同济大学
TONGJI UNIVERSITY

高级语言程序设计大作业（一）

大作业报告

矩阵运算与 OpenCV 图像处理

学　号：　2353900
姓　名：　汪嘉晨
学　院：　国豪书院
班　级：　工科二班

二零二四年五月十五日

一、设计思路与功能描述

（一）设计思路

0，对于整体的架构，“矩阵运算”程序分为三个板块，即一开始库函数、全局变量和声明函数等，main 函数，其他需要调用的封装函数例如矩阵加法函数，暂停程序函数等。

1，至于其他的设计思路，与大作业的要求与指令如出一辙，如图 1.1 所示，~~不再赘述!!!~~

矩阵运算

```
1   #include <conio.h>
2   #include <iostream>
3   #include <opencv2/opencv.hpp>
4   using namespace cv;
5   using namespace std;
6    //全局变量
7    int A[10000] = { 0 }; //矩阵A（最大100*100）
8    int B[10000] = { 0 }; //矩阵B（最大100*100）
9    int C[10000] = { 0 }; //矩阵C
10
11   void wait_for_enter();//暂停程序，等待回车键继续
12   void menu();
13   void matriplus();
14   void nummulti();
15   void matritrans();
16   void matrimulti();
17   void hadamulti();
18   void conv();
19   void demo();
20   void ostu();
21   void segementation();
22
23  int main() { ... }
77
78  void menu() { ... }
89  void wait_for_enter() { ... }
98  void matriplus() { ... }
129 void nummulti() { ... }
153 void matritrans() { ... }
176 void matrimulti() { ... }
220 void hadamulti() { ... }
251 void conv() { ... }
302 void demo() { ... }
441 void ostu() { ... }
453 void segementation() { ... }
```

牛爷爷 1.1 整体架构

（二）功能描述

0，分块讲解之前一些有的没的

（0）输入 1-9 分别对应矩阵加法、矩阵数乘、矩阵转置、矩阵乘法、Hadmard 乘积、矩阵卷积、卷积应用（不同卷积核处理 Lena 图像）、OSTU 算法（二值化算法）与其相关应用（分割背景），功能丰富，基本涵盖了矩阵的~~基本~~运算。

（1）功能 1-7 均使用一维数组，对应加分项一；功能 8 对应加分项二；功能 9 对应加分项三。

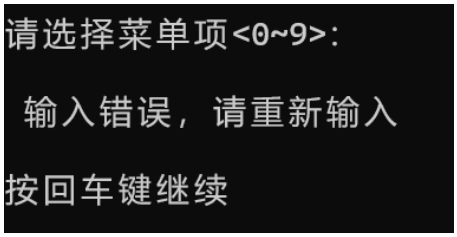
1，主菜单（如图 1.2）

（0）开始时，选择菜单项后不需要输入回车直接跳转到相应选项中（_getch()）。如果输入非法字符（除 0~9 的其他字符），提示输入错误（如图 1.3），按回车即可刷新界面重新回到菜单界面。

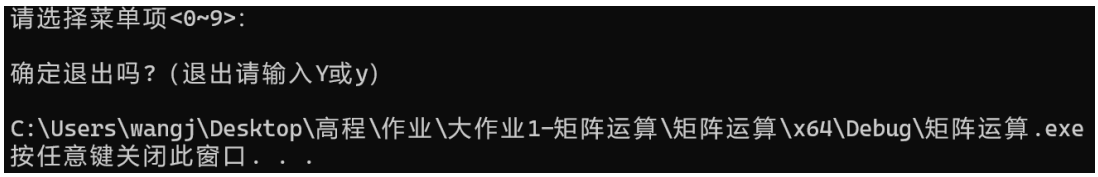
（1）退出系统前用户可进行任意次操作，直到输入 0 并在提示后输入“y”或“Y”方可退出，同样的无需输入回车，方便操作（如图 1.4）。

（2）主菜单不仅展示了功能 1-9 与退出按键 0，还生动形象地指出了自己的姓名与学号，~~并表达了作者对高程尤其是大作业的热爱之情~~。较题目中所要求的<0~8>按键，又添加了“9 Segementation”，方便展示 OSTU 算法对图像处理的更多应用。

