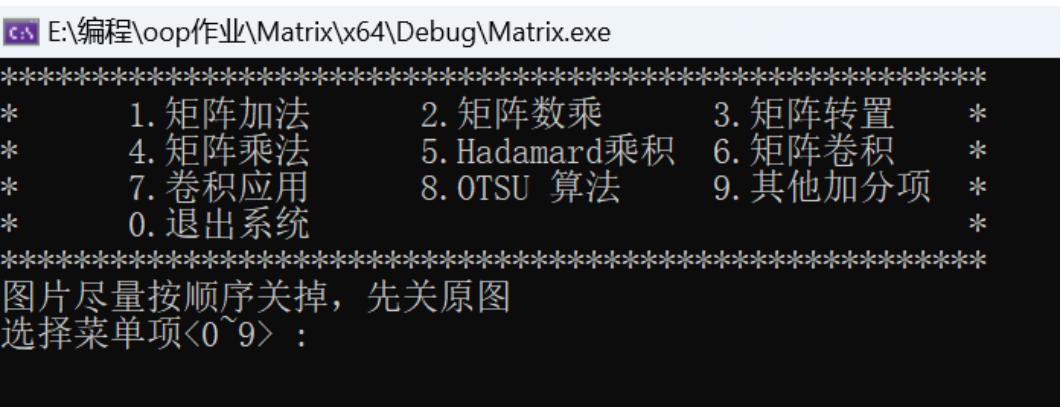
# 1 设计思路与功能描述

## 1.1 设计思路

**第一步：**分成多个源程序：

main: 实现调用各个功能  
menu: 实现菜单页面的设计  
normal\_operate: 包含矩阵运算函数  
opencv\_operate: 包含 opencv 图像处理函数

**第二步：**整体框架：由于题目已经给出了整体框架，所以只需要稍作修改，先把各个函数建立，不必写内容，再搞好头文件和常量就大功告成。



```
E:\编程\oop作业\Matrix\x64\Debug\Matrix.exe
*****
*      1. 矩阵加法      2. 矩阵数乘      3. 矩阵转置      *
*      4. 矩阵乘法      5. Hadamard 乘积  6. 矩阵卷积      *
*      7. 卷积应用      8. OTSU 算法      9. 其他加分项  *
*      0. 退出系统          *                  *                  *
*****
图片尽量按顺序关掉，先关原图
选择菜单项<0~9> :
```

图 1: menu

**第三步：**然后是填充函数，前六个都很简单，建立 struct MAREIX，用 int 型指针申请空间来存放一维数组。

**第四步：**功能七：

1. Mat image = imread("images/demolena.jpg", IMREAD\_GRAYSCALE);  
图像生成 MAT 对象
2. 循环六次，每次建立 Mat image\_copy = image.clone() 克隆原图像，用 int \*image\_matrix 复刻 image 的矩阵，然后调用卷积函数处理，再返回 image\_copy，显示图像

**第五步：**功能八：掉用 threshold 进行二值化，注意 type: 阈值类型，应该

调用 THRESH\_BINARY

Enumerator	
THRESH_BINARY	$\text{dst}(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV	$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC	$\text{dst}(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$
THRESH_TOZERO	$\text{dst}(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV	$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$
THRESH_MASK	
THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

图 2: threshold

第六步：功能九：

1. 读图像，进行二值化生成 binary\_img
2. 使用形态学闭运算来填充 binary\_img 内部的空洞，去除黑点
3. 使用形态学开运算来去除 binary\_img 外部的噪声，去除白斑
4. 创建一个新的图像，将 binary\_img 为 0 的区域设置为 0（黑色）
5. 显示原图和处理后的图

```
1 | cv2.morphologyEx(img, op, kernel)
```

参数op的取值	含义
cv2.MORPH_OPEN	开运算(open),先腐蚀后膨胀的过程。开运算可以用来消除小黑点，在纤细点处分离物体、平滑较大物体的边界的同时并不明显改变其面积。
cv2.MORPH_CLOSE	闭运算(close), 先膨胀后腐蚀的过程。闭运算可以用来排除小黑洞。
cv2.MORPH_GRADIENT	形态学梯度(morph-grad), 可以突出团块(blob)的边缘, 保留物体的边缘轮廓。
cv2.MORPH_TOPHAT	顶帽(top-hat), 将突出比原轮廓亮的部分。
cv2.MORPH_BLACKHAT	黑帽(black-hat), 将突出比原轮廓暗的部分。

图 3: morphologyEx 形态学变化函数

### 第七步：功能十：小小整活

1. VideoCapture cap(0); 获取摄像机画面
2. 每次循环延迟 25ms 获取画面，然后显示图片达到连续效果
3. Canny 函数边缘检测
4. createTrackbar 每次滑动进度条调用一次函数重新 Canny

### Steps

1. Filter out any noise. The Gaussian filter is used for this purpose. An example of a Gaussian kernel of *size* = 5 that might be used is shown below:

$$K = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

- a. Apply a pair of convolution masks (in *x* and *y* directions):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

- b. Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

3. *Non-maximum suppression* is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.

4. *Hysteresis*: The final step. Canny does use two thresholds (upper and lower):

- a. If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge
- b. If a pixel gradient value is below the *lower* threshold, then it is rejected.
- c. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

Canny recommended a *upper/lower* ratio between 2:1 and 3:1.

图 4: Canny 边缘检测

## 1.2 功能描述

直接上图

### 1.2.1 矩阵加法

```
1. 矩阵加法:  
请输入矩阵的行和列，中间用空格隔开:  
2 3  
请输入矩阵的项，中间用空格隔开:  
2 4 45  
2 342 4  
请输入矩阵的行和列，中间用空格隔开:  
2 3  
请输入矩阵的项，中间用空格隔开:  
2 34 1  
234 12 3  
运算结果是:  
4 38 46  
236 354 7  
按回车键继续...
```

图 5: 矩阵加法

### 1.2.2 矩阵数乘

```
2. 矩阵数乘:  
请输入矩阵的行和列，中间用空格隔开:  
2 3  
请输入矩阵的项，中间用空格隔开:  
23 34 3  
234 34 2  
请输入一个整数:  
34  
运算结果是:  
782 1156 102  
7956 1156 68  
按回车键继续...
```

图 6: 矩阵数乘

### 1.2.3 矩阵转置

```
3. 矩阵转置:  
请输入矩阵的行和列，中间用空格隔开:  
4 2  
请输入矩阵的项，中间用空格隔开:  
2 3  
21 2  
12 3  
12 4  
运算结果是:  
2 21 12 12  
3 2 3 4  
  
按回车键继续...
```

