高程实验报告——

# 从矩阵操作到初识 OPENCV


班级：济勤学堂八班

学号：1952731

姓名：姚忠豪


完成日期：2019/12/06

*（第一次修改:2019/12/12）*

## （1） 设计思路与功能描述：

**设计思路：**

在灵感方面，由于矩阵运算是线性代数中最基本的数学运算并且是数字图像处理的基础，所以思考能否设计一个**矩阵操作器**，既能够实现矩阵的基本运算（如加减，数乘，乘法，转置，卷积等操作），又可以应用矩阵卷积简单处理图像问题，最后借此机会也可以学习 OpenCV 并且实现简单的图像处理；

在实现方面，基本矩阵运算可以使用数组实现，由于作业鼓励使用一维数组，所以整个工程**只使用一维数组**（二维数组在内存中的实际储存形式其实就是一维数组，只要通过简单的下标转换即可实现），对于图像处理，则需要先**配置 OpenCV** 环境并且**自学基于 C++的 OpenCV**（其中了解 Mat 类型的本质和熟悉 **Mat 类型与数组的相互转化**尤为关键）；

在用户体验方面，首先，尽量给予用户以更简便，**更舒适的体验**，大到整体布局，小到输入提示与错误检测，（就拿输入矩阵的方式来说，既要简化用户的输入方式，又要确保用户输入的准确性，这就需要优化输入方式）；其次就是要让用户细致地了解每一个功能（这些则需要通过输出文字性提示解决的）等等；（具体实现方法见后文）

**功能描述**:

**1. 能够在主动退出程序之前进行多次操作**

**2. 能够完成基本矩阵运算（数乘，加法，乘法，转置，卷积，Hadamard 乘积）**

**3. 能够应用卷积对灰度图实现自定义卷积滤波器操作**

**4. 能够使用阈值算法对灰度图进行二值化**

**5. 能够使用 OpenCV 库函数对图像进行提取分割**

## （2） 遇到的问题与解决方法：

**问题 1：**在设计主菜单的过程中，发现每个函数执行完之后显示结果的时间极短就跳至主菜单处;

**解决方法：**

参考给出模板中的_getch函数（*功能：在windows平台下从控制台无回显地取一个字符，在Linux下是有回显的；返回值：从键盘读取的字符;*）所以可以根据他的功能设计下列语句，实现等待观察运算或显示结果并且键入任意键返回的功能，（代码如下）;

```
//输出返回菜单提示
    cout << "\n\t按任意键返回菜单...";
    back = _getch ();
```

**问题 2：**在定义矩阵的过程中，出现了堆栈的问题（*错误 C6262：此警告指出在函数内检测到了超出预设阈值的堆栈使用率。默认情况下，当堆栈大*

*小超过 16K 字节时会生成此警告*)

**解决方法：**

主要的可行的解决方法有两种，第一种为使用动态内存来代替静态内存，或者更改为使用全局变量；由于动态变量可能会带来内存泄漏并且在所学知识点之外，所以我选择使用后者全局变量

```
//定义全局变量
int matrix_A [sizeL * sizeL] = {0} ……
```

**问题 3：**在输入矩阵的过程中，由于需要优化用户的使用体验，所以最好不要让用户在输入矩阵之前输入矩阵的行与列，应该在用户输入矩阵之后自动计算出行与列，并判断是否可以进行相应的矩阵运算；如果不能进行计算，则重新输入矩阵而不是直接返回主菜单；

**解决方法：**

参考 MATLAB 的矩阵输入方式，可以使用 cin 一个数字再 getchar () 一个字符的形式进行输入，遇到指定字符时表示换行（如：输入 3,2,4,1;2,3,4,5;3,4,5,6 表示输入了一个 3 行 4 列的矩阵，并且可以计数来保存矩阵的行数和列数）（代码如下）；

```
rowA = 0; colA = 0; rowB = 0; colB = 0; i = 0; j = 0;
        cout << "\n请输入矩阵A：" << endl;
        cin >> matrixA[i];
        while ((c = getchar ())! = '\n') {
            cin >> matrixA[++i];
            if (c == ';') rowA++;
        }
        rowA++;
        i++;
        colA = i / rowA;
```

对于如何判断是否能够进行运算，判断行列数即可，再使用死循环输入，能够运算时跳出循环（注意更新数组的值）；

**问题 4：**进行多次不同运算操作时，发现矩阵的数值会发生改变，通过 debug 发现由于使用了全局变量，每个函数使用的数组内存地址都相同，所以会继承第一次使用函数的数组值（也就是矩阵中的元素值）；

**解决方法：**每次函数前使用遍历以更新全局变量的值（代码如下）；

```
//更新全局变量的值
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixB[i] = 0;
    }
```

**问题 5：**卷积运算过程中的运用一维数组将一个矩阵实现 padding 操作遇到的困难

**解决方法：**

定义一个新的全局变量，数组大小比 256 阶 高 2 阶，通过原矩阵一维数组下标与填充后矩阵的一维数组下标的关系比较，进行遍历赋值，（代码如下）；

```
void conv_padding (int A [], int APad [], int row, int col) {
    int colAPad, rowAPad, i;
    colAPad = col + 2;
    rowAPad = row + 2;
    for (i = 0; i < row * col; i++) {
        APad [i + col + 3 + 2 * int (i / col)] = A[i];
```

```
    }
}
```

