《矩阵运算》大作业报告

班级：智能交通与车辆 14 班

学号：2152402

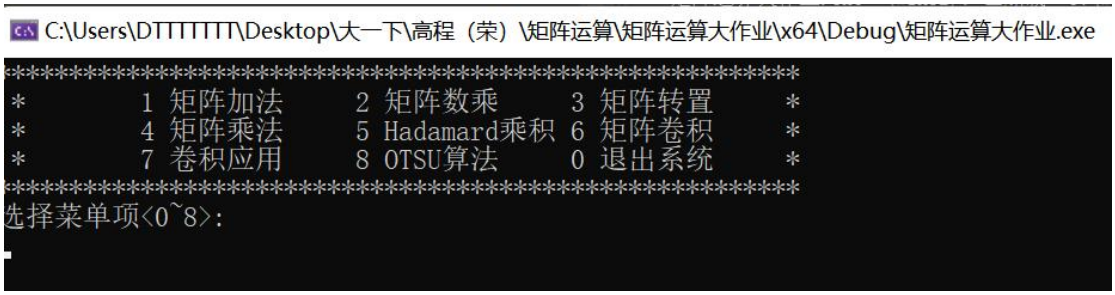
姓名：段婷婷

完成日期：2022.3.30

一. 设计思路与功能描述

1. 主菜单

主菜单如下，用户可以自行选择菜单项进行所需运算。在退出系统前，用户可以进行任意次操作。



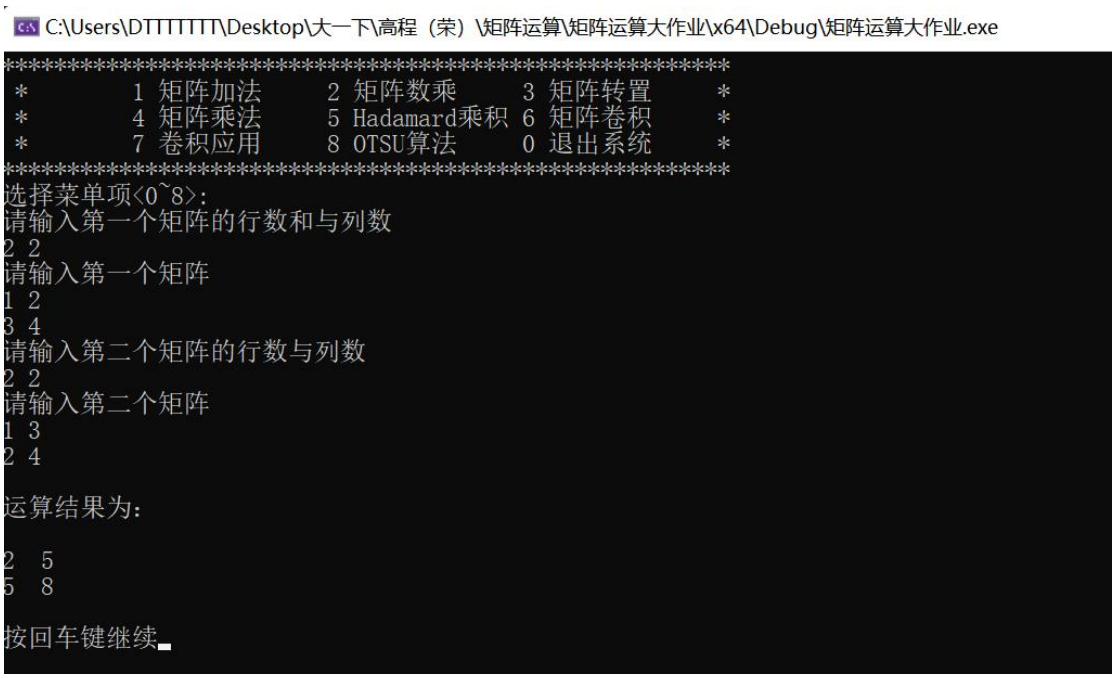
（注：用户在选择菜单项后不需要输入回车就能直接跳转到相应选项中）

如果输入非法字符（除 0~8 的其他字符），提示输入错误，按回车刷新界面后回到菜单界面。

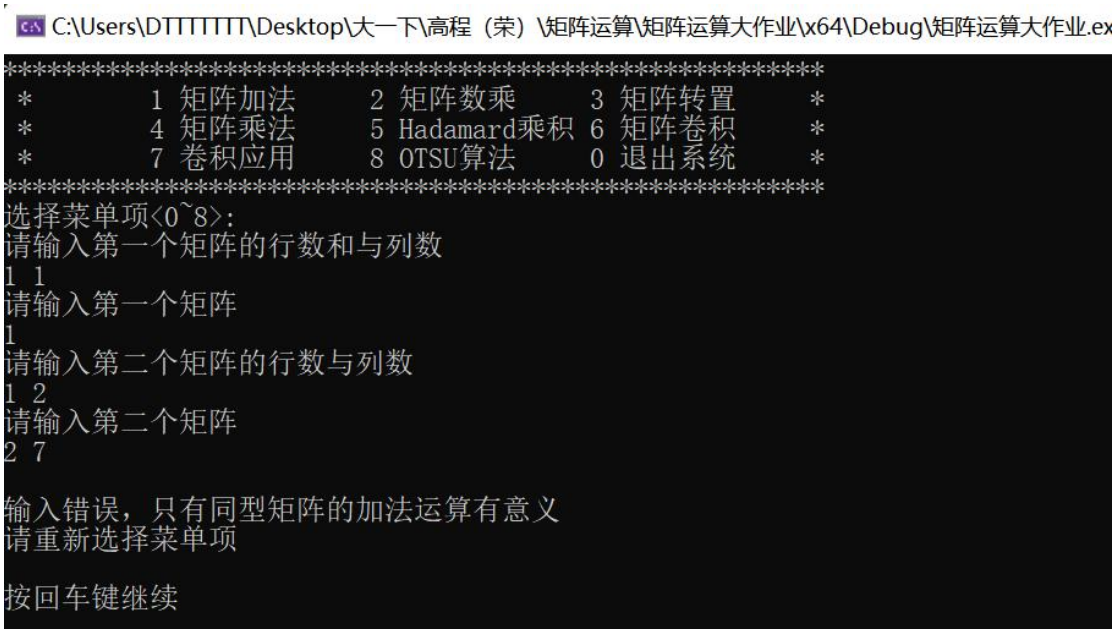
2. 菜单所包括的功能运算：

（1）矩阵加法：

选择 1 后，按照输入提示依次输入第一个矩阵的行数、列数、完整矩阵和第二个矩阵的行数、列数、完整矩阵后，会输出答案矩阵。（样例如下：）

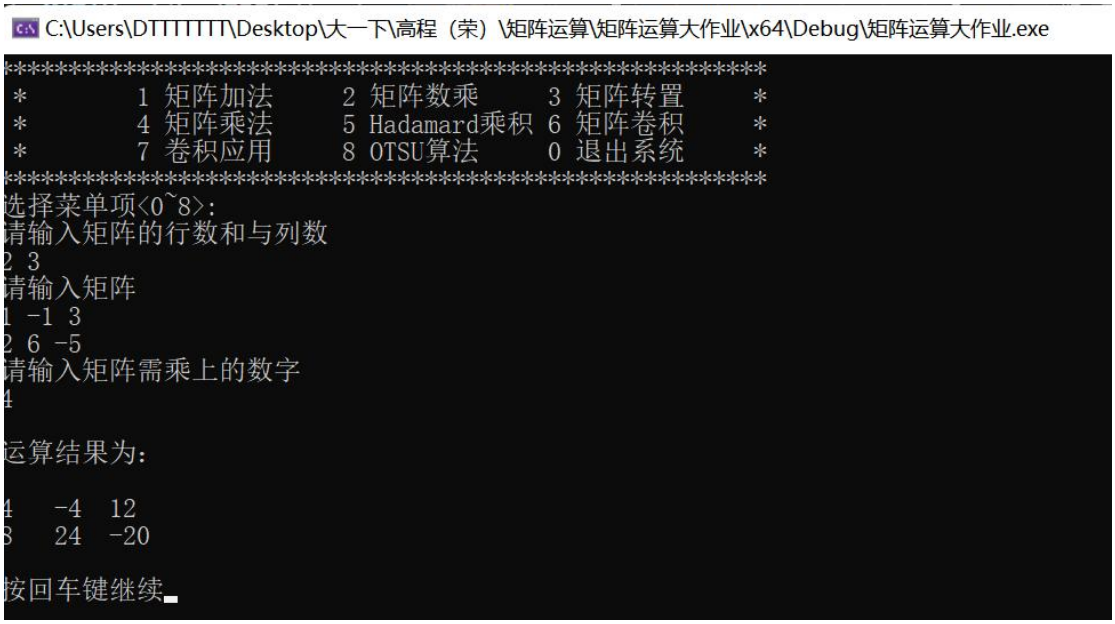


