**报告名称：矩阵运算**

**班级：国06**

**学号：2252075**

**姓名：刘文飞**

**完成日期：2023.5.21**

1. **设计思路与功能描述：**
2. **设计思路：本题目旨在完成八个小项目，前6个是矩阵的一些基础运算，而78项是通过矩阵卷积和ostu算法进行图像的处理**
3. **功能描述：**
4. 前五个功能不加赘述，是矩阵的运算，不复杂也没有处理难度
5. 第六七项涉及到矩阵卷积运算，要了解卷积运算规则以及如何进行二值化处理图像
6. 第八项是ostu算法对图像进行二值化
7. **遇到的问题和解决方法**
8. **卷积算法的处理，需要耐心注意每一个细节**
9. **OSTU**算法要自己进行资料查询与学习
10. **如何对图像进行提取和处理，要遍历每一个像素，根据不同的灰度值进行运算分析处理，自己要查找一些资料，学会初步处理图像**
11. **心得体会**
12. **本次的大作业相对比较简单，工作量不大，前五项工作没什么难点，就是初步的矩阵数字运算，第六项的卷积运算要学会运算规则，并且转换成代码，七八项功能涉及到图像的处理，需要自己学习一些图像处理的算法与知识**
13. **主要的工作量在于学习卷积运算以及图像处理的知识，代码量并不大，整个大作业完成度也比较高**
14. **源代码**
15. #include <conio.h>
16. #include <iostream>
17. #include <opencv2/opencv.hpp>
18. using namespace cv;
19. using namespace std;
20. // 此框架若有不完美可以在作业中任意修改
21. void wait\_for\_enter()
22. {
23. cout << endl
24. << "按回车键继续";
25. while (\_getch() != '\r')
26. ;
27. cout << endl
28. << endl;
29. }
30. void application(Mat& ima,int i);
31. void demo()
32. {
33. system("cls"); // 清屏函数
35. Mat image[6];
36. for (int i = 0; i < 6; i++)
37. image[i] = imread("demolena.jpg");// 图像的灰度值存放在格式为Mat的变量image中
39. for (int i = 0; i < 6; i++)
40. application(image[i], i);
42. imshow("Image-result\_0", image[0]);
43. imshow("Image-result\_1", image[1]);
44. imshow("Image-result\_2", image[2]);
45. imshow("Image-result\_3", image[3]);
46. imshow("Image-result\_4", image[4]);
47. imshow("Image-result\_5", image[5]);
48. waitKey(0);
49. // 提示：Mat格式可与数组相互转换
50. wait\_for\_enter();
52. return;
53. }
54. void menu(char\*p);
55. void matriplus();
56. void nummulti();
57. void matritrans();
58. void matrimulti();
59. void hadamulti();
60. void conv();
61. void application\_(Mat& ima);
62. void otsu() {
63. system("cls"); // 清屏函数
64. Mat image[4];
66. image[0]=imread("snowball.jpg");// 图像的灰度值存放在格式为Mat的变量image中
67. image[1] = imread("polyhedrosis.jpg");
68. image[2] = imread("ship.jpg");
69. image[3] = imread("brain.jpg ");
71. for (int i = 0; i < 4; i++)
72. application\_(image[i]);
73. imshow("Image-result\_0", image[0]);
74. imshow("Image-result\_1", image[1]);
75. imshow("Image-result\_2", image[2]);
76. imshow("Image-result\_3", image[3]);
77. waitKey(0);
78. // 提示：Mat格式可与数组相互转换
79. wait\_for\_enter();
80. }
81. int main()
82. {
83. // 定义相关变量
84. char choice = '0';
85. char ch;
86. wait\_for\_enter();
87. while (true) // 注意该循环退出的条件
88. {
89. system("cls"); // 清屏函数
90. menu(&choice); // 调用菜单显示函数，自行补充完成
91. // 按要求输入菜单选择项choice
92. if (choice == '0') // 选择退出
93. {
94. cout << "\n 确定退出吗?" << endl;
95. cin >> ch;
96. if (ch == 'y' || ch == 'Y')
97. break;
98. else
99. continue;
100. }
101. switch (choice)
102. {
103. // 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），若选择加分项，请自行补充
104. case '1':
105. matriplus();
106. break;
107. case '2':
108. nummulti();
109. break;
110. case '3':
111. matritrans();
112. break;
113. case '4':
114. matrimulti();
115. break;
116. case '5':
117. hadamulti();
118. break;
119. case '6':
120. conv();
121. break;
122. case '7':
123. demo();
124. break;
125. case '8':
126. otsu();
127. break;
128. default:
129. cout << "\n 输入错误，请从新输入" << endl;
130. wait\_for\_enter();
131. }
132. }
133. return 0;
134. }
135. #include <conio.h>
136. #include <iostream>
137. // #include <opencv2/opencv.hpp>
138. // using namespace cv;
139. using namespace std;
140. void wait\_for\_enter();
141. void convolve(int\*\* input, int\*\* output, int\*\* kernel, int rows, int cols, int kernel\_size = 3, int padding = 1, int stride = 1) {