图 7: 矩阵转置

### 1.2.4 矩阵乘法

```
4. 矩阵乘法:  
请输入矩阵的行和列，中间用空格隔开:  
2 3  
请输入矩阵的项，中间用空格隔开:  
324 34 2  
23 3 5  
请输入矩阵的行和列，中间用空格隔开:  
3 5  
请输入矩阵的项，中间用空格隔开:  
2 3 34 2 3  
2 5 6 3 2  
23 3 5 34 4  
运算结果是:  
762 1148 11230 818 1048  
167 99 825 225 95  
  
按回车键继续...
```

图 8: 矩阵乘法

### 1.2.5 Hadamard 乘积

```
5. Hadamard乘积  
请输入矩阵的行和列，中间用空格隔开:  
3 1  
请输入矩阵的项，中间用空格隔开:  
3  
2  
3  
请输入矩阵的行和列，中间用空格隔开:  
3 1  
请输入矩阵的项，中间用空格隔开:  
23  
3  
2  
运算结果是:  
69  
6  
6  
按回车键继续...
```

图 9: Hadamard 乘积

### 1.2.6 矩阵卷积

```
6. 矩阵卷积: kernel size = 3, padding = 1, stride = 1, dilation = 1;  
请输入方阵的阶数:  
3  
请输入矩阵的项，中间用空格隔开:  
3 4 2  
3 2 4  
6 3 2  
kernel矩阵的阶数为: 3  
请输入矩阵的项，中间用空格隔开:  
2 4 2  
2 4 2  
3 4 5  
运算结果是:  
42 63 38  
75 88 53  
46 50 34  
按回车键继续...
```