若输入不合法，会报错，刷新后重新选择菜单项：



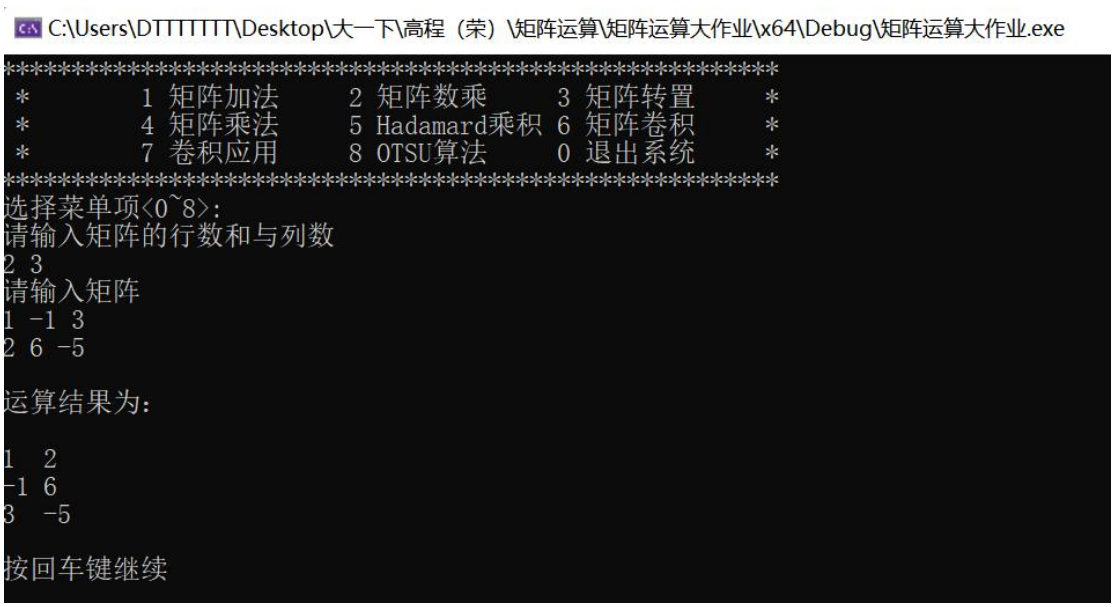
（2）矩阵数乘：

选择 2 后，按照输入提示依次输入第一个矩阵的行数、列数、完整矩阵和矩阵需要乘上的数后，会输出答案矩阵。（样例如下：）



（3）矩阵转置：

选择 3 后，按照输入提示依次输入矩阵的行数、列数和完整矩阵后，会输出转置后的矩阵。（样例如下：）



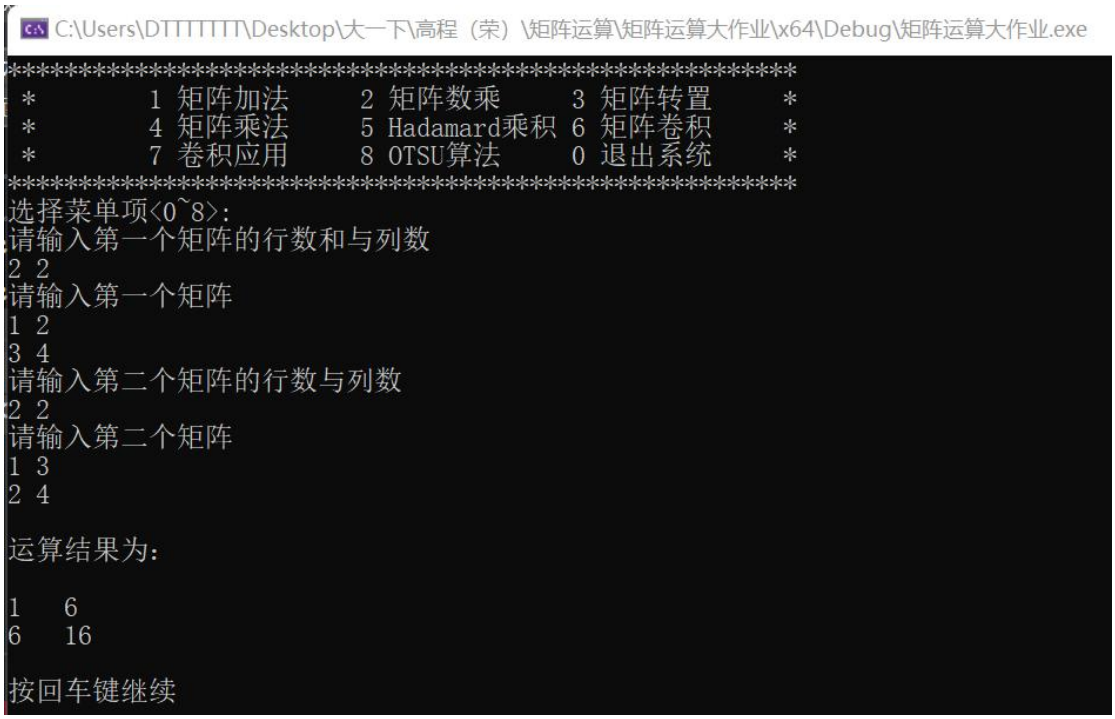
（4）矩阵乘法：

选择 4 后，按照输入提示依次输入第一个矩阵的行数、列数、完整矩阵和第二个矩阵的行数、列数、完整矩阵后，会输出答案矩阵。（样例如下：）

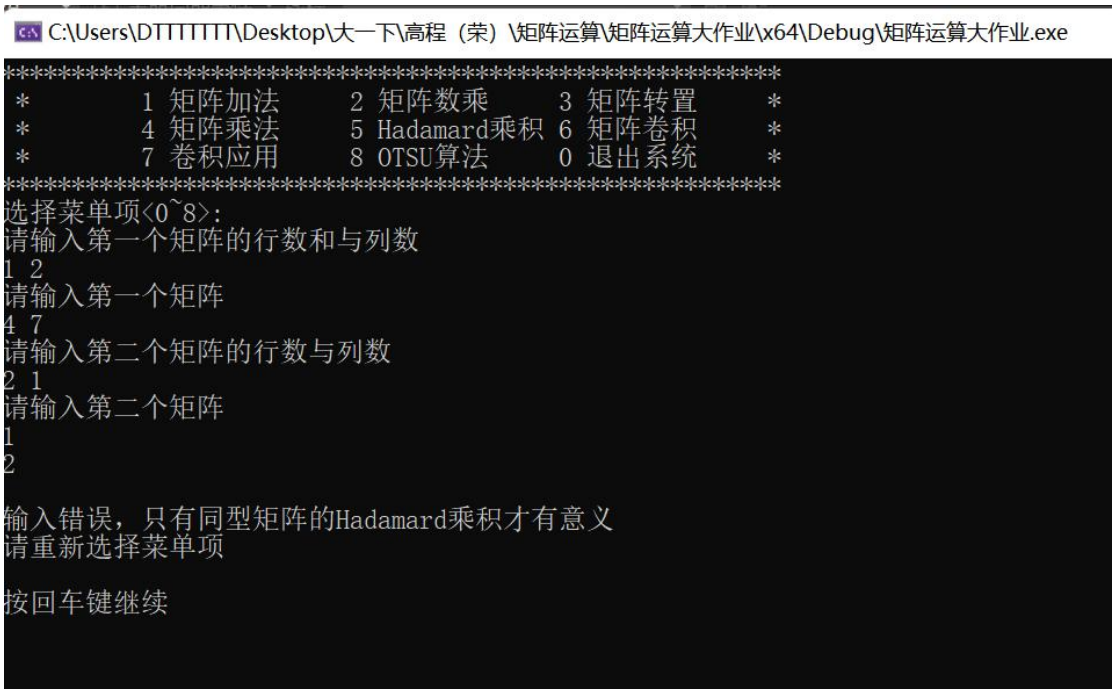
若输入不合法，会报错，刷新后重新选择菜单项：

(5) Hadamard 乘积：

选择 5 后，按照输入提示依次输入第一个矩阵的行数、列数、完整矩阵和第二个矩阵的行数、列数、完整矩阵后，会输出答案矩阵。（样例如下：）