牛爷爷 1.2 菜单项



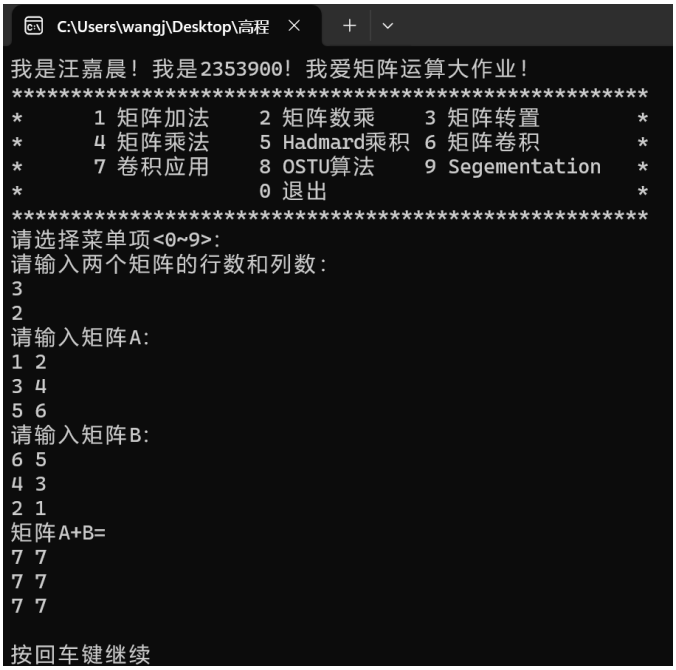
牛爷爷 1.3 输入错误



牛爷爷 1.4 程序退出

2，矩阵加法，矩阵数乘，矩阵 Hadamulti 乘积

这三个较为相似，为了寻求“一维数组”能解决的方式，直接开 A[10000],B[10000],C[10000]来解决问题，可以容纳 100*100 大小的矩阵，对于基本的运算是足够的（如果还需要增加可以在全局变量中更改）。如图 1.5，以矩阵的加法来展示这三种运算的界面。



牛爷爷 1.5 以矩阵的加法为例展示功能 1,2,5 的操作过程

3，矩阵的数乘

同样地，输入行和列，输出只需要行列调换即可（如图 1.6）。

牛爷爷 1.6 矩阵的转置

4，矩阵的乘法

矩阵的乘法要求第一个矩阵的列数（column）和第二个矩阵的行数（row）相同时才有意义。所以先判断第一个矩阵列数是否等于第二个矩阵行数，如果不相等，会输出“矩阵 A 的列数不等于矩阵 B 的行数，无法相乘”。如果符合规则，那么按照规则进行运算（如图 1.7）。

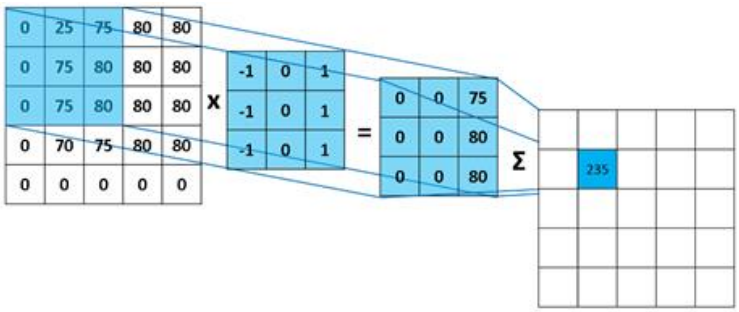
```
请选择菜单项<0~9>:
请输入矩阵A的行数和列数:
2 3
请输入矩阵A
1 0 0
0 0 1
请输入矩阵B的行数和列数:
3 4
请输入矩阵B:
1 0 1 0
0 1 0 1
1 0 1 0
矩阵AB=
1 0 1 0
1 0 1 0

按回车键继续
```