图 10: 矩阵卷积

### 1.2.7 卷积应用

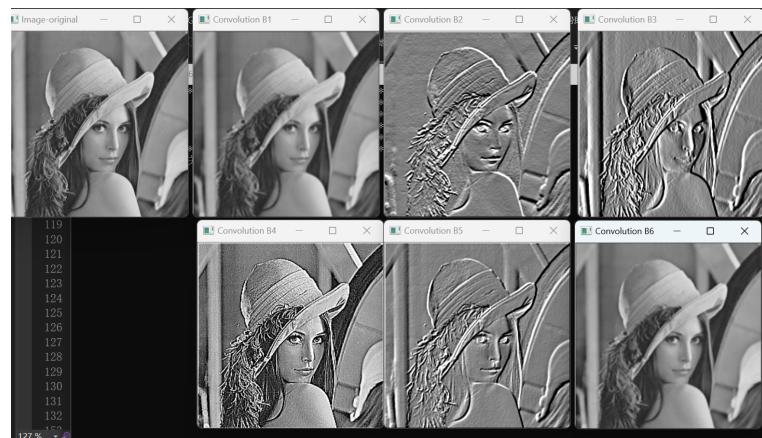


图 11: 卷积应用

### 1.2.8 OTSU 算法

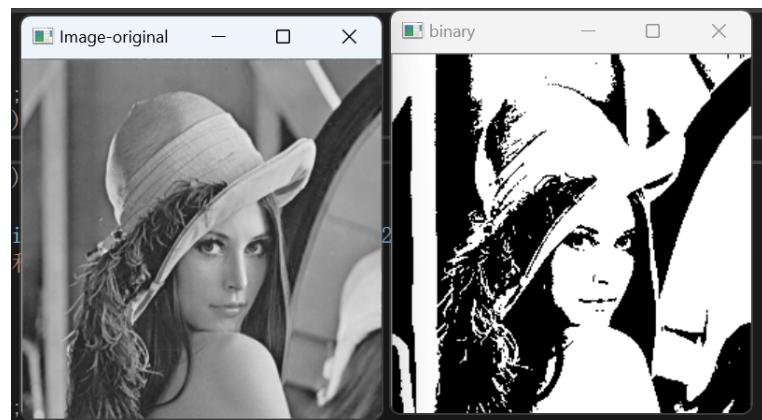


图 12: OTSU 算法

### 1.2.9 其他加分项

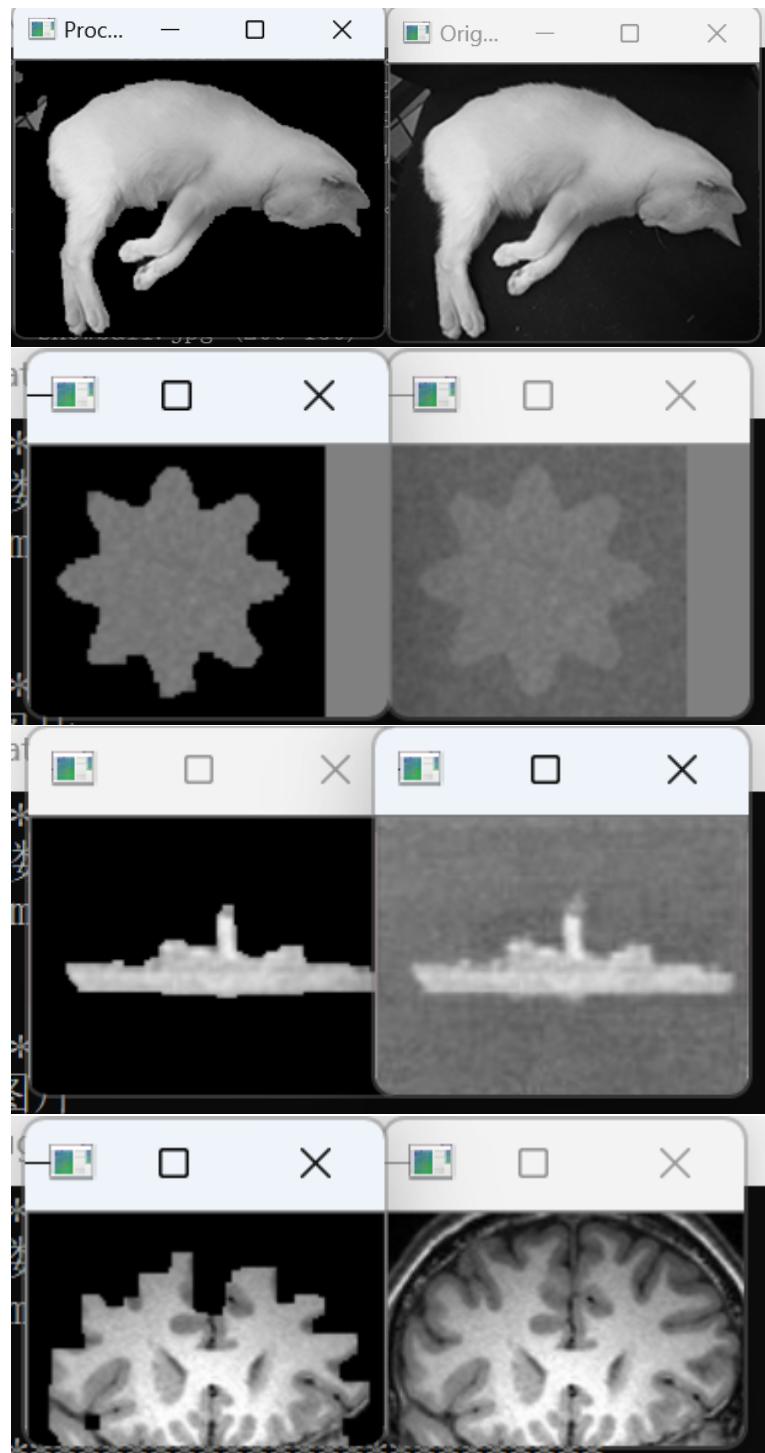


图 13: 其他加分项

### 1.2.10 your picture

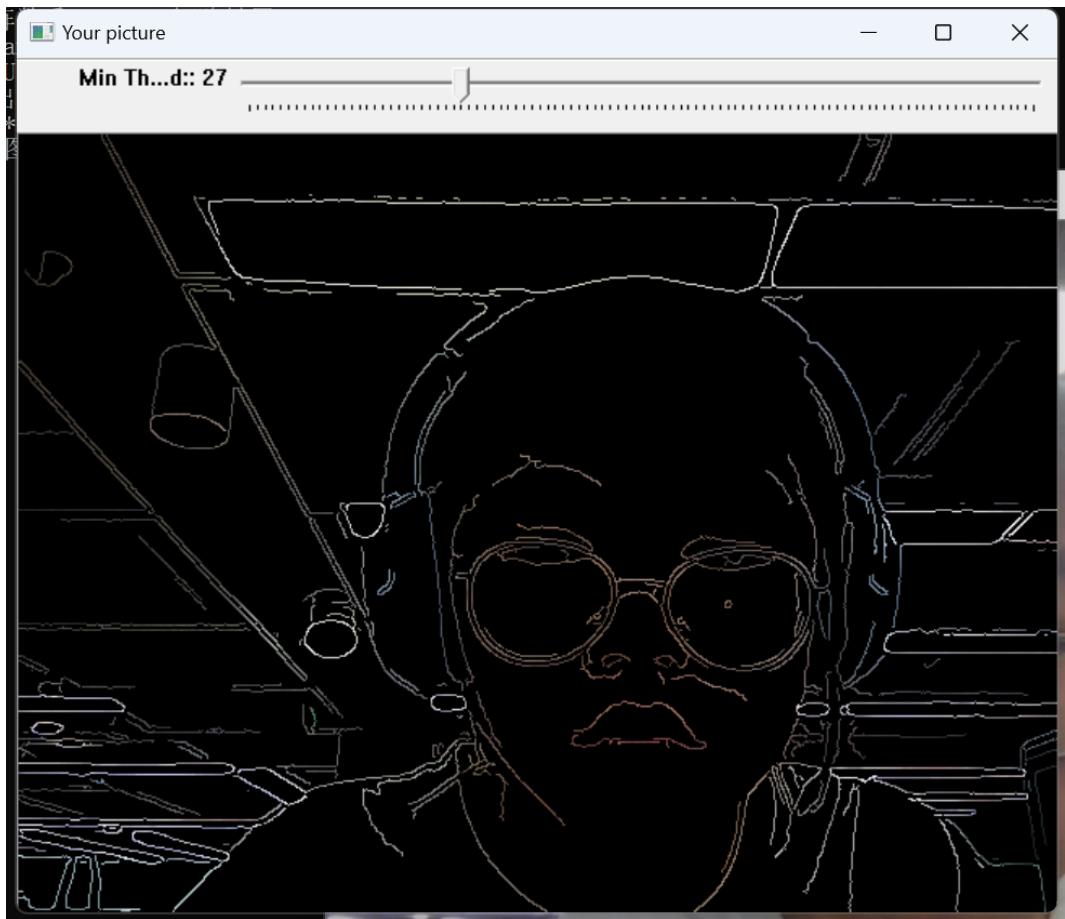


图 14: my picture

## 2 问题及解决方法

### 2.1 问题一：图片斑驳???

**问题描述：**问题七：最开始我用 opencv 自带函数进行卷积，但是飞姐说不行，我就只好重新写函数，手搓卷积，于是。。。

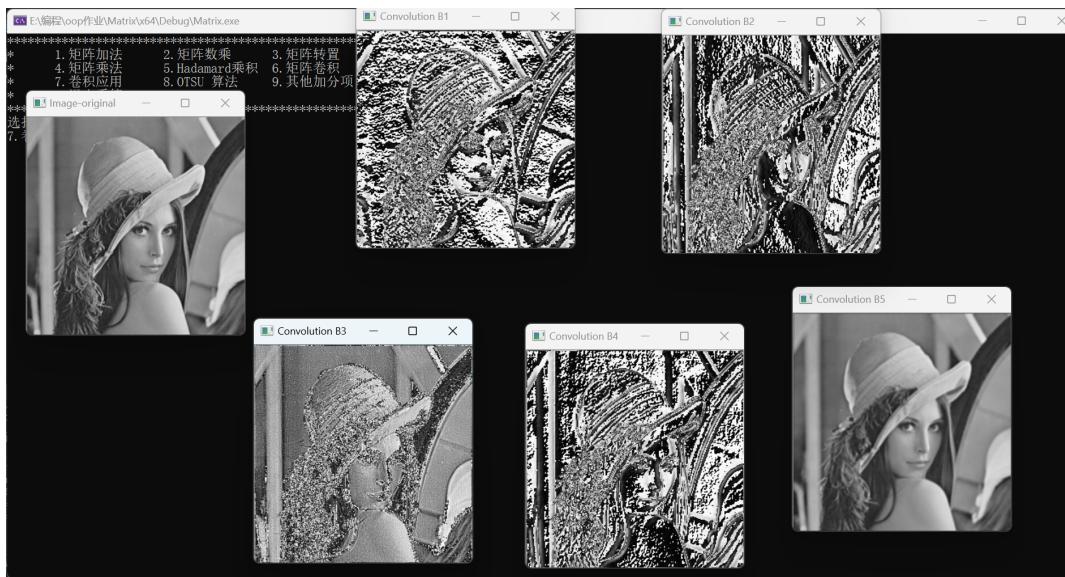


图 15: 手搓—错误示范

**问题解答：**由于没有看到有参考资料，所以我非常悲惨的，一个一个输出像素值，看是哪里的问题。

```

486 646 643 646 646 648 641 633 627 629 632 624 619 618 616 617 617 619 6
407 413 419 427 430 425 427 433 430 428 432 434 435 431 423 426 440 4
515 515 518 518 515 518 528 530 524 525 529 530 531 533 533 533 534 5