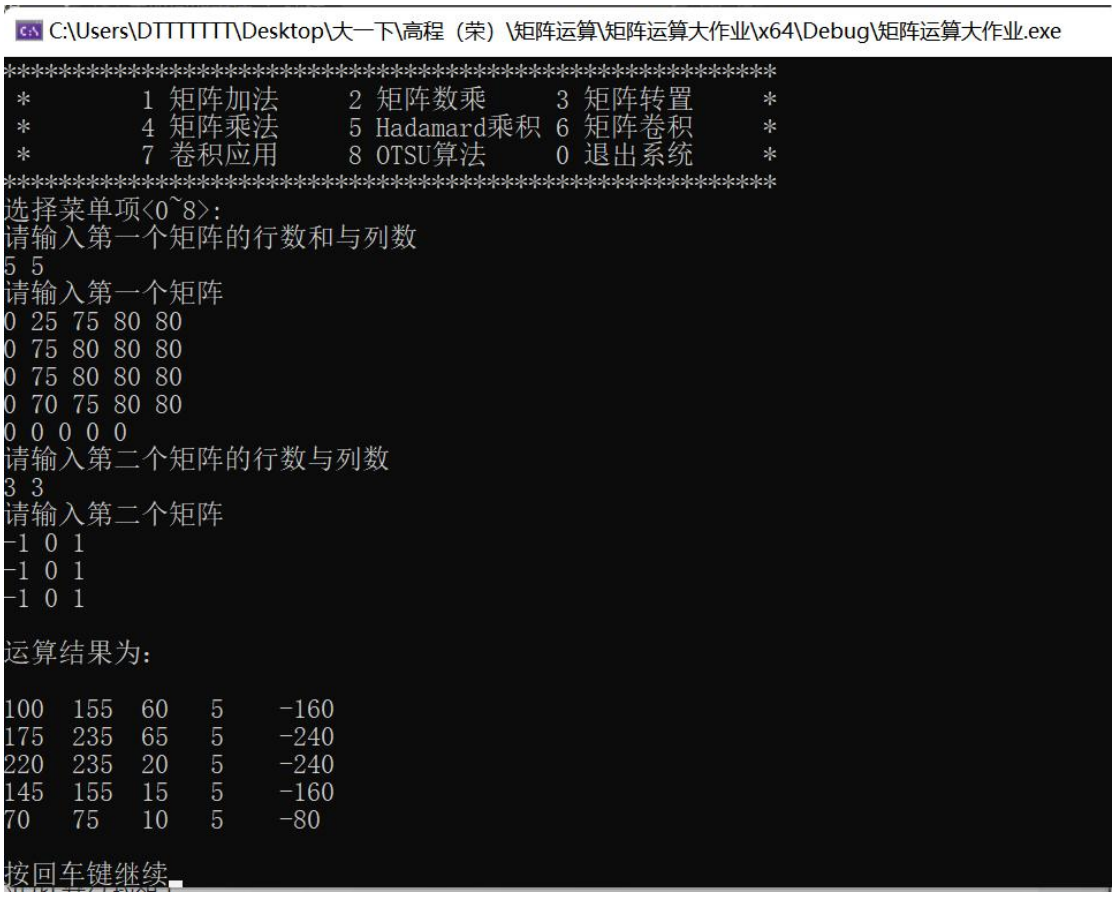


若输入不合法，会报错，刷新后重新选择菜单项：



(6) 矩阵卷积：

选择 6 后，按照输入提示依次输入第一个矩阵的行数、列数、完整矩阵和第二个矩阵的行数、列数、完整矩阵后，会输出答案矩阵。（样例如下：）



需要注意的是，此处计算的卷积，stride=1，padding=1，dilation=1，kernel size=3。

（7）卷积应用：

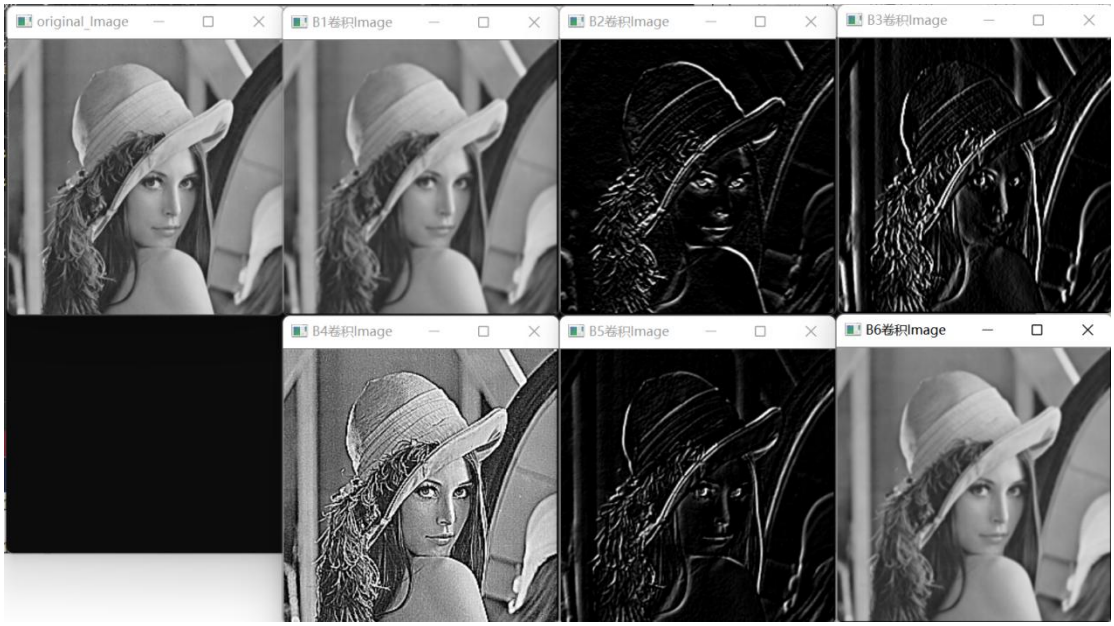
实现一个利用卷积操作应用与图像图例的简单示例。对 demolena.jpg 的灰度值矩阵对给出的六个卷积核进行卷积运算，并将得到的矩阵除以卷积核的总和（若卷积核总和为 0 则不做处理），保存为灰度图并观察结果。

给出的卷积核矩阵：

$$B1)\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \quad B2)\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \quad B3)\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

$$B4)\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix} \quad B5)\begin{bmatrix}-1 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix} \quad B6)\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

程序运行结果：



(8) OTSU 算法

使用 OTSU 算法对 lena 图像进行二值化，程序运行结果如下：

二. 在实验过程中遇到的困难及解决方法

1. 如何输出一个上下对齐的菜单或矩阵?