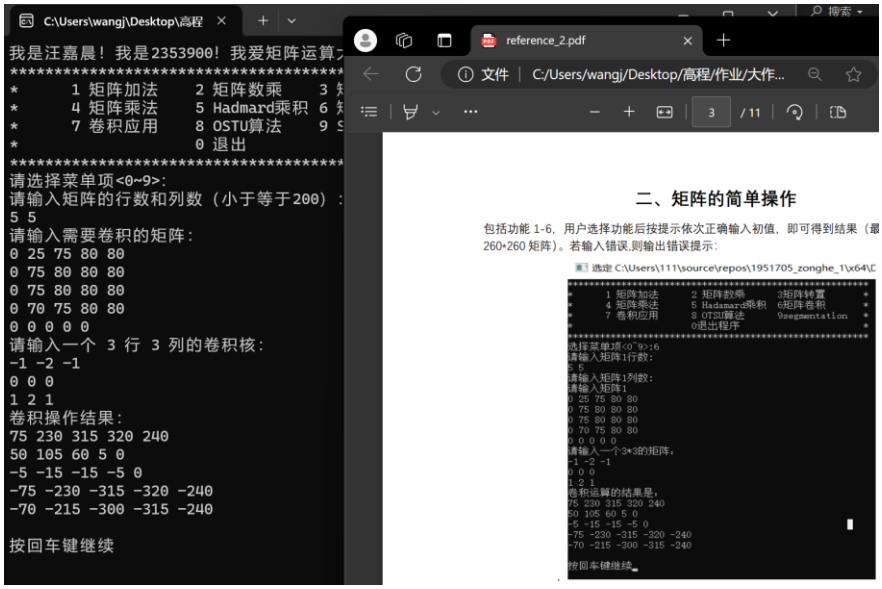
牛爷爷 1.7 矩阵的乘法

5，矩阵卷积

矩阵卷积正如图 1.8 所示，改变卷积核的位置即可，如图 1.9 为卷积运算，没有一点点误差。



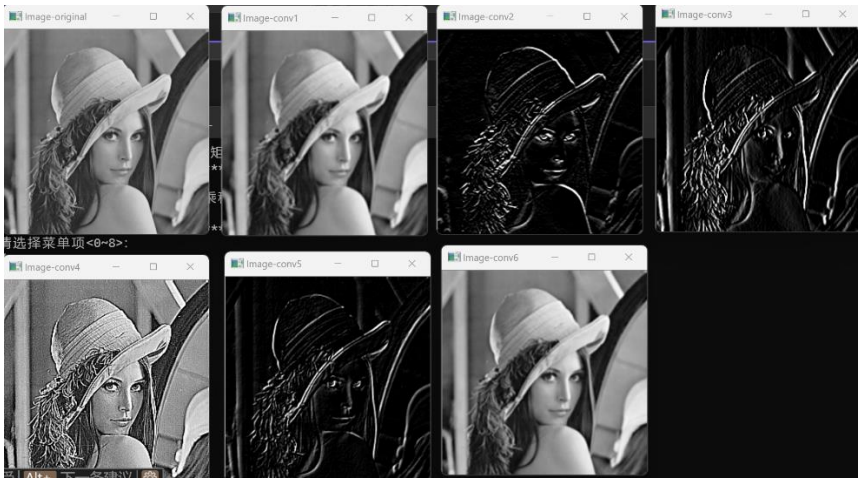
牛爷爷 1.8 题图



牛爷爷 1.9 卷积示例与结果比较

6，卷积应用

卷积应用是 Lena 在六个不同卷积核卷积运算下的处理结果，如图 1.10 与图 1.11 所示。输入“7”直接弹出原图，每输一次空格弹出卷积结果与对应的图像，最后输入回车，所有图像关闭。



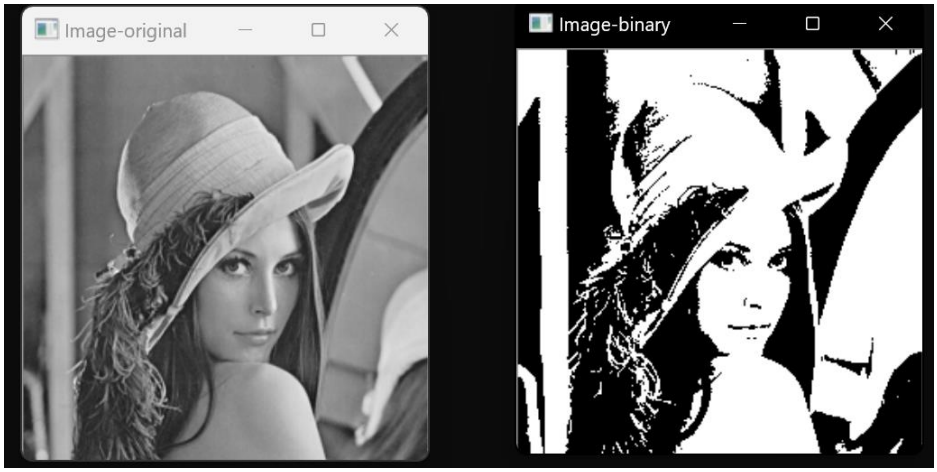
牛爷爷 1.10 卷积应用输出图像

```
 53,  71,  80,  80,  64,  63,  69,  84, 106, 122, 118,  86,  67,  59,  59,  49,  42,
 43,  45,  45,  42,  42,  41,  42,  43,  49,  58,  66,  69,  68,  67,  65,  62,  59,
61,  66,  76,  88,  99, 108, 115, 119, 122, 123, 126, 128, 130, 131, 132, 134, 134, 1
5, 136, 136, 136, 137, 139, 140, 139, 138, 139, 141, 142, 142, 142, 143, 144, 145, 14
, 144, 144, 143, 143, 142, 142, 141, 140, 139, 139, 139, 139, 139, 139, 140, 141, 143
 144, 146, 147, 149, 151, 154, 156, 158, 159, 162, 165, 168, 170, 172, 175, 178, 182,
185, 189, 192, 196, 198, 202, 204, 206, 208, 210, 210, 211, 212, 213, 214, 193, 147,
04,  85,  91,  96,  99, 100, 102, 103, 102, 100,  99, 100,  97,  94,  91,  92,  95,
6,  94,  95, 110, 140, 171, 189, 189, 173, 143, 113,  91,  80,  72,  67,  66,  68,  7
,  73,  72,  73,  77,  87,  97, 104, 104,  99,  91,  84,  76,  70,  67,  71,  80,  90
 100, 106, 111, 110, 106, 100,  94,  83,  73,  66,  65,  66,  67,  64,  57;
 41,  63,  80,  90,  96, 100, 111, 131, 153, 168, 175, 174, 164, 144, 116, 100, 100,
19, 138, 154, 165, 172, 176, 178, 179, 180, 181, 180, 176, 170, 147, 116,  93,  79,
7,  50,  43,  45,  48,  49,  52,  50,  55,  56,  55,  50,  47,  61,  61,  60,  47,  4
,  53,  58,  75,  79,  70,  57,  62,  84,  97,  95,  78,  61,  48,  55,  67,  69,  60
  49,  63,  81,  85,  75,  63,  69,  83, 104, 123, 126, 105,  78,  60,  54,  47,  42,
 42,  43,  44,  44,  45,  44,  44,  44,  47,  52,  56,  58,  57,  58,  59,  63,  68,
77,  88,  99, 106, 112, 116, 121, 123, 125, 125, 127, 128, 129, 130, 132, 133, 133, 1
4, 135, 136, 136, 137, 138, 138, 138, 138, 139, 140, 141, 141, 143, 143, 144, 144, 14