**问题 6：**Mat 与数组的相互转化

**解决方法**：首先，要对 Mat 类型有一个清晰的认识，了解 Mat 的属性以及数据存储形式（*在 C++的编程体系下，Mat 类在 OpenCV 中是最重要的一种图像表示形式（即为 Matrix）；这里的 Mat （如 CV_8UC3）可以指定存储的数据类型、行列数，以及每个元素中的数值个数（通道数）等等;*）我的思路是先使用了 new 分配空间一个新的数组空间，将 Mat 类型的值遍历赋值到数组中，对该数组（矩阵）进行卷积操作后，定义一个新的 Mat 值，再将新数组（矩阵）遍历赋值到 Mat 类型中保存，最后 delete 分配的空间；（部分代码如下）；

```
//将Mat类型转化为一维数组
    int* array = new int [sizeL * sizeL];
    for (i = 0; i < sizeL * sizeL; i++) {
        array[i] = 0;
    }
    for (i = 0; i < image. rows * image.cols; i++) {
        array[i] = image.at<uchar> (i / image. cols, i % image. cols);
    }
    rowA = image. rows;
    colA= image. cols;


  ……//卷积操作……

//将一维数组转化为Mat类型
    Mat result = Mat (image. rows, image. cols, CV_8UC1, Scalar::
all(0));
    for (i = 0; i < image. rows * image. cols; i++) {
        result.at<uchar> (i / image. cols, i % image. cols) =
convD[i];
    }
```

最后再将 Mat 类型显示为图像;

**问题 7：**成功转化图像，但图像结果总是只显示一小部分（如图）:


判断应该不是 Mat 与数组相互转换的问题，因为 Mat 类型的数据储存形式与数组是类似的，是连续的，而图像的显示则是原图像截取的，显然不是 Mat 与数组互转的问题

**解决方法：**

考虑是否读取图片时发生了错误，我读取图片的代码为

```
Mat image = imread("demolena.jpg");
```

当灰度参数为默认参数时，读取为了 3 通道数据

（*其函数原型为 imread(const string& filename, int flag=1) flag=-1 时，8 位深度，原通道 flag=0，8 位深度，1 通道 flag=1，8 位深度 ，3 通道 flag=2，原深度，1 通道 flag=3，原深度，3 通道 flag=4，8 位深度 ，3 通道*）

于是读取单通道灰度图时应该用以下读取方式

```
Mat image = imread("demolena.jpg", 1);
```

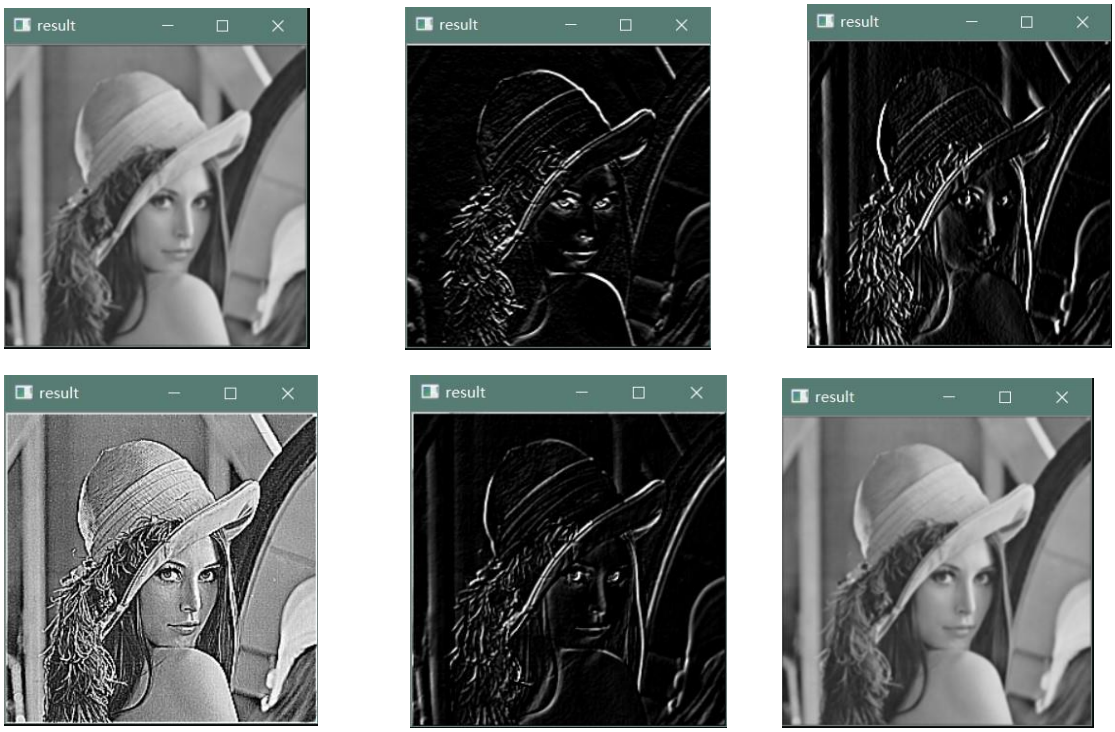
**问题 8：**转化了全图但是显示的图像并不是期望显示的图像（经过查阅资料发现，题中的滤波器包括边缘检测滤波，中值滤波，锐化滤波

等等，图片效果可以想象)；

第一时间考虑是否卷积操作有问题，但是通过输出检测发现每一个卷积值都是正确的，为什么得不到想要的图像呢？

**解决方法：**

通过对灰度值的了解发现，灰度值为 uchar 类型（8 位）的取值范围为 0~255（*白色与黑色之间按对数关系分成若干级，称为“灰度等级”；范围一般从 0 到 255，白色为 255，黑色为 0，故黑白图片也称灰度图像*)，而卷积后的结果中有部分会发生越界，若不处理，则就会发生上图情况；我的处理方法是，当灰度值小于 0 时，将其设置为 0，当灰度值大于 255 时，将其设置为 255；于是得到了以下结果：