519 520 522 520 522 531 534 531 530 527 523 527 534 534 535 535 536 530 5
518 517 513 510 510 511 513 507 495 488 484 477 463 448 431 418 431 4
629 626 617 613 611 611 611 608 608 613 612 609 616 624 623 620 627 6
459 471 475 485 487 480 479 481 490 495 485 480 485 487 485 485 488 4
589 666 663 459
-1 -6 -6 -6 -8 -8 -5 -5 -8 -6 -3 0 0 0 3 -1 -7 -6 -2 1 0 -1 2 3 0
1 -2 0 4 3 2 0 0 0 -4 -3 -3 -6 -3 0 0 -2 -5 -5 -1 1 -1 -1 0 -1 -1 1 1
1 -2 -6 -7 -4 -3 -3 -2 -3 -3 -4 -6 -5 -3 -1 0 -3 -6 -6 -8 -8 -5 -4 -5
1 1 -3 -5 -3 0 2 0 -1 -1 -1 -1 -4 -5 -5 -5 -2 -2 -5 -4 0 1 -1 -1 1 3
1 -3 -6 -3 -1 -3 -4 -3 -4 -8 -10 -9 -5 -1 0 1 2 3 9 19 20 8 -4 -6 -1
1 -3 -3 -2 -2 -1 -2 -5 -5 -3 -1 -1 -2 -2 -3 0 8 6 -11 -36 -67 -67