使用 setw()并且使用 **std::left** 使之左对齐。

2. 如何更加简便地修改矩阵输出格式?

将矩阵的输出写成一个单独的函数，每次需要输出矩阵的时候调用函数即可。若需修改矩阵输出的格式，也只需要在一个函数里修改一次就好，简便很多。

3. 如何将处理后的灰度值矩阵转成图像并显示?

将矩阵的值赋给原来的图像（Mat 类型），并显示即可。

```
for (int i = 0;i < len1;i++) //将卷积结果复制给Mat类型的m并返回
    for (int j = 0;j < wid1;j++) {
        m.at<uchar>(i,j) = m3[i][j];
    }
```

4. 在做 demo 的时候遇到的各种情况

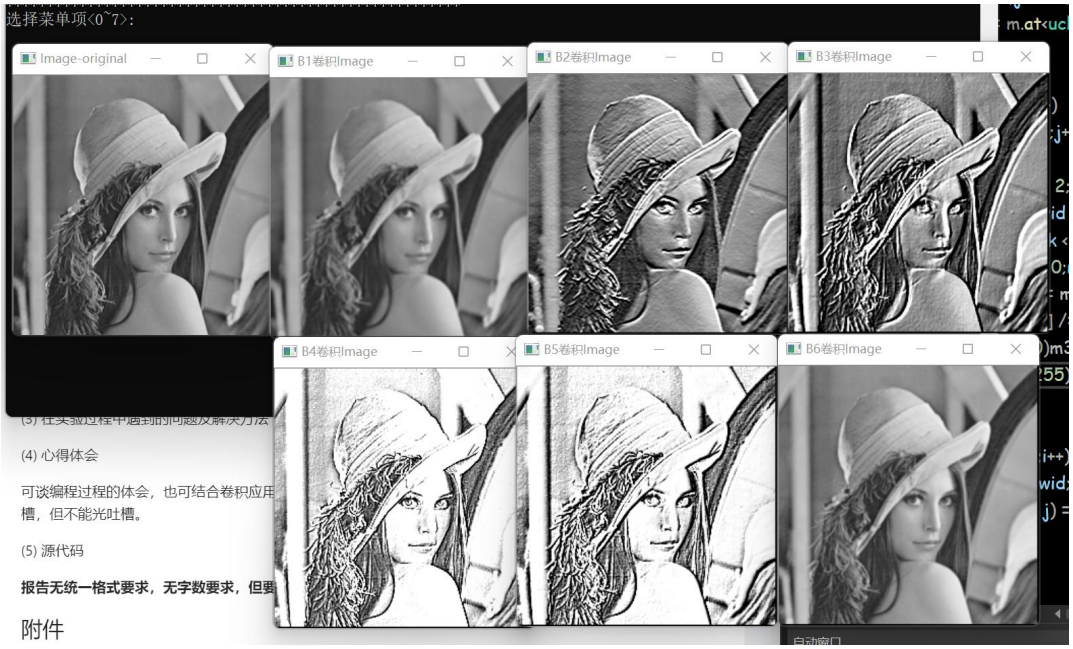
（1）

错误点及解决方法:

没有注意到灰度值的取值范围是[0,255]，对于卷积后的矩阵，应当对矩阵存储的灰度值进行判断，大于 255 的统一取成 255，小于 0 的统一取成 0。

```
if (m3[i][j] < 0)m3[i][j] = 0; //注意到灰度值的取值范围是0~255!
if (m3[i][j] > 255)m3[i][j] = 255;
```

(2)



错误点及解决方法:

没有对存储卷积结果的二维数组初始化，多次调用导致错误。在每次调用卷积函数的时候初始化该数组就可。

```
memset(m3, 0, sizeof(m3)); //由于多次调用此函数，需要初始化m3即答案矩阵
```

三．心得体会

其实总的来说，我个人感觉这次作业比图形时钟要简单一些，做得也顺畅许多，基本上是一气呵成，不想做图形时钟那么坎坷和一波三折，所以写图形时钟的报告里的困难和解决办法应该可以写出很多很多来（误）。

以下才是正文 hhh：

认真来说，最大的两个体会是：1．将重复多次的部分写成单独的函数，多次调用即可，不必要每次重复，冗长又不便捷（如例 1，2，3）；2．将一些多次使用的固定的值定义成常量，这样若想改变该值，只需要在定义中改变调试，不必要到每一个使用到它的地方修改（如例 4）。在这个作业里，具体如下：

例 1．将输入矩阵的部分写成单独的三个函数，以应对需要输入两个矩阵和只需要输入一个矩阵的两种情况：

```
void matri_input() {
    cout << "请输入矩阵的行数和与列数" << endl;
    cin >> len1 >> wid1;
    cout << "请输入矩阵" << endl;
    for (int i = 1;i <= len1;i++)
        for (int j = 1;j <= wid1;j++)
            cin >> m1[i][j];
}
void matri1_input() {
    cout << "请输入第一个矩阵的行数和与列数" << endl;
    cin >> len1 >> wid1;
    cout << "请输入第一个矩阵" << endl;
    for (int i = 1;i <= len1;i++)
        for (int j = 1;j <= wid1;j++)
            cin >> m1[i][j];
}
void matri2_input() {
    cout << "请输入第二个矩阵的行数与列数" << endl;
    cin >> len2 >> wid2;
    cout << "请输入第二个矩阵" << endl;
    for (int i = 1;i <= len2;i++)
        for (int j = 1;j <= wid2;j++)
            cin >> m2[i][j];
}
```