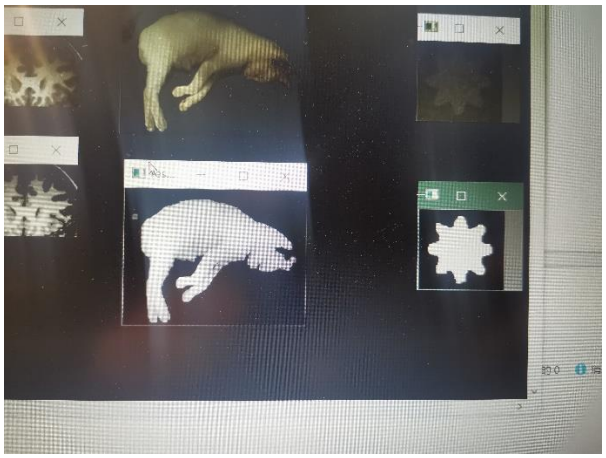


**问题 9：**在 OTSU 算法二值化的基础上对图像进行处理时，使用 OpenCV 库函数发现并不尽人意；本想直接使用阈值分割函数将背景调为黑色，其余部分不变，但结果还是不行，有些图像有噪点（不好意思，忘了保存），有些图像彩色变成了灰度图……

```
cvtColor (src2, src2_br, COLOR_BGRA2GRAY);
threshold (src2_br, src2_br, 0, 255, THRESH_OTSU | THRESH_TOZERO);
```

**解决方法：**

对于有**噪点**的图片，采用 **OpenCV 的形态学开闭操作去除噪点**，但是并不是所有图像都可以采用此方法：



上图中多角星形图案是直接去除噪点后加亮，效果不错，可是猫却不理想；

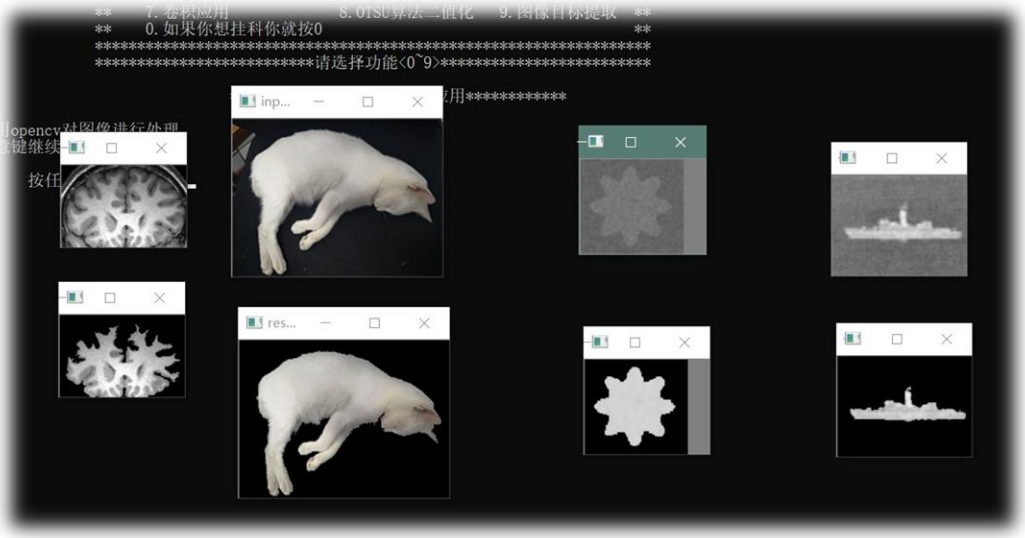
因此，对于图像分割提取，我决定采用的方法是**二值化+OpenCV.findContours 函数去除多余的轮廓，再将原图对保留下来的轮廓进行覆盖**，达到对图像分割提取的目的；

```
    /*获取与过滤轮廓*/
    vector<vector<Point>>contours1;
    findContours (src1_br, contours1, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_NONE);
    vector<vector<Point> >::const_iterator it1 = contours1.begin();
    while (it1! = contours1.end ())
    {
```

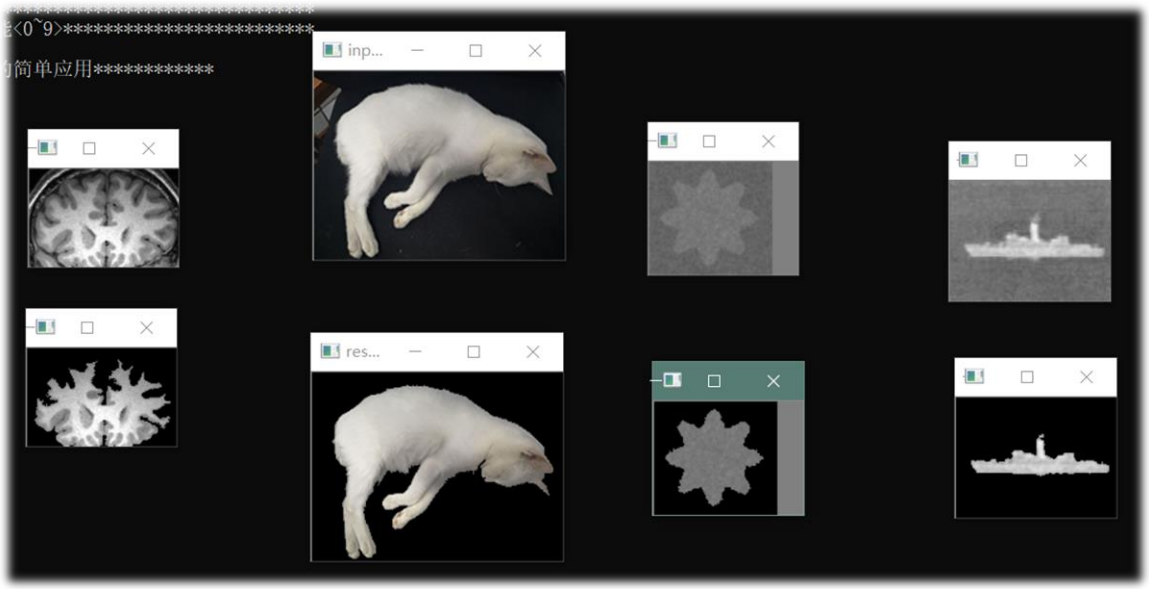
```
        if (it1->size () < 100)
            it1 = contours1.erase(it1);
        else
            ++it1;
    }
    Mat dst1(src1.size(), CV_8U, Scalar (0));
    drawContours (dst1, contours1, -1, Scalar (255), CV_FILLED);
    cvtColor (dst1, dst1, CV_GRAY2RGB);

    /*遍历保留与抹去*/
    for (int i = 0; i < src1.rows; i++) {
        for (int j = 0; j < 3 * src1.cols; j++) {
            if (dst1.at<uchar> (i, j) == 0) {
                src1.at<uchar> (i, j) = 0;
            }
        }
    }
```