, 145, 145, 144, 143, 142, 142, 141, 141, 139, 138, 139, 140, 141, 140, 140, 141, 142
 143, 145, 147, 149, 151, 153, 155, 157, 159, 161, 164, 167, 171, 173, 176, 178, 181,
184, 188, 191, 194, 197, 200, 203, 205, 207, 209, 210, 211, 212, 213, 214, 203, 160,
14,  85,  87,  93,  96,  99, 101, 103, 103, 103, 102, 102,  99,  97,  97,  99, 102, 1
3, 101, 100, 109, 130, 154, 166, 161, 141, 113,  92,  80,  77,  75,  73,  74,  75,  7
,  74,  74,  77,  85,  95, 103, 106, 102,  92,  82,  76,  70,  67,  66,  74,  86,  97
 106, 110, 113, 110, 105,  99,  90,  79,  69,  64,  64,  64,  64,  59,  54;
 42,  56,  69,  79,  87,  94, 106, 127, 149, 166, 173, 173, 164, 146, 121, 104, 102,
18, 137, 153, 165, 172, 176, 178, 179, 180, 182, 181, 176, 170, 152, 116,  90,  66,
0,  47,  45,  47,  49,  50,  50,  50,  56,  58,  57,  53,  49,  65,  63,  63,  47,  4
,  53,  58,  75,  76,  67,  53,  60,  84, 101, 106,  90,  69,  50,  52,  69,  79,  70
  52,  52,  73,  86,  83,  69,  67,  85, 103, 119, 120, 110,  89,  68,  55,  47,  45,
 45,  47,  47,  45,  44,  42,  42,  44,  47,  50,  53,  56,  61,  66,  72,  80,  89,
98, 106, 112, 115, 118, 120, 123, 124, 126, 126, 126, 127, 128, 129, 131, 132, 133, 1
4, 134, 136, 136, 137, 137, 136, 137, 138, 140, 140, 141, 141, 142, 143, 144, 144, 14
, 146, 145, 144, 143, 142, 143, 142, 142, 140, 139, 139, 141, 141, 141, 141, 141, 142
 143, 145, 147, 149, 150, 152, 155, 157, 158, 160, 163, 166, 169, 172, 176, 178, 181,
184, 187, 190, 193, 195, 199, 201, 204, 206, 208, 210, 211, 212, 212, 214, 209, 173,
26,  88,  83,  89,  94,  97, 100, 103, 105, 106, 106, 104, 102, 101, 103, 106, 109, 1
0, 108, 106, 109, 119, 132, 137, 129, 112,  94,  84,  81,  81,  81,  81,  81,  79,  7
,  74,  75,  81,  91, 100, 104, 102,  94,  84,  76,  71,  67,  65,  69,  81,  94, 104
 110, 113, 114, 110, 105,  97,  87,  75,  67,  63,  62,  62,  59,  55,  45;
 47,  53,  62,  71,  81,  88, 100, 121, 145, 164, 172, 173, 164, 148, 125, 108, 105,
18, 136, 152, 164, 172, 176, 178, 179, 181, 182, 181, 176, 170, 154, 116,  86,  59,
```