例 2. 将输出矩阵的部分写成单独的函数，带上输出矩阵的行数、列数和数组的起始下标（0 or 1），足以应对此作业中所有输出矩阵的需要：

```
void matri_output(int start, int len, int wid) { //输出答案矩阵

    int Width = 1; //获取矩阵输出时 合适的 每一个数字所占宽度
    for (int i = start;i <= len;i++)
        for (int j = start;j <= wid;j++)
            while ((m3[i][j] / (int)pow(10, Width)) != 0)Width++;
    Width += 2;

    cout << endl << "运算结果为：" << endl << endl;

    for (int i = start;i <= len;i++) {
        for (int j = start;j <= wid;j++)
            cout << setw(Width) << m3[i][j];
        cout << endl;
    }

    wait_for_enter();

    return;
}
```

例 3. 将 demo 里需要使用的对图像灰度值矩阵进行卷积的操作写成单独的函数，可以多次调用，不必重复书写：

```
Mat demo_conv(Mat m, int B[3][3]) {

    cvtColor(m, m, CV_BGR2GRAY); //将待处理图形转为灰度图
    len1 = m.rows, wid1 = m.cols;//得到待处理图形的行数和列数

    for (int i = 0;i < len1;i++) //将Mat转化成二维数组以进行运算
        for (int j = 0;j < wid1;j++) {
            m1[i][j] = m.at<uchar>(i, j);
        }

    int sum=0; //计算卷积核的总和sum
    for (int i = 0;i < 3;i++)
        for (int j = 0;j < 3;j++)
            sum += B[i][j];

    memset(m3, 0, sizeof(m3)); //由于多次调用此函数，需要初始化m3即答案矩阵

    //开始卷（积）啦
    for (int i = 0;i < len1 - 2;i++) { //枚举卷积答案矩阵的坐标
        for (int j = 0;j < wid1 - 2;j++) {

            for (int lk = 0;lk < 3;lk++) //计算该坐标的值
                for (int rk = 0;rk < 3;rk++)
                    m3[i][j] += m1[i + lk][j + rk] * B[lk][rk];

            if (sum)m3[i][j] /= sum;

            if (m3[i][j] < 0)m3[i][j] = 0; //注意到灰度值的取值范围是0~255！！！
            if (m3[i][j] > 255)m3[i][j] = 255;


        }
    }

    for (int i = 0;i < len1;i++) //将卷积结果复制给Mat类型的m并返回
        for (int j = 0;j < wid1;j++) {
            m.at<uchar>(i,j) = m3[i][j];
        }

    return m;
}
```

例 4. 将菜单输出宽度设置成常量，在调试合适的输出宽度时，只需要改变一个地方的值，不必要在每次输出的地方改变。

```
void menu() {

    const int width = 15;

    for (int i = 1;i <= width*4-4;i++)cout << '*';
    cout << endl;

    cout << " *";
    for (int i = 1;i <= width - 7;i++)cout << ' ';
    cout << std::left << setw(width) << "1 矩阵加法";
    cout << std::left << setw(width) << "2 矩阵数乘";
    cout << std::left << setw(width) << "3 矩阵转置";
    cout << '*' << endl;
```