第一次的效果如下（与题目要求的图片提取范围与效果基本符合）:




**由于加分项要求是需要应用一个函数将四张图片达到要求的效果，所以对四张图片均使用处理雪球的方法，就可以达到目的。（最终效果图如下）**




**总结：**以上问题都是在实验过程中遇到的，解决的方法或网上查阅相关文献和社区博客，或自学 OpenCV 相关内容（具体见心得），或 TA 和老师给出的相关提示，或自我思考所得；

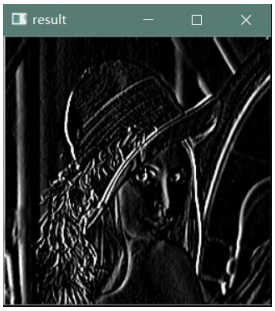
## （3） 心得体会：

通过这一次大作业，体验感其实还不错，本以为是简单的用一维数组实现矩阵操作，没想到还要自学 OpenCV 相关知识，即让我们的自学能力得到了提高，也让我们学到了简单数字图像处理技术；然后关于我自己对卷积处理图像的理解如下；

不同的卷积核对应这不同的算子，对图像处理后的结果也不相同，题给的卷积核的功能包括**模糊，锐化，边缘检测**；比如算子 `-1, -1 ,-1; -1, 9, -1;  -1, -1, -1` 为锐化算子，由该卷积核结构可知，相当于中间的像素加上四周像素的分差，自然就将图片锐化了 ；对于卷积核 1 和 6，从图像上看与原图并没有什么区别，但仔细看会发现清晰度发生了变化，很好理解，卷积核 1 属于中值滤波 `1,  1 , 1;  1, 1 ,  1; 1,  1,  1`，相

当于将一张图片的像素值平均化了，自然就会变得模糊；当然，这些都是比较简单的个人理解；

**锐化算子** **Sobel 算子** **Sobel 算子**




对于更具体的理解，我查阅了相关知识文献，拿 Sobel 算子举个栗子，Sobel 有两个滤波卷积算子，一个用来计算横向的梯度 Gx，一个用来计算纵向梯度 Gy，如果梯度大于某个值（阈值）则被认定为是边缘点并标记；`-1，0，1，-2，0，2，-1，0，1`与 `-1，-2，-1，0，0，0，1，2，1`（*Sobel 算子的原理，对传进来的图像像素做卷积，卷积的实质是在求梯度值，或者说给了一个加权平均，其中权值就是所谓的卷积核；然后对生成的新像素灰度值做阈值运算，以此来确定边缘信息*）Sobel 算子的优势还对于像素位置的影响做了加权，使边缘检测更精确；

对于加分第三项，其实给的图像还是很仁慈的，处理的部分也还可以接受，（只不过是要对 OpenCV 库函数及其功能熟悉）我感觉最重要的还是要先观测二值化以后的图像和需要提取的部分有什么联系，比如是否多出了噪点（如果有，就利用 OpenCV 库函数去除），是否有较大的不需要的部分（如果有，就通过某种方法过滤掉），最后再达到目的；

还有,感谢学习网站 B 站,我就是在 B 站了解 OpenCV 的 QAQ!