```

图 16: 漫长的修改之路 1

原来是像素值超过了 0-255，我加上判断，把 0-255 之外的像素改成 0 和 255，就变成了。。。

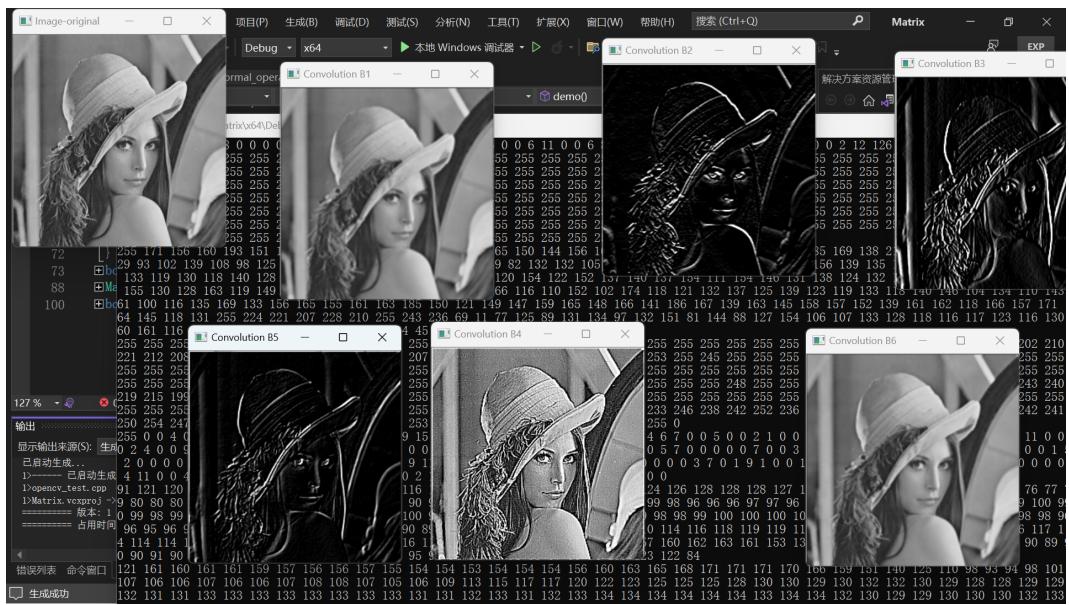


图 17: 漫长的修改之路 2

oh, Lena 我的 Lena 怎么这么黑（哭），原来是因为没有加上 128 的修正。。。

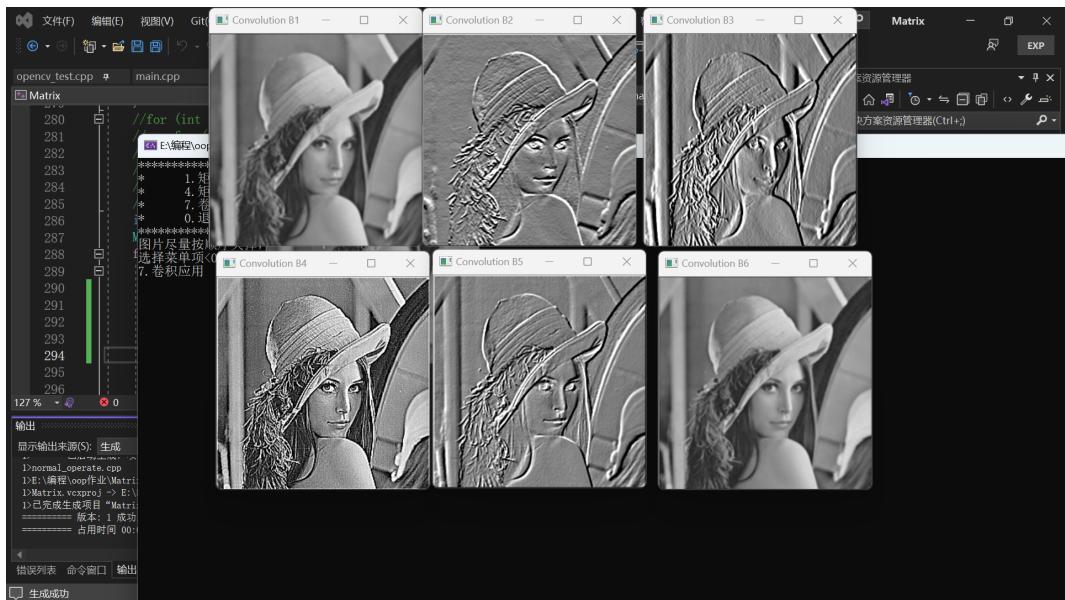


图 18: 漫长的修改之路 3

好我的 Lena 大美女出现了，泪目 www

### 3 心得体会

#### 3.1 10月10日

以下是我10月10日的心得体会，由于还没有做页面的设计，不确定后面会不会改，但是先把报告写了吧 quq

现在是正文：半天写完了所有内容，还是挺顺利的，就是改问题一花了很多时间。

opencv 在 robotmaster 视觉组培训里已经有了初步了解，所以写起来倒不怎么难，很多东西和识别灯柱是一样的。不过还是学到了很多新知识，比如形态学处理函数，二值化函数的其他参数，以及 Mat 和矩阵相互转化，还有建立一个 mask 来实现 setTo

```
1 | src.setTo(value, mask);
```

当默认不添加 `mask` 的时候，表明 `mask` 是一个与原图尺寸大小一致的且元素值全为非0的矩阵，因此不加 `mask` 的时候，会将原矩阵的像素值全部赋值为 `value`；当带有 `mask` 这个参数的时候，该函数会把矩阵 `mask` 中元素不为0的点全部变为 `value` 值（下面详述该函数带参形式的用法）。

首先，`mask` 是一个和 `src` 同 size 的矩阵，该函数的功能，是针对图像矩阵 `src`，在矩阵 `src` 中，把矩阵 `mask` 中元素 不为0 的相应像素点位置的像素值全部变为 `value` 值<sup>1</sup>；也即对于 `mask` 中非零的位置，在 `src` 中的对应位置的元素会被置为 `value` 值，其他部分不变<sup>2</sup>。

图 19: setTo

## 4 源代码

### 4.1 Matrix.h

```
1 #pragma once
2
3 #define ERROR 0
4
5 int menu();
6
7
8 bool matriplus();
9
10 bool nummult();
11
12 bool matritrans();
13
14 bool matrimulti();
15
16 bool hadamulti();
17
18 bool conv();
19
20 bool conv_eg(int*& image_matrix, int choice);
21
22
23 bool demo();
24
25 bool OTSU();
26
27 bool plus_code(const char* str, int closed_size = 5,
28                 int open_size = 3);
29
30 bool free_act();
```

### 4.2 main.app

```
1 #include <conio.h>
2
3 #include <iostream>
4
5 #include "Matrix.h"
6
7 using namespace std;
```

```
// 此框架若有不完美可以在作业中任意修改  
  
void wait_for_enter()  
{  
    cout << endl  
        << "按回车键继续";  
    while (_getch() != '\r')  
        ;  
    cout << endl  
        << endl;  
}  
  
int main()  
{  
    int choice = 0;  
  
    wait_for_enter();  
    while (true) // 注意该循环退出的条件  
    {  
        system("cls"); // 清屏函数  
  
        choice=menu(); //  
            调用菜单显示函数，自行补充完成  
  
        // 按要求输入菜单选择项choice  
  
        if (choice == '0') // 选择退出  
        {  
            char ch;
```

```

34         cout << "\n确定退出吗?(输入 Y 或 N):" <<
35             endl;
36         cin >> ch;
37         if (ch == 'y' || ch == 'Y')
38             break;
39         else
40             continue;
41     }
42
43     switch (choice)
44     {
45         // 下述矩阵操作函数自行设计并完成(包括函数参数及返回类型等)
46         case '1':
47             cout << "\n1. 矩阵加法: " << endl;
48             matriplus();
49             break;
50         case '2':
51             cout << "\n2. 矩阵数乘: " << endl;
52             nummulti();
53             break;
54         case '3':
55             cout << "\n3. 矩阵转置: " << endl;
56             matritrans();
57             break;
58         case '4':
59             cout << "\n4. 矩阵乘法: " << endl;
60             matrimulti();
61             break;

```

```
61         case '5':
62             cout << "\n5. Hadamard 乘积" << endl;
63             hadamulti();
64             break;
65         case '6':
66             cout << "\n6. 矩阵卷积: kernel size =
67                         3, padding = 1, stride =
68                         1, dilation = 1; " << endl;
69             conv();
70             break;
71         case '7':
72             cout << "\n7. 卷积应用" << endl;
73             demo();
74             break;
75         case '8':
76             cout << "\n8. OTSU 算法" << endl;
77             OTSU();
78             break;
79         case '9':
80             cout << "\n9. 其他加分项" << endl;
81             cout <<
82                 " 实验室的团宠 “雪球” -snowball.jpg
83                 (200*150)" << endl;
84             plus_code("images/snowball.jpg");
85             cout << " 多角星形 - polyhedrosis.jpg
86                 (98*90)" << endl;
87             plus_code("images/polyhedrosis.jpg");
88             cout << " 船舰 -ship.jpg (128*96)" <<
89                 endl;
```

```

84         plus_code("images/ship.jpg");
85         cout << "脑部影像截取-brain.jpg
86             (119*78)" << endl;
87         plus_code("images/brain.jpg",5,6);
88         break;
89     case 'A':
90     case 'a':
91         cout << "\nA. 自由发挥" << endl;
92         free_act();
93         break;
94     default:
95         cout << "\n输入错误, 请重新输入" <<
96             endl;
97     }
98     wait_for_enter();
99 }
100 return 0;
}

```

### 4.3 menu.app

```

1 #include <conio.h>
2 #include <iostream>
3
4 using namespace std;
5 int menu()
6 {
7     cout <<

```

```

"*****\n"
8      << "*"      1. 矩阵加法      2. 矩阵数乘
9          3. 矩阵转置      *\n"
10     << "*"      4. 矩阵乘法      5. Hadamard 乘积
11          6. 矩阵卷积      *\n"
12     << "*"      7. 卷积应用      8. OTSU 算法
13          9. 其他加分项  *\n"
14      << "*"      A. your picture  0. 退出系统
15          *\n"
16
17      <<
18
"*****\n"
19
20      <<
21          "7、8、9：点击图片，按下空格可关闭所有图片\n"
22
23      <<
24          "A：点击视频，按下空格可选择视频画面进行变换\n"
25      << "选择菜单项<0~A> :" ;
26
27      char choise = _getch();
28
29      return choise;
30
31  }