牛爷爷 1.11 卷积应用输出卷积结果（太长太长的，展示不下的）

7，OSTU 算法

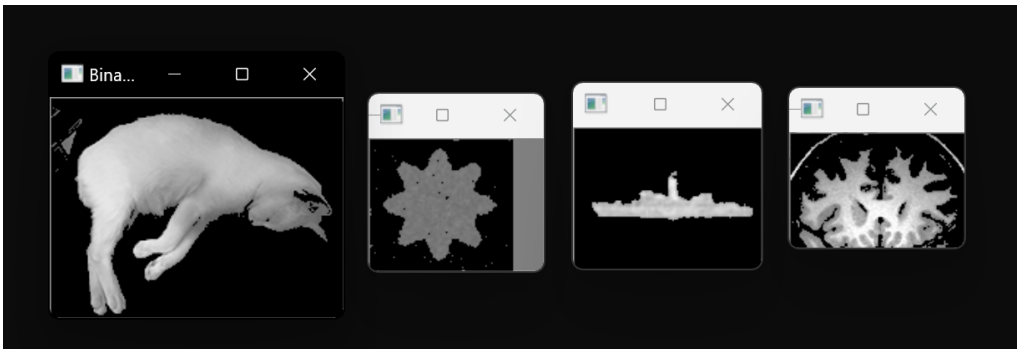
输入“8”，直接输出原图与二值化后的图像，如图 1.11。



牛爷爷 1.11 OSTU 二值化算法示例

8，Segementation——二值化的进一步应用

在本功能里，实现了基本的“抠图，即背景”黑化“，如图 1.12。



牛爷爷 1.12 二值化的进一步应用

二、遇到的问题及解决方法

0，写在前面

在将近两周断断续续程序的编写与调试与编写和调试的循环之中，我遇到了无数问题，又通过互联网搜索，copilot 询问、高程群内相关问题及其解答以及同学交流不断完成、优化每一个部分，在此致谢！

1，先审题再做题，可以事半功倍！——二维数组改一维数组

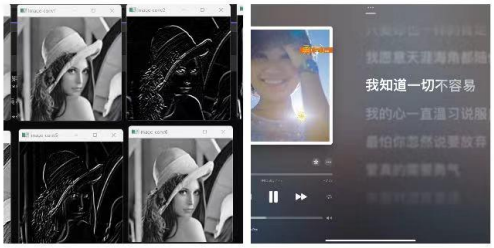
感觉自己有一个看个大概就开始做，做到差不多再继续看题的习惯，这个问题导致了后面发现加分项一（用一维数组实现功能 1-7）变得棘手，因为我前面用的都是二维数组！！！于是在那天的 00：52，在朋友圈呻吟万分（如图 2.1），可是又只能硬着头皮改一维数组。