## 源代码:

```
//包含头文件
#include <iostream>
#include <conio.h>
#include <opencv2/opencv.hpp>
#include <cmath>

//命名空间预处理
using namespace cv;
using namespace std;
#define sizeL 256

//定义全局变量
int matrixA[sizeL * sizeL] = { 0 }, matrixB[sizeL * sizeL] = { 0 }, matrixC[sizeL *
sizeL] = { 0 }, matrixAPad[(sizeL + 2) * (sizeL + 2)] = { 0 }, convD[sizeL * sizeL];

//声明函数
void tip();
void matriplus();
void nummulti();
void matritrans();
void matrimulti();
void hadamulti();
void conv_padding(int A[], int APad[], int row, int col);
void conv_ope(int APad[], int B[], int conv[], int row, int col);
int conv_sum(int array[], int row, int col);
void conv();
void wait_for_enter();
void menu();
void demo();
void otsu();
Mat extract_ope(Mat src);
void extract();

 //主函数
int main()
{
```

```
    // 定义相关变量
    char choice = 0, ch = 0;

    wait_for_enter();
    while (true)
    {
        system("cls"); //清屏
        menu(); //调用菜单显示函数
        choice = _getch();//选择
        if (choice == '0') //选择退出
        {
            cout << "\n\t\t\t\t你确定要退出吗? (按 \"Y\" 退出)" << endl;
            ch = _getch();
            if (ch == 'y' || ch == 'Y')
                break;
            else
                continue;
        }

        switch (choice)
        {
        case '1':matriplus(); break;//矩阵加法
        case '2':nummulti(); break;//矩阵数乘
        case '3':matritrans(); break;//矩阵转置
        case '4':matrimulti(); break;//矩阵乘法
        case '5':hadamulti(); break;//Hadamard乘积
        case '6':conv(); break;//矩阵卷积
        case '7':demo(); break;//卷积应用
        case '8':otsu(); break;//二值法
        case '9':extract(); break;//图像目标提取
        default:
            cout << "\n\t\t\t\t   输入错误，请重新输入QAQ..." << endl;
            wait_for_enter();
        }
    }
    return 0;
}

//清屏并等待输入
void wait_for_enter()
{
    cout << endl << "\t\t\t\t\t按回车继续...";
    while (_getch() != '\r')
        ;
```

```
    cout << endl << endl;
}

//菜单函数
void menu() {
    cout << "\t\t**********************欢迎来到矩阵操作器**********************\n";
    cout << "\t\t*****************************************************************\n";
    cout << "\t\t**                          --菜单--                           **\n";
    cout << "\t\t**    1.矩阵加法          2.矩阵数乘          3.矩阵转置       **\n";
    cout << "\t\t**    4.矩阵乘法          5.Hadamard乘积      6.矩阵卷积       **\n";
    cout << "\t\t**    7.卷积应用          8.OTSU算法二值化    9.图像目标提取   **\n";
    cout << "\t\t**    0.如果你想挂科你就按0                                    **\n";
    cout << "\t\t*****************************************************************\n";
    cout << "\t\t**************************请选择功能<0~9>**************************\n";
}

//提示函数
void tip() {
    cout << "\n(注:输入矩阵时,请使用英文输入法,每行元素之间以逗号',’隔开,以分号';’为分行的标志,末行结尾不需要分号。)\n例：2,3,1;4,3,2;3,6,8\n";
}

//矩阵加法函数
void matriplus() {

    //输出提示并定义相关变量
    cout << "\n\t\t\t\t************矩阵加法************\n";
    tip();
    int rowA = 0, colA = 0, rowB = 0, colB = 0;
    int i = 0, j = 0;
    char back, c;

    //更新全局变量的值
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixB[i] = 0;
    }

    //输入两矩阵并判断运算条件，不满足则重新输入
    while (true) {
        for (i = 0; i < rowA * colA; i++) {
            matrixA[i] = 0;
```

```
        }
        for (j = 0; j < rowB * colB; j++) {
            matrixB[j] = 0;
        }
        rowA = 0; colA = 0; rowB = 0; colB = 0; i = 0; j = 0;
        cout << "\n请输入矩阵A：" << endl;
        cin >> matrixA[i];
        while ((c = getchar()) != '\n') {
            cin >> matrixA[++i];
            if (c == ';') rowA++;
        }
        rowA++;
        i++;
        colA = i / rowA;
        cout << "\t矩阵A\n";
        for (i = 0; i < rowA * colA; i++) {
            cout << "\t" << matrixA[i];
            if ((i + 1) % colA == 0) cout << endl;
        }
        cout << "请输入矩阵B：" << endl;
        cin >> matrixB[j];
        while ((c = getchar()) != '\n') {
            cin >> matrixB[++j];
            if (c == ';') rowB++;
        }
        rowB++;
        j++;
        colB = j / rowB;
        if (rowA == rowB && colA == colB) break;
        cout << "\t矩阵A与矩阵B不能相加QAQ\n\t请重新输入两个同型矩阵...\n";
    }
    cout << "\t矩阵B\n";
    for (i = 0; i < rowB * colB; i++) {
        cout << "\t" << matrixB[i];
        if ((i + 1) % colB == 0) cout << endl;
    }

    //计算并输出两矩阵之和
    cout << "\n\t矩阵A矩阵B之和为：\n";
    for (i = 0; i < rowA * colA; i++) {
        matrixA[i] = matrixA[i] + matrixB[i];
    }
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << matrixA[i];
```

```
        if ((i + 1) % colA == 0) cout << endl;
    }

    //输出返回菜单提示
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}

//矩阵数乘函数
void nummulti() {

    //输出提示并定义相关变量
    cout << "\n\t\t\t\t**********矩阵数乘**********\n";
    tip();
    int rowA = 0, colA = 0;
    int i = 0, n = 0;
    char back, c;

    //更新全局变量的值
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    i = 0;

    //输入矩阵
    cout << "\n请输入矩阵A: " << endl;