142. int output\_rows = (rows - kernel\_size + 2 \* padding) / stride + 1;
143. int output\_cols = (cols - kernel\_size + 2 \* padding) / stride + 1;
145. int\*\* padded\_input = new int\* [rows + 2 \* padding]();
146. for (int i = 0; i < rows + 2 \* padding; i++) {
147. padded\_input[i] = new int[cols + 2 \* padding]();
148. }
149. for (int i = 0; i < rows; i++) {
150. for (int j = 0; j < cols; j++) {
151. padded\_input[i + padding][j + padding] = input[i][j];
152. }
153. }
154. for (int i = 0; i < output\_rows; i++) {
155. for (int j = 0; j < output\_cols; j++) {
156. int sum = 0;
157. for (int m = 0; m < kernel\_size; m++) {
158. for (int n = 0; n < kernel\_size; n++) {
159. sum += padded\_input[i \* stride + m][j \* stride + n] \* kernel[m][n];
160. }
161. }
162. output[i][j] = sum;
163. }
164. }
165. for (int i = 0; i < rows + 2 \* padding; i++) {
166. delete[] padded\_input[i];
167. }
168. delete[] padded\_input;
169. }
170. void matriplus() {
171. system("cls"); // 清屏函数
172. int\*\* matrix1, \*\* matrix2, \*\* sum;
173. int row1, col1, row2, col2;
174. cout << "请输入矩阵1的行和列数：";
175. cin >> row1 >> col1;
176. matrix1 = new int\* [row1];
177. for (int i = 0; i < row1; i++) {
178. matrix1[i] = new int[col1];
179. }
180. cout << "请输入矩阵1的元素（以空格分隔）：" << endl;
181. for (int i = 0; i < row1; i++) {
182. for (int j = 0; j < col1; j++) {
183. cin >> matrix1[i][j];
184. }
185. }
186. cout << "请输入矩阵2的行和列数：";
187. cin >> row2 >> col2;
188. matrix2 = new int\* [row2];
189. for (int i = 0; i < row2; i++) {
190. matrix2[i] = new int[col2];
191. }
192. cout << "请输入矩阵2的元素（以空格分隔）：" << endl;
193. for (int i = 0; i < row2; i++) {
194. for (int j = 0; j < col2; j++) {
195. cin >> matrix2[i][j];
196. }
197. }
198. if (row1 == row2 && col1 == col2) {
199. sum = new int\* [row1];
200. for (int i = 0; i < row1; i++) {
201. sum[i] = new int[col1];
202. for (int j = 0; j < col1; j++) {
203. sum[i][j] = matrix1[i][j] + matrix2[i][j];
204. }
205. }
206. cout << "矩阵1加矩阵2的结果是：" << endl;
207. for (int i = 0; i < row1; i++) {
208. for (int j = 0; j < col1; j++) {
209. cout << sum[i][j] << " ";
210. }
211. cout << endl;
212. }
213. for (int i = 0; i < row1; i++) {
214. delete[] sum[i];
215. }
216. delete[] sum;
217. }
218. else {
219. cout << "矩阵1和矩阵2无法相加！" << endl;
220. }
221. for (int i = 0; i < row1; i++) {
222. delete[] matrix1[i];
223. }
224. delete[] matrix1;
225. for (int i = 0; i < row2; i++) {
226. delete[] matrix2[i];
227. }
228. delete[] matrix2;
229. wait\_for\_enter();
230. }
231. void nummulti() {
232. system("cls"); // 清屏函数
233. int row1, col1, scalar;
234. cout << "请输入矩阵的行和列数：" << endl;
235. cin >> row1 >> col1 >> scalar;
236. int\*\* matrix = new int\* [row1];
237. for (int i = 0; i < row1; i++) {
238. matrix[i] = new int[col1];
239. }
240. cout << "请输入矩阵的元素（以空格分隔）：" << endl;
241. for (int i = 0; i < row1; i++) {
242. for (int j = 0; j < col1; j++) {
243. cin >> matrix[i][j];
244. }
245. }
246. for (int i = 0; i < row1; i++) {
247. for (int j = 0; j < col1; j++) {
248. matrix[i][j] = matrix[i][j] \* scalar;
249. }
250. }
251. cout << "数乘运算后矩阵为 ：" << endl;
252. for (int i = 0; i < row1; i++) {
253. for (int j = 0; j < col1; j++) {
254. cout << matrix[i][j] << " ";
255. }
256. cout << endl;
257. }
258. for (int i = 0; i < row1; i++) {
259. delete[] matrix[i];
260. }
261. delete[] matrix;
262. wait\_for\_enter();
263. }
264. void matritrans() {
265. system("cls"); // 清屏函数
266. int\*\* matrix, \*\* transMatrix;
267. int rows, cols;
269. cout << "请输入矩阵的行和列数：";