解决方法：首先是心理层面，平复情绪一定先读题！二维数组改一维数组其实也就相当于串糖葫芦，明确行列的关系即可，改起来也快。

Chen

矩阵运算用二维数组做了半天，加分项1（就一行没看到）说你要用一维数组，我……

我要吃饭还没做😭



昨天 00:52

牛爷爷 2.1 朋友圈的咆哮

2，全局变量的使用

在矩阵运算中，开一个特别大的数组可能会导致内存占用过多。当时是在每一个函数里面声明变量，后来运行时就炸掉了！选择全局变量此时才是大法！（不过应该注意全局变量能不设就不设（？），合适才是好！）

```
//全局变量
int A[10000] = { 0 }; //矩阵A（最大100*100）
int B[10000] = { 0 }; //矩阵B（最大100*100）
int C[10000] = { 0 }; //矩阵C
```

图 2.2 全局变量的使用

3，卷积应用，注意不要超 0 与 255 的限制

群里面有很多人问为什么 1 和 6 不显示（我也出了这个问题但我没问（bushi）），于是在各位专家教授的解答之下，本人悔悟了：

悔悟一，本人忘记了除以 9，1，1，1，1，16 的问题，导致爆掉了！修改完之后如图 2.3。

悔悟二，本人明白了 uchar 还有 0~255 的限制，将 result 中每个数字排查一遍，大于 255 赋值 255，小于 0 赋值 0 解决问题！

悔悟三，float！明确类型，一开始用 int，1/16=0，误差太大！

悔悟四，Mat image = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业 1-矩阵运算\\images\\demolena.jpg", IMREAD_GRAYSCALE);可以直接改成单通道，太好！

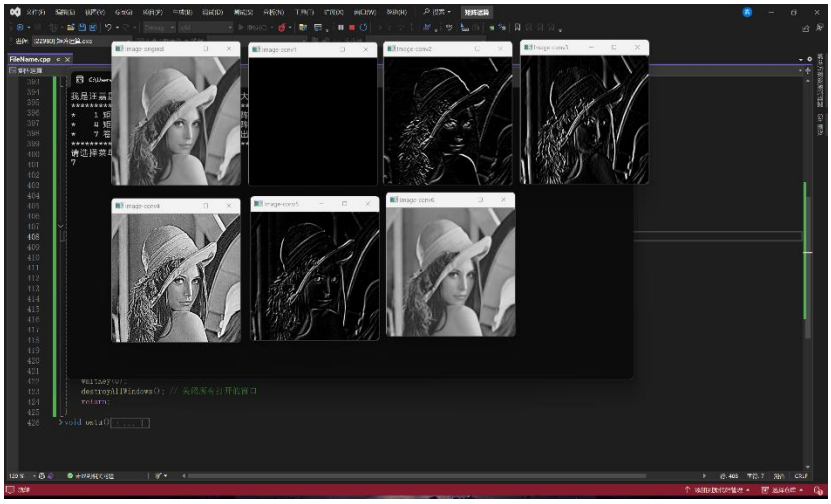


图 2.3 0~255!

4，加分项——空虚心理！恐惧心理！无耻逃避心理!

加分项可以使用库函数，~~那我就不客气了。~~最后也是非常的 amazing 竟然做出来了，加分项三考虑的是二值化图像与原图叠加，超过 255 的赋值 255，小于 0 的赋值 0。但是理想很丰满，现实很骨感！最后的结果中间主体部分有点泛白，猫猫的背景没有做好，可能需要找到具体的阈值处理，而非统一定一个阈值（我定的 127，不知道是否有更合适的）。

由于黑色==0，白色==255，在实际的控制，也是很恶心，比如 120 的像素变成 0，0+120 又回去了，达不到覆盖的效果。解决方法：想到了线代的“穿脱原理”（如图 2.4），把黑的变成 255，白的变成 0，运算之后再返回来不就好了？试了一下效果太好!

```
binary_image1 = 255 - binary_image1;//黑的变成255，白的变成0，方便后面的叠加
binary_image2 = 255 - binary_image2;
binary_image3 = 255 - binary_image3;
binary_image4 = 255 - binary_image4;
image1 = 255 - image1;
image2 = 255 - image2;
image3 = 255 - image3;
image4 = 255 - image4;
Mat result1, result2, result3, result4;
result1 = binary_image1 + image1;
result2 = binary_image2 + image2;
result3 = binary_image3 + image3;
result4 = binary_image4 + image4;
result1 = 255 - result1;
result2 = 255 - result2;
result3 = 255 - result3;
result4 = 255 - result4;//类似于穿脱原理？倒回来
// 限制结果图像的像素值不超过255
result1 = min(result1, 255);
result2 = min(result2, 255);
result3 = min(result3, 255);
result4 = min(result4, 255);
```

图 2.4 穿脱原理

5，图片无法加载，图片无法自动关闭:

waitKey(0);
destroyAllWindows();
可以解决问题

三、心得体会

0，耗时断断续续两周，到馆还是宿舍，白天还是深夜，感谢大作业的陪伴，也许我做的没有那么好，总算是做完了，成就感满满!!!