    cin >> matrixA[i];
    while ((c = getchar()) != '\n') {
        cin >> matrixA[++i];
        if (c == ';') rowA++;
    }
    rowA++;
    i++;
    colA = i / rowA;
    cout << "\t矩阵A\n";
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << matrixA[i];
        if ((i + 1) % colA == 0) cout << endl;
    }

    //输入整数
    cout << "请输入整数B: \n";
    cin >> n;
```

```
    //计算结果并输出
    cout << "\t矩阵A与整数B的乘积为：\n";
    for (i = 0; i < rowA * colA; i++) {
        matrixA[i] = matrixA[i] * n;
    }
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << matrixA[i];
        if ((i + 1) % colA == 0) cout << endl;
    }

    //输出返回菜单提示
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}

//转置函数
void matritrans() {

    //输出提示并定义相关变量
    cout << "\n\t\t\t\t**********矩阵转置**********\n";
    tip();
    int rowA = 0, colA = 0;
    int i = 0, n = 0, j = 0, k = 0;
    char back, c;

    //更新全局变量的值
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    i = 0;

    //输入矩阵
    cout << "\n请输入矩阵A：" << endl;
    cin >> matrixA[i];
    while ((c = getchar()) != '\n') {
        cin >> matrixA[++i];
        if (c == ';') rowA++;
    }
    rowA++;
    i++;
    colA = i / rowA;
    cout << "\t矩阵A\n";
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << matrixA[i];
```

```
            if ((i + 1) % colA == 0) cout << endl;
        }

        //输出转置后的结果
        cout << "\n\t矩阵A转置之后的矩阵为：\n";
        for (i = 0, n = 0; j < colA; j++, n++) {
            for (k = 0, i = n; k < rowA; i = i + colA, k++) {
                cout << "\t" << matrixA[i];
            }
            cout << endl;
        }

        //输出返回菜单提示
        cout << "\n\t按任意键返回菜单...";
        back = _getch();
}

//矩阵乘法函数
void matrimulti() {

        //输出提示并定义相关变量
        cout << "\n\t\t\t\t************矩阵乘法************\n";
        tip();
        int rowA = 0, colA = 0, rowB = 0, colB = 0;
        int i = 0, j = 0;
        char back;
        char c;

        //更新全局变量的值
        for (i = 0; i < sizeL * sizeL; i++) {
            matrixA[i] = 0;
        }
        for (i = 0; i < sizeL * sizeL; i++) {
            matrixB[i] = 0;
        }
        for (i = 0; i < sizeL * sizeL; i++) {
            matrixC[i] = 0;
        }

        //输入两矩阵并判断运算条件，不满足则重新输入
        while (true) {
            for (i = 0; i < rowA * colA; i++) {
                matrixA[i] = 0;
            }
```

```
        for (j = 0; j < rowB * colB; j++) {
            matrixB[j] = 0;
        }
        rowA = 0; colA = 0; rowB = 0; colB = 0; i = 0; j = 0;
        cout << "\n请输入矩阵A：" << endl;
        cin >> matrixA[i];
        while ((c = getchar()) != '\n') {
            cin >> matrixA[++i];
            if (c == ';') rowA++;
        }
        rowA++;
        i++;
        colA = i / rowA;
        cout << "\t矩阵A\n";
        for (i = 0; i < rowA * colA; i++) {
            cout << "\t" << matrixA[i];
            if ((i + 1) % colA == 0) cout << endl;
        }
        cout << "请输入矩阵B：" << endl;
        cin >> matrixB[j];
        while ((c = getchar()) != '\n') {
            cin >> matrixB[++j];
            if (c == ';') rowB++;
        }
        rowB++;
        j++;
        colB = j / rowB;
        if (colA == rowB) break;
        cout << "\t矩阵A与矩阵B不能相乘QAQ\n\t请重新输入两个矩阵,保证矩阵A的列数等于B
的行数...\n";
    }
    cout << "\t矩阵B\n";
    for (i = 0; i < rowB * colB; i++) {
        cout << "\t" << matrixB[i];
        if ((i + 1) % colB == 0) cout << endl;
    }

    //进行乘法运算并输出结果
    cout << "\n\t矩阵A矩阵B相乘为：\n";
    for (i = 0; i < rowA; i++) {
        for (j = 0; j < colB; j++) {
            for (int a = 0; a < colA; a++) {
                matrixC[colB * i + j] += matrixA[i * colA + a] * matrixB[a * colB +
j];
```

```
            }
        }
    }
    for (i = 0; i < rowA * colB; i++) {
        cout << "\t" << matrixC[i];
        if ((i + 1) % colB == 0) cout << endl;
    }

    //输出返回菜单提示
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}

//矩阵Hadamard乘积函数
void hadamulti() {

    //输出提示并定义相关变量
    cout << "\n\t\t\t\t**********矩阵Hadamard乘积**********\n";
    tip();
    int rowA = 0, colA = 0, rowB = 0, colB = 0;
    int i = 0, j = 0;
    char back;
    char c;

    //更新全局变量的值
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixB[i] = 0;
    }

    //输入两矩阵并判断运算条件，不满足则重新输入
    while (true) {
        for (i = 0; i < rowA * colA; i++) {
            matrixA[i] = 0;
        }
        for (j = 0; j < rowB * colB; j++) {
            matrixB[j] = 0;
        }
        rowA = 0; colA = 0; rowB = 0; colB = 0; i = 0; j = 0;
        cout << "\n请输入矩阵A：" << endl;
        cin >> matrixA[i];
        while ((c = getchar()) != '\n') {
```

```
            cin >> matrixA[++i];
            if (c == ';') rowA++;
        }
        rowA++;
        i++;
        colA = i / rowA;
        cout << "\t矩阵A\n";
        for (i = 0; i < rowA * colA; i++) {
            cout << "\t" << matrixA[i];
            if ((i + 1) % colA == 0) cout << endl;