另外就是觉得对图像的灰度值矩阵进行卷积运算可以得到各种不同处理的图像真的很奇妙啊！！！

四. 源代码

```
#include<iostream>
#include<conio.h>
#include<iomanip>
#include<cstring>
#include<cmath>
#include<opencv2/opencv.hpp>

using namespace cv;
using namespace std;

const int N = 300;
int len1, wid1, len2, wid2;
int m1[N][N], m2[N][N], m3[N][N];

void wait_for_enter(){
 cout << endl << "按回车键继续";
 while (_getch() != '\r');
 cout << endl << endl;
 return;
}

void menu() {
```

```
23.
24.     const int width = 15;
25.
26.     for (int i = 1;i <= width*4-4;i++)cout << '*';
27.     cout << endl;
28.
29.     cout << " *";
30.     for (int i = 1;i <= width - 7;i++)cout << ' ';
31.     cout << std::left << setw(width) << "1 矩阵加法";
32.     cout << std::left << setw(width) << "2 矩阵数乘";
33.     cout << std::left << setw(width) << "3 矩阵转置";
34.     cout << '*' << endl;
35.
36.     cout << " *";
37.     for (int i = 1;i <= width - 7;i++)cout << ' ';
38.     cout << std::left << setw(width) << "4 矩阵乘法";
39.     cout << std::left << setw(width) << "5 Hadamard 乘积";
40.     cout << std::left << setw(width) << "6 矩阵卷积";
41.     cout << '*' << endl;
42.
43.     cout << " *";
44.     for (int i = 1;i <= width - 7;i++)cout << ' ';
45.     cout << std::left << setw(width) << "7 卷积应用";
46.     cout << std::left << setw(width) << "8 OTSU 算法";
47.     cout << std::left << setw(width) << "0 退出系统";
```

```
48.     cout << '*' << endl;
49.
50.     for (int i = 1;i <= width*4-4;i++)cout << '*';
51.     cout << endl << "选择菜单项<0~8>:" << endl;
52.   }
53.
54.   void matri_input() {
55.     cout << "请输入矩阵的行数和与列数" << endl;
56.     cin >> len1 >> wid1;
57.     cout << "请输入矩阵" << endl;
58.     for (int i = 1;i <= len1;i++)
59.       for (int j = 1;j <= wid1;j++)
60.         cin >> m1[i][j];
61.   }
62.   void matri1_input() {
63.     cout << "请输入第一个矩阵的行数和与列数" << endl;
64.     cin >> len1 >> wid1;
65.     cout << "请输入第一个矩阵" << endl;
66.     for (int i = 1;i <= len1;i++)
67.       for (int j = 1;j <= wid1;j++)
68.         cin >> m1[i][j];
69.   }
70.   void matri2_input() {
```

```
71.     cout << "请输入第二个矩阵的行数与列数" << endl;
72.     cin >> len2 >> wid2;
73.     cout << "请输入第二个矩阵" << endl;
74.     for (int i = 1;i <= len2;i++)
75.      for (int j = 1;j <= wid2;j++)
76.       cin >> m2[i][j];
77.    }
78.
79.    void matri_output(int start, int len, int wid) { //输出答案矩阵
80.
81.     int Width = 1; //获取矩阵输出时 合适的 每一个数字所占宽度
82.     for (int i = start;i <= len;i++)
83.      for (int j = start;j <= wid;j++)
84.       while ((m3[i][j] / (int)pow(10, Width)) != 0)Width++;
85.     Width += 2;
86.
87.     cout << endl << "运算结果为：" << endl << endl;
88.
89.     for (int i = start;i <= len;i++) {
90.      for (int j = start;j <= wid;j++)
91.       cout << setw(Width) << m3[i][j];
92.      cout << endl;
93.     }
94.
95.     wait_for_enter();
```

```
96.
97.     return;
98.   }
99.
100.  void matriplus() { //1：矩阵加法
101.
102.   matri1_input(); //输入两个矩阵
103.   matri2_input();
104.
105.   if (len1 != len2 || wid1 != wid2) { //若输入不合法
106.    cout << endl << "输入错误，只有同型矩阵的加法运算有意义" << endl;
107.    cout << "请重新选择菜单项" << endl;
108.    wait_for_enter();
109.    return;
110.  }
111.
112.  for (int i = 1;i <= len1;i++)
113.   for (int j = 1;j <= wid1;j++)
114.    m3[i][j] = m1[i][j] + m2[i][j];
115.
116.  matri_output(1, len1, wid1);
117.
118.  return;
119.  }
120.
121.  void nummulti() { //2：矩阵数乘
```

```
122.
123.     matri_input(); //输入矩阵
124.
125.     int num; //输入数字
126.     cout << "请输入矩阵需乘上的数字" << endl;
127.     cin >> num;
128.
129.     for (int i = 1;i <= len1;i++)
130.      for (int j = 1;j <= wid1;j++)
131.       m3[i][j] = m1[i][j] * num;
132.
133.     matri_output(1, len1, wid1);
134.
135.     return;
136. }
137.
138. void matritrans() { //3：矩阵转置
139.
140.     matri_input();
141.
142.     for (int i = 1;i <= wid1;i++)
143.      for (int j = 1;j <= len1;j++)
144.       m3[i][j] = m1[j][i];
145.
146.     matri_output(1, wid1, len1);
147.
148.     return;
```

```
149.  }
150.  void matrimulti() { //4：矩阵乘法
151.
152.   matri1_input();
153.   matri2_input();
154.
155.   if (len2 != wid1) {
156.    cout << endl << "输入错误，第一个矩阵的列数与第二个矩阵的行数相等运算才有意义" << endl;
157.    cout << "请重新选择菜单项" << endl;
158.    wait_for_enter();
159.    return;
160.  }
161.
162.   for (int i = 1;i <= len1;i++)
163.    for (int j = 1;j <= wid2;j++)
164.     for (int k = 1;k <= len2;k++)
165.      m3[i][j] += m1[i][k] * m2[k][j];
166.
167.   matri_output(1, len1, wid2);
168.
169.   return;
170.  }
171.
172.  void hadamulti() { //5：Hadamard 乘积
173.
174.   matri1_input();
```

```
175.     matri2_input();
176.
177.     if (len1 != len2 || wid1 != wid2) {
178.         cout << endl << "输入错误，只有同型矩阵的 Hadamard 乘积才有意义" << endl;
179.         cout << "请重新选择菜单项" << endl;
180.         wait_for_enter();
181.         return;
182.     }
183.
184.     for (int i = 1;i <= len1;i++)
185.         for (int j = 1;j <= wid1;j++)
186.             m3[i][j] = m1[i][j] * m2[i][j];
187.
188.     matri_output(1, len1, wid1);
189.
190.     return;
191. }
192.
193. void conv() { //6：矩阵卷积
194.
195.     matri1_input();
196.     matri2_input();
197.
198.     len1++, wid1++; //由于 padding(填补)=1;
199.
200.     if (len2 != 3 || wid2 != 3) {
```

```