```

#### 4.4 normal\_operate.app

```

1 #include <conio.h>
2
3 #include <iostream>
4
5 #include <numeric>
6
7 #include "Matrix.h"
8
9 #define overflow 0
10
11 #define SQUARE_MATRIX 1
12
13 using namespace std;

```

```
8     const int prime_size = 90000;
9     const int increase_size = 1000;
10    const int kernel_size = 3;
11    const int padding = 1;
12    const int stride = 1;
13    const int dilation = 1;
14
15 /*
16     int size;
17
18     // 动态分配内存
19     int *arr = new int[size];
20
21     // 检查内存是否分配成功
22     if (!arr) {
23         cout << "Memory allocation failed." << endl;
24         return 1;
25     }
26
27     // 输入数组元素
28     cout << "Enter " << size << " elements for the
29         array:" << endl;
30     for (int i = 0; i < size; i++) {
31         cin >> arr[i];
32     }
33
34     // 在使用完后释放内存
35     delete[] arr;*/
```

```
36 struct MATRIX {  
37     private:  
38         int size;  
39         int MALLOC(){  
40             Matrix = (int*)malloc(size * sizeof(int));  
41             if (!Matrix)  
42                 return overflow;  
43             for (int i = 0; i < size; i++)  
44                 Matrix[i] = 0;  
45             return true;  
46         }  
47     public:  
48         int row;//行  
49         int col;//列  
50         int* Matrix;  
51  
52         bool Matrix_in(bool square_matrix = 0);  
53         void Matrix_out();  
54         void Matrix_free();  
55         MATRIX(int r = 0, int c = 0, int* matrix = NULL) {  
56             row = r;  
57             col = c;  
58             size = r * c;  
59             if (matrix)  
60                 Matrix = matrix;  
61             else  
62                 MALLOC();  
63         }  
64     };
```

```
65     bool MATRIX::Matrix_in(bool square_matrix)
66 {
67     if (square_matrix) { //方阵
68         if (!row) {
69             cout << "请输入方阵的阶数: " << endl;
70             cin >> row;
71             if (!cin.good() || row <= 0) {
72                 cout << "输入错误, 返回菜单" << endl;
73                 return ERROR;
74             }
75             col = row;
76         }
77         else
78             cout << "kernel矩阵的阶数为: " <<
79                         kernel_size << endl;
80     }
81     else {
82         cout << "请输入矩阵的行和列, 中间用空格隔开: "
83                     << endl;
84         cin >> row;
85         cin >> col;
86         if (!cin.good() || col <= 0 || row <= 0) {
87             cout << "输入错误, 返回菜单" << endl;
88             return ERROR;
89         }
90         //动态申请内存
91         size = row * col;
92         if (MALLOC() == overflow)
```

```
92         return ERROR;
93
94     // 输入项数
95     cout << "请输入矩阵的项， 中间用空格隔开: " <<
96         endl;// 优化一下输入， 做个表格试试???
97
98     for (int i = 0; i < row ; i++) {
99
100         for (int j = 0; j < col; j++) {
101
102             cin >> Matrix[i * col + j];
103
104             if (!cin.good()) {
105
106                 cout << "输入错误，返回菜单" << endl;
107
108                 return ERROR;
109
110             }
111
112         }
113
114     }
115
116     return true;
117
118     void MATRIX::Matrix_out()
119     {
120
121         cout << "运算结果是: " << endl;
122
123         for (int i = 0; i < row; i++) {
124
125             for (int j = 0; j < col; j++)
126
127                 cout << Matrix[i * col + j] << ' ';
128
129             cout << endl;
130
131         }
132
133     }
134
135     void MATRIX::Matrix_free(){
136
137         free(Matrix);
138
139     }
140
141     //1. 矩阵加法
142
143     bool matriplus()
```

```
120 {
121     MATRIX matrix1, matrix2;
122     if (!matrix1.Matrix_in())
123         return ERROR;
124     if (!matrix2.Matrix_in())
125         return ERROR;
126     if (matrix1.row != matrix2.row || matrix1.col != matrix2.col){
127         cout <<
128             "两个矩阵行和列不相等，无法进行加法运算" <<
129             endl;
130         return ERROR;
131     }
132     int row = matrix1.row;
133     int col = matrix1.col;
134     for (int i = 0; i < row; i++) {
135         for (int j = 0; j < col; j++)
136             matrix1.Matrix[i * col + j] +=
137                 matrix2.Matrix[i * col + j];
138     }
139     matrix1.Matrix_out();
140     matrix1.Matrix_free();
141     matrix2.Matrix_free();
142     return true;
143 }
144 //2. 矩阵数乘
145 bool nummulti()
146 {
147     MATRIX matrix;
```

```
145     int a;
146
147     if (!matrix.Matrix_in())
148         return ERROR;
149
150     cout << "请输入一个整数：" << endl;
151
152     cin >> a;
153
154     if (!cin.good()) {
155         cout << "输入错误，返回菜单" << endl;
156         return ERROR;
157     }
158
159     int row = matrix.row;
160
161     int col = matrix.col;
162
163     for (int i = 0; i < row; i++) {
164
165         for (int j = 0; j < col; j++)
166             matrix.Matrix[i * col + j] *= a;
167
168     }
169
170     matrix.Matrix_out();
171
172     matrix.Matrix_free();
173
174     return true;
175 }
176
177 //3. 矩阵转置
178
179 bool matritrans()
180 {
181
182     MATRIX matrix, matrix_trans;
183
184     if (!matrix.Matrix_in())
185         return ERROR;
186
187     matrix_trans.col = matrix.row;
188
189     matrix_trans.row = matrix.col;
190
191     for (int i = 0; i < matrix.row; i++) {
192
193         for (int j = 0; j < matrix.col; j++)
```

```

174         matrix_trans.Matrix[j * matrix_trans.col +
175             i] = matrix.Matrix[i * matrix.col + j];
176     }
177
178     matrix_trans.Matrix_out();
179     matrix.Matrix_free();
180     matrix_trans.Matrix_free();
181     return true;
182 }
183
184 //4. 矩阵乘法
185
186 bool matrimulti()
187 {
188     MATRIX matrix1, matrix2;
189
190     if (!matrix1.Matrix_in())
191         return ERROR;
192
193     if (!matrix2.Matrix_in())
194         return ERROR;
195
196     if (matrix1.col != matrix2.row) {
197         cout << "矩阵 A 的列数和矩阵 B
198             的行数相等, 运算才有意义! " << endl;
199         return ERROR;
200     }
201
202     MATRIX matrix_multi(matrix1.row, matrix2.col);
203
204
205     for (int i = 0; i < matrix_multi.row; i++) {
206         for (int j = 0; j < matrix_multi.col; j++) {
207             for (int k = 0; k < matrix1.col; k++)
208                 matrix_multi.Matrix[i *
209                     matrix_multi.col + j] +=
210                     matrix1.Matrix[i * matrix1.col + k]