        }
        cout << "请输入矩阵B：" << endl;
        cin >> matrixB[j];
        while ((c = getchar()) != '\n') {
            cin >> matrixB[++j];
            if (c == ';') rowB++;
        }
        rowB++;
        j++;
        colB = j / rowB;
        if (rowA == rowB && colA == colB) break;
        cout << "\t矩阵A与矩阵B不能进行Hadamard相乘QAQ\n\t请重新输入两个同型矩
阵...\n";
    }
    cout << "\t矩阵B\n";
    for (i = 0; i < rowB * colB; i++) {
        cout << "\t" << matrixB[i];
        if ((i + 1) % colB == 0) cout << endl;
    }

    //Hadamard乘积运算并输出结果
    cout << "\n\t矩阵A矩阵B的Hadamard乘积为：\n";
    for (i = 0; i < rowA * colA; i++) {
        matrixA[i] = matrixA[i] * matrixB[i];
    }
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << matrixA[i];
        if ((i + 1) % colA == 0) cout << endl;
    }

    //提示返回菜单
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}
```

```
//卷积填充函数
void conv_padding(int A[], int APad[], int row, int col) {
    int colAPad, rowAPad, i;
    colAPad = col + 2;
    rowAPad = row + 2;
    for (i = 0; i < row * col; i++) {
        APad[i + col + 3 + 2 * int(i / col)] = A[i];
    }
}

//卷积操作函数
void conv_ope(int APad[], int B[], int conv[], int row, int col) {
    int i, colAPad, con_i;
    colAPad = col + 2;
    for (i = 0; i < row * col; i++) {
        con_i = i + col + 3 + 2 * int(i / col);
        conv[i] = B[0] * APad[con_i - colAPad - 1] + B[1] * APad[con_i - colAPad] +
B[2] * APad[con_i - colAPad + 1]
            + B[3] * APad[con_i - 1] + B[4] * APad[con_i] + B[5] * APad[con_i + 1]
            + B[6] * APad[con_i + colAPad - 1] + B[7] * APad[con_i + colAPad] + B[8] *
APad[con_i + colAPad + 1];
    }
}

//卷积函数
void conv() {

    //输出提示并定义相关变量
    cout << "\n\t\t\t\t**********矩阵卷积**********\n";
    tip();
    int rowA = 0, colA = 0, rowB = 0, colB = 0, colAPad = 0, rowAPad = 0, con_i = 0;
    int const kernelSize = 3;
    int i = 0, j = 0;
    char c, back;

    //更新全局变量的值
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixB[i] = 0;
    }
    for (i = 0; i < (sizeL + 2) * (sizeL + 2); i++) {
```

```
        matrixAPad[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        convD[i] = 0;
    }
    i = 0;

    //输入被卷积矩阵
    cout << "\n请输入被卷积矩阵A: " << endl;
    cin >> matrixA[i];
    while ((c = getchar()) != '\n') {
        cin >> matrixA[++i];
        if (c == ';') rowA++;
    }
    rowA++;
    i++;
    colA = i / rowA;
    cout << "\t被卷积矩阵\n";

    //输出被卷积矩阵
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << matrixA[i];
        if ((i + 1) % colA == 0) cout << endl;
    }

    //输入卷积核矩阵
    while (true) {
        for (j = 0; j < rowB * colB; j++) {
            matrixB[j] = 0;
        }
        rowB = 0; colB = 0; j = 0;
        cout << "请输入卷积核方阵B（3阶）: " << endl;
        cin >> matrixB[j];
        while ((c = getchar()) != '\n') {
            cin >> matrixB[++j];
            if (c == ';') rowB++;
        }
        rowB++;
        j++;
        colB = j / rowB;
        if (rowB == 3 && colB == 3) break; //判断卷积核是否为三阶方阵
        cout << "\t你输入的不是3阶方阵，请重新输入...\n";
    }
```

```
    //输出卷积核
    cout << "\t卷积核\n";
    for (i = 0; i < rowB * colB; i++) {
        cout << "\t" << matrixB[i];
        if ((i + 1) % colB == 0) cout << endl;
    }

    //填充操作
    conv_padding(matrixA, matrixAPad, rowA, colA);
    rowAPad = rowA + 2;
    colAPad = colA + 2;

    //卷积操作
    conv_ope(matrixAPad, matrixB, convD, rowA, colA);

    //输出卷积后的矩阵
    cout << "\n\tAB卷积结果为：\n";
    for (i = 0; i < rowA * colA; i++) {
        cout << "\t" << convD[i];
        if ((i + 1) % colA == 0) cout << endl;
    }

    //提示返回菜单
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}

//卷积核元素之和
int conv_sum(int array[],int row,int col) {
    int sum=0,i=0;
    for (; i < row * col;i++) {
        sum = sum + array[i];
    }
    return sum;
}

//卷积的应用——图像处理
void demo() {

    //输出提示并定义相关变量
    cout << "\n\t\t\t\t**********矩阵卷积的应用**********\n";
    tip();
    cout << "\n你可以利用3阶的自定义卷积核对Lena图像进行操作\n";
    int rowA = 0, colA = 0, rowB = 0, colB = 0, colAPad = 0, rowAPad = 0, con_i =
```

```
0,i=0,j=0;
    char c,back;
    Mat image = imread("demolena.jpg",0);//图像的灰度值存放在格式为Mat的变量image中


    //更新全局变量
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixA[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        matrixB[i] = 0;
    }
    for (i = 0; i < (sizeL + 2) * (sizeL + 2); i++) {
        matrixAPad[i] = 0;
    }
    for (i = 0; i < sizeL * sizeL; i++) {
        convD[i] = 0;
    }



    //将Mat类型转化为一维数组
    int* array = new int[sizeL * sizeL];
    for (i = 0; i < sizeL * sizeL; i++) {
        array[i] = 0;
    }
    for (i = 0; i < image.rows * image.cols; i++) {
        array[i] = image.at<uchar>(i / image.cols, i % image.cols);
    }
    rowA = image.rows;
    colA= image.cols;

    //输入卷积核矩阵
    while (true) {
        for (j = 0; j < rowB * colB; j++) {
            matrixB[j] = 0;
        }
        rowB = 0; colB = 0; j = 0;