1，有时不一定使用二维数组，一维数组也可以解决问题，串糖葫芦谁怕谁啊，但是就是有·麻烦。

2，一开始还想行和列都是变化的，是不是要使用动态数组（我还没学啊），后来发现一个数组开足够大就能解决。

3，多多使用 ctrl+s，程序突然爆掉，再打开一夜回到解放前我真的会谢!

4，还有个东西叫“行等价变换”，还有个东西叫“求伴随矩阵”，还有个东西叫“求行列式”，看起来简单，但是如果要寻找计算的各个步骤似乎很难…

5，图片显示太黑或者太白其实还可以调亮度，+-128 似乎不错。

6，旧图新整，嘉人们一定记得读题读题读题!

7，致谢：

感谢陈老师，感谢各位助教，感谢各位直接或者间接帮助我的同学，感谢 baidu，感谢 CSDN，感谢知乎，感谢 copilot chat，感谢那个日夜奋战的自己! ~~（以上感谢除了第一个第二个和最后一个不分前后）~~

四、源代码

```
#include <conio.h>
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace cv;
```

```
using namespace std;
//全局变量
int A[10000] = { 0 }; //矩阵A（最大100*100）
int B[10000] = { 0 }; //矩阵B（最大100*100）
int C[10000] = { 0 }; //矩阵C

void wait_for_enter();//暂停程序，等待回车键继续
void menu();
void matriplus();
void nummulti();
void matritrans();
void matrimulti();
void hadamulti();
void conv();
void demo();
void ostu();
void segementation();


int main()
{

	char choice, ch;
	while (true)
	{
		system("cls"); // 清屏函数
		menu();
		char choice = _getch();//不用输入回车，直接读取
		if (choice == '0') // 选择退出
		{
			cout << "\n确定退出吗?（退出请输入Y或y）" << endl;
			char ch = _getch();
			if (ch == 'y' || ch == 'Y')
				break;
			else
				continue;
		}
		switch (choice)
		{
		case '1':
			matriplus();
			break;
		case '2':
			nummulti();
```

```
            break;
        case '3':
            matritrans();
            break;
        case '4':
            matrimulti();
            break;
        case '5':
            hadamulti();
            break;
        case '6':
            conv();
            break;
        case '7':
            demo();
            break;
        case '8':
            ostu();
            break;
        case '9':
            segementation();
            break;
        default:
            cout << "\n 输入错误，请重新输入" << endl;
        }
        wait_for_enter();
    }
    return 0;
}
void menu()//菜单
{
    cout << "我是汪嘉晨！我是2353900！我爱矩阵运算大作业！" << endl;
    cout << "*************************************************" << endl;
    cout << "*      1 矩阵加法    2 矩阵数乘    3 矩阵转置      *" << endl;
    cout << "*      4 矩阵乘法    5 Hadmard乘积 6 矩阵卷积      *" << endl;
    cout << "*      7 卷积应用    8 OSTU算法    9 Segementation *" << endl;
    cout << "*                    0 退出                       *" << endl;
    cout << "*************************************************" << endl;
    cout << "请选择菜单项<0~9>：" << endl;
}
void wait_for_enter()//暂停程序，等待回车键继续
{
    cout << endl
        << "按回车键继续";
```

```
    while (_getch() != '\r')
        ;
    cout << endl
        << endl;
}
void matriplus()//矩阵加法
{
    cout << "请输入两个矩阵的行数和列数：" << endl;
    int m, n;//矩阵行数，列数
    cin >> m >> n;
    cout << "请输入矩阵A：" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> A[i * n + j];
        }
    }
    cout << "请输入矩阵B：" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> B[i * n + j];
        }
    }
    cout << "矩阵A+B=" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << A[i * n + j] + B[i * n + j] << " ";
        }
        cout << endl;
    }
}
void nummulti()//矩阵数乘
{
    cout << "请输入矩阵的行数和列数：" << endl;
    int m, n,num;//矩阵行数，列数与数
    cin >> m >> n;
    cout << "请输入矩阵：" << endl;
    for (int i = 0; i < m; i++)
    {
```

```
            for (int j = 0; j < n; j++)
            {
                cin >> A[i * n + j];
            }
        }
        cout << "请输入数乘的数：" << endl;
        cin >> num;
        cout<< "矩阵*num=" << endl;
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cout << A[i * n + j] * num << " ";
            }
            cout << endl;
        }
}
void matritrans()//矩阵转置
{
        cout << "请输入矩阵的行数和列数：" << endl;
        int m, n;
        cin >> m >> n;
        cout << "请输入矩阵：" << endl;
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cin >> A[i * n + j];
            }
        }
        cout << "矩阵转置为：" << endl;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                cout << A[j * n + i] << " ";
            }
            cout << endl;
        }
}
void matrimulti()//矩阵乘法
{
        cout << "请输入矩阵A的行数和列数：" << endl;
        int m, n;//矩阵A行数，列数
```

```
    cin >> m >> n;
    cout << "请输入矩阵A" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> A[i * n + j];
        }
    }
    cout << "请输入矩阵B的行数和列数：" << endl;
    int p, q;//矩阵B行数，列数
    cin >> p >> q;
    cout << "请输入矩阵B：" << endl;
    for (int i = 0; i < p; i++)
    {
        for (int j = 0; j < q; j++)
        {
            cin >> B[i * q + j];
        }
    }
    if (n != p)
    {
        cout << "矩阵A的列数不等于矩阵B的行数，无法相乘" << endl;
        return;
    }
    cout << "矩阵AB=" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < q; j++)
        {
            int sum = 0;
            for (int k = 0; k < n; k++)
            {
                sum += A[i * n + k] * B[k * q + j];
            }
            cout << sum << " ";
        }
        cout << endl;
    }
}
void hadamulti()//Hadamard乘积
{
    cout << "请输入两个矩阵的行数和列数：" << endl;
    int m, n;//矩阵行数，列数
```

```
    cin >> m >> n;
    cout << "请输入矩阵A： " << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> A[i * n + j];
        }
    }
    cout << "请输入矩阵B： " << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> B[i * n + j];
        }
    }
    cout << "Hadamard乘积=" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << A[i * n + j] * B[i * n + j] << " ";
        }
        cout << endl;
    }
}
void conv()
{
    char back = '0';
    int m = 0, n = 0;
    cout << "请输入矩阵的行数和列数： " << endl;
    cin >> m >> n;
    cout << "请输入需要卷积的矩阵： " << endl;
    for (int i = 0; i < m; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            cin >> A[i * n + j];
        }
    }
    cout << "请输入一个 3 行 3 列的卷积核： " << std::endl;
    for (int i = 0; i < 3; ++i)
    {
```

```
            for (int j = 0; j < 3; ++j)
            {
                cin >> B[i * 3 + j];
            }
        }
        // 卷积操作并输出结果
        cout << "卷积操作结果：" << endl;
        for (int i = 0; i < m; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                int sum = 0;
                for (int ki = -1; ki <= 1; ++ki)
                {
                    for (int kj = -1; kj <= 1; ++kj)
                    {
                        int ni = i + ki;
                        int nj = j + kj;
                        // 对超出边界的部分进行padding
                        if (ni < 0 || ni >= m || nj < 0 || nj >= n)
                        {
                            sum += 0; // padding 0
                        }
                        else
                        {
                            sum += A[ni * n + nj] * B[(ki + 1) * 3 + (kj + 1)];
                        }
                    }
                }
                cout << sum << " ";
            }
            cout << endl;
        }
        return;
}
void demo()//卷积应用
{
        Mat image = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业1-矩阵运算
\\images\\demolena.jpg", IMREAD_GRAYSCALE);
        //图像的灰度值存放在格式为Mat的变量image中
        // 宽度和高度
        int width = image.cols;
        int height = image.rows;
        //输入卷积核B1-B6
```

```