```

```

                * matrix2.Matrix[k * matrix2.col +
                j];
            }
        }
        matrix_multi.Matrix_out();
        matrix1.Matrix_free();
        matrix2.Matrix_free();
        matrix_multi.Matrix_free();
        return true;
    }
    //5.Hadamard 乘积
    bool hadamulti()
    {
        MATRIX matrix1, matrix2;
        if (!matrix1.Matrix_in())
            return ERROR;
        if (!matrix2.Matrix_in())
            return ERROR;
        if (matrix1.row != matrix2.row || matrix1.col != matrix2.col) {
            cout <<
                "两个矩阵行和列不相等，无法进行Hadamard乘积运算"
                << endl;
            return ERROR;
        }
        int row = matrix1.row;
        int col = matrix1.col;
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++)

```

```

223             matrix1.Matrix[i * col + j] *=
224                 matrix2.Matrix[i * col + j];
225
226         matrix1.Matrix_out();
227
228         matrix1.Matrix_free();
229
230         matrix2.Matrix_free();
231
232         return true;
233     }
234
235 //6. 矩阵卷积
236
237     int hadamulti(const MATRIX kernel, const MATRIX
238
239         matrix_padding, const int i, const int
240         j)//Hadamard乘积
241
242     {
243
244         int a = 0;
245
246         for (int m = 0; m < kernel_size; m++) {
247
248             for (int n = 0; n < kernel_size; n++)
249
250                 a += kernel.Matrix[m * kernel_size + n] *
251
252                     matrix_padding.Matrix[(m + i) *
253
254                         matrix_padding.col + (n + j)];
255
256         }
257
258         return a;
259     }
260
261     bool conv()//矩阵卷积
262
263     {
264
265         MATRIX matrix, kernel(kernel_size, kernel_size);
266
267         if (!matrix.Matrix_in())
268
269             return ERROR;
270
271         if (!kernel.Matrix_in(SQUARE_MATRIX))
272
273             return ERROR;

```

```
247
248     int N_padding_row = matrix.row + 2 * padding;
249     int N_padding_col = matrix.col + 2 * padding;
250     if (N_padding_row < kernel_size || N_padding_col <
251         kernel_size) {
252         cout << "行列太小，无法做卷积" << endl;
253         return ERROR;
254     }
255     //建造扩展后的矩阵
256     MATRIX matrix_padding(N_padding_row,
257                           N_padding_col);
258     for (int i = 0; i < matrix.row; i++) {
259         for (int j = 0; j < matrix.col; j++) {
260             matrix_padding.Matrix[(i + padding) *
261                                 N_padding_col + (j + padding)] =
262             matrix.Matrix[i * matrix.col + j];
263         }
264     }
265
266     //卷积后的矩阵
267     int N_conv_row = (N_padding_row - kernel_size + 1)
268         / dilation;
269     int N_conv_col = (N_padding_col - kernel_size + 1)
270         / dilation;
271
272     MATRIX matrix_conv(N_conv_row, N_conv_col);
273     for (int i = 0; i < N_conv_row; i++) {
274         for (int j = 0; j < N_conv_col; j++) {
275             matrix_conv.Matrix[i * N_conv_col + j] =
```

```

                hadamulti(kernel, matrix_padding, i, j);
270
        }
271
    }
272
    matrix_conv.Matrix_out();
273
    kernel.Matrix_free();
274
    matrix.Matrix_free();
275
    matrix_padding.Matrix_free();
276
    matrix_conv.Matrix_free();
277
    return true;
}
278
//7. 矩阵应用_求卷积
279
bool conv_eg(int*& image_matrix, int choice)
{
281
    //卷积核
282
    int B[6][9] = { 1, 1, 1, 1, 1, 1, 1, 1, 1,
283
                    -1, -2, -1, 0, 0, 0, 1, 2, 1,
284
                    -1, 0, 1, -2, 0, 2, -1, 0, 1,
285
                    -1, -1, -1, -1, 9, -1, -1, -1, -1,
286
                    -1, -1, 0, -1, 0, 1, 0, 1, 1,
287
                    1, 2, 1, 2, 4, 2, 1, 2, 1 };
288
    int sum = accumulate(B[choice], B[choice] + 9, 0);
289
    MATRIX matrix(256, 256, image_matrix);
290
    MATRIX kernel(kernel_size, kernel_size, B[choice]);
291
    //建造扩展后的矩阵
292
    int N_padding = matrix.row + 2 * padding;
293
    MATRIX matrix_padding(N_padding, N_padding);
294
    for (int i = 0; i < matrix.row; i++) {
295
        for (int j = 0; j < matrix.row; j++) {
296
            matrix_padding.Matrix[(i + padding) *

```

```

N_padding + (j + padding)] =
matrix.Matrix[i * matrix.row + j];

298    }
299 }
300
301 //卷积后的矩阵
302 int N_conv = (N_padding - kernel_size + 1) /
dilation;
303 MATRIX matrix_conv(N_conv, N_conv);
304 for (int i = 0; i < N_conv; i++) {
305     for (int j = 0; j < N_conv; j++) {
306         matrix_conv.Matrix[i * N_conv + j] =
hadamulti(kernel, matrix_padding, i, j);
307         //超出0-255的处理以及加128的修正
308         if (sum)
309             matrix_conv.Matrix[i * N_conv + j] /= sum;
310         else
311             matrix_conv.Matrix[i * N_conv + j] += 128;
312         if (matrix_conv.Matrix[i * N_conv + j] < 0)
313             matrix_conv.Matrix[i * N_conv + j] = 0;
314         else if (matrix_conv.Matrix[i * N_conv +
315             j] > 255)
matrix_conv.Matrix[i * N_conv + j] =
255;
316     }
317 }
318 image_matrix = matrix_conv.Matrix;

```

```
319         matrix.Matrix_free();
320
321         matrix_padding.Matrix_free();
322
323         return true;
324     }
325 }
```