        cout << "\n请输入卷积核B（3阶）：" << endl;
        cin >> matrixB[j];
        while ((c = getchar()) != '\n') {
            cin >> matrixB[++j];
            if (c == ';') rowB++;
        }
```

```
        rowB++;
        j++;
        colB = j / rowB;
        if (rowB == 3 && colB == 3) break; //判断卷积核是否为三阶方阵
        cout << "\t你输入的不是3阶方阵，请重新输入...\n";
    }

    //输出卷积核
    cout << "\t卷积核\n";
    for (i = 0; i < rowB * colB; i++) {
        cout << "\t" << matrixB[i];
        if ((i + 1) % colB == 0) cout << endl;
    }

    //填充操作
    conv_padding(array, matrixAPad, rowA, colA);
    rowAPad = rowA + 2;
    colAPad = colA + 2;

    //卷积操作
    conv_ope(matrixAPad, matrixB, convD, rowA, colA);

    //除以卷积核
    if (conv_sum(matrixB, rowB, colB) != 0) {
        for (i = 0; i < rowA * colA; i++) {
            convD[i] = convD[i] / conv_sum(matrixB, rowB, colB);
        }
    }

    //处理越界像素
    for (i = 0; i < image.rows * image.cols; i++) {
        if (convD[i] < 0) convD[i] = 0;
        if (convD[i] > 255) convD[i] = 255;
    }

    //数组转Mat类型
    Mat result = Mat(image.rows, image.cols, CV_8UC1, Scalar::all(0));
    for (i = 0; i < image.rows * image.cols; i++) {
        result.at<uchar>(i / image.cols, i % image.cols) = convD[i];
    }

    //显示原图
    namedWindow("original", WINDOW_AUTOSIZE);
    imshow("original", image);
```

```
    //显示处理后图像
    namedWindow("result", WINDOW_AUTOSIZE);
    imshow("result", result);
    waitKey(0);
    delete[] array;

    //提示返回菜单
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}

//OTSU算法二值化
void otsu() {

    //输出提示
    char  begin,back;
    cout << "\n\t\t\t\t**********OTSU算法二值化**********\n";
    cout << "\n你将用OTSU算法对图像进行二值化\n按任意键显示结果...\n";
    begin = _getch();

    //读入图片
    Mat src = imread("demolena.jpg");

    //判断读入图片是否成功
    if (src.empty()) {
        printf("could not load image...\n");
    }

    //显示原图
    namedWindow("input");
    imshow("input", src);

    //二值法操作(未使用openCV的二值法阈值函数)
    int t = 127;
    Mat gray, binary;
    cvtColor(src, gray, COLOR_BGR2GRAY);
    Scalar m = mean(src);
    t = int(m[0]);
    binary = Mat::zeros(src.size(), CV_8UC1);
    int height = src.rows;
    int width = src.cols;// 直接读取图像像素
    for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
```

```
            int pv = gray.at<uchar>(row, col);
            if (pv > t) {
                binary.at<uchar>(row, col) = 255;
            }
            else {
            binary.at<uchar>(row, col) = 0;
            }
        }
    }


    //显示二值法操作后的图片
    imshow("result", binary);
    waitKey(0);


    //提示返回菜单
    cout << "\n\t按任意键返回菜单...";
    back = _getch();

}

//图像分割提取
void extract() {

    //输出提示
    char  begin, back;
    cout << "\n\t\t\t\t**********openCV的简单应用**********\n";
    cout << "\n你将用openCV对图像进行分割提取处理\n按任意键显示结果...\n";
    begin = _getch();

    //读入图片，建立新Mat变量
    Mat src1 = imread("brain.jpg");
    Mat src2 = imread("ship.jpg");
    Mat src3 = imread("snowball.jpg");
    Mat src4 = imread("polyhedrosis.jpg");
    Mat result1, result2, result3, result4;

    //显示原图
    namedWindow("input1", CV_WINDOW_AUTOSIZE);
    imshow("input1", src1);
    namedWindow("input2", CV_WINDOW_AUTOSIZE);
    imshow("input2", src2);
    namedWindow("input3", CV_WINDOW_AUTOSIZE);
```

```
    imshow("input3", src3);
    namedWindow("input4", CV_WINDOW_AUTOSIZE);
    imshow("input4", src4);

    //目标提取操作
    result1 = extract_ope(src1);
    result2 = extract_ope(src2);
    result3 = extract_ope(src3);
    result4 = extract_ope(src4);

    //显示结果
    namedWindow("result1", CV_WINDOW_AUTOSIZE);
    imshow("result1", result1);
    namedWindow("result2", CV_WINDOW_AUTOSIZE);
    imshow("result2", result2);
    namedWindow("result3", CV_WINDOW_AUTOSIZE);
    imshow("result3", result3);
    namedWindow("result4", CV_WINDOW_AUTOSIZE);
    imshow("result4", result4);

    waitKey(0);

    //提示返回菜单
    cout << "\n\t按任意键返回菜单...";
    back = _getch();
}

//图像分割提取操作函数
Mat extract_ope(Mat src) {

    /*定义相关变量*/
    Mat src_br;

    /*二值化*/
    cvtColor(src, src_br, COLOR_BGRA2GRAY);
    threshold(src_br, src_br, 0, 255, THRESH_OTSU | THRESH_BINARY);

    /*获取与过滤轮廓*/
    vector<vector<Point>>contours;
    findContours(src_br, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
    vector<vector<Point> >::const_iterator it = contours.begin();
    while (it != contours.end())
    {
        if (it->size() < 100)
```

```
            it = contours.erase(it);
        else
            ++it;
    }
    Mat dst(src.size(), CV_8U, Scalar(0));
    drawContours(dst, contours, -1, Scalar(255), CV_FILLED);
    cvtColor(dst, dst, CV_GRAY2RGB);

    /*遍历保留与抹去*/
    for (int i = 0; i < src.rows; i++) {
        for (int j = 0; j < 3 * src.cols; j++) {
            if (dst.at<uchar>(i, j) == 0) {
                src.at<uchar>(i, j) = 0;
            }
        }
    }

    /*返回结果图像*/
    return src;
}
```