    float kernel1[9] = { 1.0f / 9, 1.0f / 9, 1.0f / 9, 1.0f / 9, 1.0f / 9, 1.0f / 9, 1.0f / 9, 1.0f / 9, 1.0f / 9 };
    float kernel2[9] = { -1, -2, -1, 0, 0, 0, 1, 2, 1 };
    float kernel3[9] = { -1, 0, 1, -2, 0, 2, -1, 0, 1 };
    float kernel4[9] = { -1, -1, -1, -1, 9, -1, -1, -1, -1 };
    float kernel5[9] = { -1, -1, 0, -1, 0, 1, 0, 1, 1 };
    float kernel6[9] = { 1.0f/16, 2.0f/16, 1.0f/16, 2.0f/16, 4.0f/16, 2.0f/16, 1.0f/16, 2.0f/16, 1.0f/16 };
    //卷积运算
    Mat result1 = image.clone();
    for (int y = 1; y < image.rows - 1; y++)
    {
        for (int x = 1; x < image.cols - 1; x++)
        {
            float sum1 = 0;
            for (int j = -1; j <= 1; j++)
            {
                for (int i = -1; i <= 1; i++)
                {
                    sum1 += kernel1[(j + 1) * 3 + (i + 1)] * (float)(image.at<uchar>(y - j, x - i));
                }
            }
            if(sum1>=255) sum1=255;
            if(sum1<=0) sum1=0;
            result1.at<uchar>(y, x) = sum1;
        }
    }
    Mat result2= image.clone();
    for (int y = 1; y < image.rows - 1; y++)
    {
        for (int x = 1; x < image.cols - 1; x++)
        {
            float sum2 = 0;
            for (int j = -1; j <= 1; j++)
            {
                for (int i = -1; i <= 1; i++)
                {
                    sum2 += kernel2[(j + 1) * 3 + (i + 1)] * (float)(image.at<uchar>(y - j, x - i));
                }
            }
            if(sum2>=255) sum2=255;
            if(sum2<=0) sum2=0;
```

```
                result2.at<uchar>(y, x) = sum2;
            }
        }
        Mat result3= image.clone();
        for (int y = 1; y < image.rows - 1; y++)
        {
            for (int x = 1; x < image.cols - 1; x++)
            {
                float sum3 = 0;
                for (int j = -1; j <= 1; j++)
                {
                    for (int i = -1; i <= 1; i++)
                    {
                        sum3 += kernel3[(j + 1) * 3 + (i + 1)] * (float)(image.at<uchar>(y - j,
x - i));
                    }
                }
                if(sum3>=255) sum3=255;
                if(sum3<=0) sum3=0;
                result3.at<uchar>(y, x) = sum3;
            }
        }
        Mat result4= image.clone();
        for (int y = 1; y < image.rows - 1; y++)
        {
            for (int x = 1; x < image.cols - 1; x++)
            {
                float sum4 = 0;
                for (int j = -1; j <= 1; j++)
                {
                    for (int i = -1; i <= 1; i++)
                    {
                        sum4 += kernel4[(j + 1) * 3 + (i + 1)] * (float)(image.at<uchar>(y - j,
x - i));
                    }
                    if(sum4>=255) sum4=255;
                    if(sum4<=0) sum4=0;
                    result4.at<uchar>(y, x) = sum4;//亮度太低了！！！！
                }
            }
        }
        Mat result5= image.clone();
        for (int y = 1; y < image.rows - 1; y++)
        {
```

```
            for (int x = 1; x < image.cols - 1; x++)
            {
                float sum5= 0;
                for (int j = -1; j <= 1; j++)
                {
                    for (int i = -1; i <= 1; i++)
                    {
                        sum5 += kernel5[(j + 1) * 3 + (i + 1)] * (float)(image.at<uchar>(y - j,
x - i));
                    }
                }
                if(sum5>=255) sum5=255;
                if(sum5<=0) sum5=0;
                result5.at<uchar>(y, x) = sum5;
            }
        }
        Mat result6= image.clone();
        for (int y = 1; y < image.rows - 1; y++)
        {
            for (int x = 1; x < image.cols - 1; x++)
            {
                float sum6 = 0;
                for (int j = -1; j <= 1; j++)
                {
                    for (int i = -1; i <= 1; i++)
                    {
                        sum6 += kernel6[(j + 1) * 3 + (i + 1)] * (float)(image.at<uchar>(y - j,
x - i));
                    }
                    if(sum6>=255) sum6=255;
                    if(sum6<=0) sum6=0;
                    result6.at<uchar>(y, x) = sum6;
                }
            }
        }
        cout << result1 << endl;
        imshow("Image-original", image);//显示原图像
        waitKey(0);//等待按键
        imshow("Image-conv1", result1);
        waitKey(0);
        cout << result2 << endl;
        imshow("Image-conv2", result2);
        waitKey(0);
        cout << result3 << endl;
```

```
	imshow("Image-conv3", result3);
	waitKey(0);
	cout << result4 << endl;
	imshow("Image-conv4", result4);
	waitKey(0);
	cout << result5 << endl;
	imshow("Image-conv5", result5);
	waitKey(0);
	cout << result6 << endl;
	imshow("Image-conv6", result6);
	waitKey(0);
	destroyAllWindows(); // 关闭所有打开的窗口
	return;
}
void ostu()//大津法 加分项2 二值化
{
	Mat image = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业1-矩阵运算
\\images\\demolena.jpg", IMREAD_GRAYSCALE);
	//图像的灰度值存放在格式为Mat的变量image中
	Mat binaryImage;
	threshold(image, binaryImage, 0, 255, THRESH_OTSU);
	imshow("Image-original", image);//显示原图像
	waitKey(0);
	imshow("Image-binary", binaryImage);
	waitKey(0);
	destroyAllWindows();
	return;
}
void segementation()//二值化PLUS，加分项3
{
	Mat image1 = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业1-矩阵运算
\\images\\snowball.jpg", IMREAD_GRAYSCALE);
	Mat image2 = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业1-矩阵运算
\\images\\polyhedrosis.jpg", IMREAD_GRAYSCALE);
	Mat image3 = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业1-矩阵运算
\\images\\ship.jpg", IMREAD_GRAYSCALE);
	Mat image4 = imread("C:\\Users\\wangj\\Desktop\\高程\\作业\\大作业1-矩阵运算
\\images\\brain.jpg", IMREAD_GRAYSCALE);
	//二值化图像（取一个黑白模子）
	Mat binary_image1, binary_image2, binary_image3, binary_image4;
	threshold(image1, binary_image1, 127, 255, THRESH_OTSU);
	threshold(image2, binary_image2, 127, 255, THRESH_OTSU);
	threshold(image3, binary_image3, 127, 255, THRESH_OTSU);
	threshold(image4, binary_image4, 127, 255, THRESH_OTSU);
```

```
    binary_image1 = 255 - binary_image1;//黑的变成255，白的变成0，方便后面的叠加
    binary_image2 = 255 - binary_image2;
    binary_image3 = 255 - binary_image3;
    binary_image4 = 255 - binary_image4;
    image1 = 255 - image1;
    image2 = 255 - image2;
    image3 = 255 - image3;
    image4 = 255 - image4;
    Mat result1, result2, result3, result4;
    result1 = binary_image1 + image1;
    result2 = binary_image2 + image2;
    result3 = binary_image3 + image3;
    result4 = binary_image4 + image4;
    result1 = 255 - result1;
    result2 = 255 - result2;
    result3 = 255 - result3;
    result4 = 255 - result4;//类似于穿脱原理？倒回来
    // 限制结果图像的像素值不超过255
    result1 = min(result1, 255);
    result2 = min(result2, 255);
    result3 = min(result3, 255);
    result4 = min(result4, 255);
    imshow("Binary Image1", result1);
    waitKey(0);
    imshow("Binary Image2", result2);
    waitKey(0);
    imshow("Binary Image3", result3);
    waitKey(0);
    imshow("Binary Image4", result4);
    waitKey(0);
    destroyAllWindows();
    return;
}
```

（共计 507 行）