## 4.5 opencv\_operate.app

```
22  /*
23   对vs2019+opencv正确配置后方可使用，此处只给出一段读取并显示图像的
24   */
25
26  Mat image = imread("images/demolena.jpg",
27                      IMREAD_GRAYSCALE); // 图像的灰度值存放在格式为Mat的变量image中
28
29  if (image.empty()) {
30      cerr << "Error: Could not read the image." <<
31          endl;
32
33      return ERROR;
34
35  }
36
37  imshow("Image-original", image);
38
39
40  for (int i = 0; i < 6; i++) {
41      Mat image_copy = image.clone();
42
43      //克隆原图像素值
44
45      int *image_matrix;
46
47      image_matrix = (int*)malloc(sizeof(int) *
48
49          image_size);
50
51      for (int y = 0; y < image.rows; ++y) {
52          for (int x = 0; x < image.cols; ++x)
53              image_matrix[y * image.cols + x] =
54                  (int)image.at<uchar>(y, x);
55
56      }
57
58
59      conv_eg(image_matrix, i);
60
61      for (int y = 0; y < image.rows; ++y) {
62          for (int x = 0; x < image.cols; ++x) {
63
64              image_copy.at<uchar>(y, x) =
```

```

                image_matrix[y * image.cols + x];

43             }

44         }

45         switch (i + 1)

46     {

47         case 1:

48             imshow("Convolution B1", image_copy);

49             break;

50         case 2:

51             imshow("Convolution B2", image_copy);

52             break;

53         case 3:

54             imshow("Convolution B3", image_copy);

55             break;

56         case 4:

57             imshow("Convolution B4", image_copy);

58             break;

59         case 5:

60             imshow("Convolution B5", image_copy);

61             break;

62         case 6:

63             imshow("Convolution B6", image_copy);

64             break;

65         default:

66             break;
67         }
68     }

69     if (waitKey(20000) == ' ')
70         destroyAllWindows();

```

```
71         return true;
72     }
73
74     //8. OTSU 算法
75     bool OTSU()
76     {
77
78         Mat gray_img = imread("images/demolena.jpg",
79                               IMREAD_GRAYSCALE); //  
图像的灰度值存放在格式为Mat的变量image中
80
81         if (gray_img.empty())
82             cerr << "Error: Could not read the image." <<
83             endl;
84
85         return ERROR;
86     }
87
88     imshow("Image-original", gray_img);
89
90     // 进行二值化
91
92     Mat binary_img;
93
94     threshold(gray_img, binary_img, 0, 255,
95               THRESH_BINARY | THRESH_OTSU);
96
97     imshow("binary", binary_img);
98
99     if (waitKey(20000) == ' ')
100
101         destroyAllWindows();
102
103     return true;
104 }
105
106 //9. 其他加分项
107
108 Mat morph_close_open(const Mat& input, int closed_size
109                      = 5, int open_size = 3)
110 {
111
112     Mat kernel = getStructuringElement(MORPH_RECT,
113                                         Size(closed_size, closed_size));
```

```
94     Mat closed;
95
96     morphologyEx(input, closed, MORPH_CLOSE, kernel);
97
98     Mat open;
99     kernel = getStructuringElement(MORPH_RECT,
100                                   Size(open_size, open_size));
101    morphologyEx(closed, open, MORPH_OPEN, kernel);
102
103    return open;
104 }
105
106 bool plus_code(const char* str, int closed_size, int
107 open_size)
108 {
109     Mat gray_img = imread(str, IMREAD_GRAYSCALE); // 图像的灰度值存放在格式为Mat的变量image中
110
111     if (gray_img.empty()) {
112         cerr << "Error: Could not read the image." <<
113             endl;
114
115         return ERROR;
116     }
117
118     Mat binary_img;
119
120     threshold(gray_img, binary_img, 0, 255,
121               THRESH_BINARY | THRESH_OTSU);
122
123     //threshold(gray_img, binary_img, 100, 255,
124     //           THRESH_TOZERO );
125
126
127     // 使用形态学闭运算来填充binary_img内部的空洞
128     binary_img = morph_close_open(binary_img,
129                                   closed_size, open_size);
```

```
116
117     //// 创建遮罩 (mask) , 将背景设置为黑色
118     //Mat mask = Mat::zeros(binary_img.size(), CV_8U);
119     //mask.setTo(255, binary_img); //
120     // 将二值化的binary_img为255区域设置为255 (白色)
121
122     // 创建一个新的图像, 将背景设置为黑色
123     Mat result = gray_img.clone();
124     result.setTo(0, ~binary_img); //
125     // 将~binary_img不为0的区域设置为value (0)
126
127     // 显示原图和处理后的图
128     imshow("Original Image", gray_img);
129     imshow("Processed Image", result);
130     if (waitKey(20000)==' ')
131         destroyAllWindows();
132
133     return true;
134
135
136 //a. 自由发挥
137
138     Mat Myimage, src_gray;
139     Mat dst, detected_edges;
140
141     static void CannyThreshold(int, void*)
142     {
143
144         blur(src_gray, detected_edges, Size(3, 3));
145         Canny(detected_edges, detected_edges,
146             lowThreshold, lowThreshold * ::ratio,
147             kernel_size);
148
149         dst = Scalar::all(0);
150
151 }
```

```
141     Myimage.copyTo(dst, detected_edges);
142     imshow("Your picture", dst);
143 }
144 bool free_act()
145 {
146     namedWindow("Display window");
147     VideoCapture cap(0);
148
149     if (!cap.isOpened())
150     {
151         cout << "cannot open camera";
152         system("pause");
153         return ERROR;
154     }
155     cout << "按下空格选择照片" << endl;
156     while (true)
157     {
158         cap >> Myimage;
159         imshow("Display window(请按下空格选择画面)", Myimage);
160         if (Myimage.empty())
161             return 1;
162         resize(Myimage, Myimage, Size(1280, 720));
163         // cv::resize(img, img, cv::Size(600, 720));
164         // 放缩一下，以便能看到图像全部
165         if (waitKey(25) == ' ')
166             break;
167     }
168     cap >> Myimage;
```

```
168     dst.create(Myimage.size(), Myimage.type());
169     cvtColor(Myimage, src_gray, COLOR_BGR2GRAY);
170     namedWindow("Your picture", WINDOW_AUTOSIZE);
171     createTrackbar("Min Threshold:", "Your picture",
172                   &lowThreshold, max_lowThreshold,
173                   CannyThreshold);
174     CannyThreshold(0, 0);
175     waitKey(0);
176     cap.release(); // 释放对象
177     destroyAllWindows();
178     return true;
179 }
```