201.     cout << endl << "输入错误，请输入 3*3 的 kernel 矩阵" << endl;
202.     cout << "请重新选择菜单项" << endl;
203.     wait_for_enter();
204.     return;
205.  }
206.
207.  for (int i = 0;i <= len1 - 2;i++)//枚举卷积结果矩阵的坐标
208.     for (int j = 0;j <= wid1 - 2;j++) {
209.        for (int lk = 0;lk < 3;lk++)//计算该坐标的值
210.           for (int rk = 0;rk < 3;rk++)
211.              m3[i][j] += m1[i + lk][j + rk] * m2[lk + 1][rk + 1];
212.     }
213.
214.  matri_output(0, len1 - 2, wid1 - 2);
215.
216.  return;
217.  }
218.
219.  Mat demo_conv(Mat m, int B[3][3]) {
220.
221.     cvtColor(m, m, CV_BGR2GRAY); //将待处理图形转为灰度图
222.     len1 = m.rows, wid1 = m.cols;//得到待处理图形的行数和列数
223.
224.     for (int i = 0;i < len1;i++) //将 Mat 转化成二维数组以进行运算
225.        for (int j = 0;j < wid1;j++) {
```

```
226.      m1[i][j] = m.at<uchar>(i, j);
227.    }
228.
229.  int sum=0; //计算卷积核的总和 sum
230.  for (int i = 0;i < 3;i++)
231.    for (int j = 0;j < 3;j++)
232.      sum += B[i][j];
233.
234.  memset(m3, 0, sizeof(m3)); //由于多次调用此函数，需要初始化 m3 即答案矩阵
235.
236.  //开始卷（积）啦
237.  for (int i = 0;i < len1 - 2;i++) { //枚举卷积答案矩阵的坐标
238.    for (int j = 0;j < wid1 - 2;j++) {
239.
240.      for (int lk = 0;lk < 3;lk++) //计算该坐标的值
241.        for (int rk = 0;rk < 3;rk++)
242.          m3[i][j] += m1[i + lk][j + rk] * B[lk][rk];
243.
244.      if (sum)m3[i][j] /= sum;
245.
246.      if (m3[i][j] < 0)m3[i][j] = 0; //注意到灰度值的取值范围是 0~255！！！
247.      if (m3[i][j] > 255)m3[i][j] = 255;
248.
249.    }
250.  }
```

```
251.
252.    for (int i = 0;i < len1;i++) //将卷积结果复制给 Mat 类型的 m 并返回
253.     for (int j = 0;j < wid1;j++) {
254.      m.at<uchar>(i,j) = m3[i][j];
255.     }
256.
257.    return m;
258.  }
259.
260.  void demo() { //7：卷积应用
261.
262.    Mat image = imread("demolena.jpg"); //获取 lena 原图
263.
264.    int B1[3][3] = { {1,1,1},{1,1,1},{1,1,1} }; //定义卷积核们
265.    int B2[3][3] = { {-1,-2,-1},{0,0,0},{1,2,1} };
266.    int B3[3][3] = { {-1,0,1},{-2,0,2},{-1,0,1} };
267.    int B4[3][3] = { {-1,-1,-1},{-1,9,-1},{-1,-1,-1} };
268.    int B5[3][3] = { {-1,-1,0},{-1,0,1},{0,1,1} };
269.    int B6[3][3] = { {1,2,1},{2,4,2},{1,2,1} };
270.
271.    Mat image1=demo_conv(image, B1); //调用函数得到不同卷积处理后的 lena
272.    Mat image2 = demo_conv(image, B2);
273.    Mat image3 = demo_conv(image, B3);
274.    Mat image4 = demo_conv(image, B4);
275.    Mat image5 = demo_conv(image, B5);
```

```
276.    Mat image6 = demo_conv(image, B6);
277.
278.    imshow("original_Image", image);//显示所有的 lena!
279.    imshow("B1 卷积 Image", image1);
280.    imshow("B2 卷积 Image", image2);
281.    imshow("B3 卷积 Image", image3);
282.    imshow("B4 卷积 Image", image4);
283.    imshow("B5 卷积 Image", image5);
284.    imshow("B6 卷积 Image", image6);
285.    waitKey(0);
286.
287.    wait_for_enter();
288.
289.    return;
290.  }
291.
292.  void otsu() { //8:OTSU 算法
293.    Mat origin_image = imread("demolena.jpg"); //获取 lena 原图
294.    Mat result_image = origin_image;
295.
296.    cvtColor(origin_image, origin_image, CV_BGR2GRAY); //将待处理图形转为灰度图
297.    cvtColor(result_image, result_image, CV_BGR2GRAY);
298.
299.    len1 = origin_image.rows, wid1 = origin_image.cols;//得到待处理图形的行数和列数
300.
```

```
int maxG = 0, ansT;

for (int T = 0;T <= 255;T++) { //用 otsu 算法找到阈值!

int N0 = 0, N1 = 0;

double w0=0, w1=0, u0=0, u1=0;

for (int i = 0;i < len1;i++)

 for (int j = 0;j < wid1;j++) {

  int val = origin_image.at<uchar>(i, j);

  if (val < T)

   N0++, u0 += (double)val;

  else

   N1++, u1 += (double)val;

 }

w0 = 1.0 * N0 / (N0 + N1), w1 = 1.0 * N1 / (N0 + N1);

u0 /= N0, u1 /= N1;

int G = w0 * w1 * pow(u0 – u1, 2);

if (G > maxG)maxG = G, ansT = T;

}

for (int i = 0;i < len1;i++)

 for (int j = 0;j < wid1;j++){
```

```
326.     int val = origin_image.at<uchar>(i, j);
327.     if(val>ansT) result_image.at<uchar>(i, j)=255;
328.     else result_image.at<uchar>(i, j)=0;
329.   }
330.
331.  imshow("Origin", origin_image);
332.  imshow("Result", result_image);
333.  waitKey(0);
334.
335.  wait_for_enter();
336.
337.  return;
338. }
339.
340. int main() {
341.
342.  wait_for_enter();
343.
344.  while (1) {
345.
346.   system("cls");//清屏并显示菜单
347.   menu();
348.
349.   char choice = _getch();//输入选择的菜单项
350.
351.   if (choice == '0') {//退出系统
352.    cout << endl << "确定退出吗？退出请输入 y，继续请输入 n" << endl;
```

```
353.     char ch;
354.     cin >> ch;
355.     if (ch == 'y')break;
356.     else continue;
357.   }
358.
359.   memset(m1, 0, sizeof(m1));//初始化矩阵数组!
360.   memset(m2, 0, sizeof(m2));
361.   memset(m3, 0, sizeof(m3));
362.
363.   switch (choice) {
364.   case'1':
365.     matriplus(); //1：矩阵加法
366.     break;
367.   case'2':
368.     nummulti(); //2：矩阵数乘
369.     break;
370.   case'3':
371.     matritrans(); //3：矩阵转置
372.     break;
373.   case'4':
374.     matrimulti(); //4：矩阵乘法
375.     break;
```

```
376.     case'5':
377.         hadamulti(); //5：Hadamard 乘积
378.         break;
379.     case'6':
380.         conv(); //6：矩阵卷积
381.         break;
382.     case'7':
383.         demo(); //7：卷积应用
384.         break;
385.     case'8':
386.         otsu();
387.         break;
388.     default:
389.         cout << endl << "输入错误，请重新输入" << endl;
390.         wait_for_enter();
391.     }
392. }
393.
394. return 0;
395. }
```