270. cin >> rows >> cols;
271. matrix = new int\* [rows];
272. for (int i = 0; i < rows; i++) {
273. matrix[i] = new int[cols];
274. }
276. cout << "请输入矩阵的元素（以空格分隔）：" << endl;
277. for (int i = 0; i < rows; i++) {
278. for (int j = 0; j < cols; j++) {
279. cin >> matrix[i][j];
280. }
281. }
283. transMatrix = new int\* [cols];
284. for (int i = 0; i < cols; i++) {
285. transMatrix[i] = new int[rows];
286. for (int j = 0; j < rows; j++) {
287. transMatrix[i][j] = 0;
288. }
289. }
291. for (int i = 0; i < cols; i++) {
292. for (int j = 0; j < rows; j++) {
293. transMatrix[i][j] = matrix[j][i];
294. }
295. }
297. cout << "转置后的矩阵是：" << endl;
298. for (int i = 0; i < cols; i++) {
299. for (int j = 0; j < rows; j++) {
300. cout << transMatrix[i][j] << " ";
301. }
302. cout << endl;
303. }
305. for (int i = 0; i < rows; i++) {
306. delete[] matrix[i];
307. }
308. delete[] matrix;
309. for (int i = 0; i < cols; i++) {
310. delete[] transMatrix[i];
311. }
312. delete[] transMatrix;
313. wait\_for\_enter();
314. }
315. void matrimulti() {
316. system("cls"); // 清屏函数
317. int\*\* matrix1, \*\* matrix2, \*\* product;
318. int row1, col1, row2, col2;
320. cout << "请输入矩阵1的行和列数：";
321. cin >> row1 >> col1;
322. matrix1 = new int\* [row1];
323. for (int i = 0; i < row1; i++) {
324. matrix1[i] = new int[col1];
325. }
326. cout << "请输入矩阵1的元素（以空格分隔）：" << endl;
327. for (int i = 0; i < row1; i++) {
328. for (int j = 0; j < col1; j++) {
329. cin >> matrix1[i][j];
330. }
331. }
332. cout << "请输入矩阵2的行和列数：";
333. cin >> row2 >> col2;
334. matrix2 = new int\* [row2];
335. for (int i = 0; i < row2; i++) {
336. matrix2[i] = new int[col2];
337. }
338. cout << "请输入矩阵2的元素（以空格分隔）：" << endl;
339. for (int i = 0; i < row2; i++) {
340. for (int j = 0; j < col2; j++) {
341. cin >> matrix2[i][j];
342. }
343. }
345. if (col1 == row2) {
346. product = new int\* [row1];
347. for (int i = 0; i < row1; i++) {
348. product[i] = new int[col2];
349. for (int j = 0; j < col2; j++) {
350. product[i][j] = 0;
351. for (int k = 0; k < col1; k++) {
352. product[i][j] += matrix1[i][k] \* matrix2[k][j];
353. }
354. }
355. }
356. cout << "矩阵1乘矩阵2的结果是：" << endl;
357. for (int i = 0; i < row1; i++) {
358. for (int j = 0; j < col2; j++) {
359. cout << product[i][j] << " ";
360. }
361. cout << endl;
362. }
363. for (int i = 0; i < row1; i++) {
364. delete[] product[i];
365. }
366. delete[] product;
367. }
368. else {
369. cout << "矩阵1和矩阵2无法相乘！" << endl;
370. }
372. for (int i = 0; i < row1; i++) {
373. delete[] matrix1[i];
374. }
375. delete[] matrix1;
376. for (int i = 0; i < row2; i++) {
377. delete[] matrix2[i];
378. }
379. delete[] matrix2;
380. wait\_for\_enter();
381. }
382. void hadamulti() {
383. system("cls"); // 清屏函数
384. int\*\* matrix1, \*\* matrix2, \*\* sum;
385. int row1, col1, row2, col2;
386. cout << "请输入矩阵1的行和列数：";
387. cin >> row1 >> col1;
388. matrix1 = new int\* [row1];
389. for (int i = 0; i < row1; i++) {
390. matrix1[i] = new int[col1];
391. }
392. cout << "请输入矩阵1的元素（以空格分隔）：" << endl;
393. for (int i = 0; i < row1; i++) {
394. for (int j = 0; j < col1; j++) {
395. cin >> matrix1[i][j];
396. }
397. }
398. cout << "请输入矩阵2的行和列数：";
399. cin >> row2 >> col2;
400. matrix2 = new int\* [row2];
401. for (int i = 0; i < row2; i++) {
402. matrix2[i] = new int[col2];
403. }
404. cout << "请输入矩阵2的元素（以空格分隔）：" << endl;
405. for (int i = 0; i < row2; i++) {
406. for (int j = 0; j < col2; j++) {
407. cin >> matrix2[i][j];
408. }
409. }
410. if (row1 == row2 && col1 == col2) {
411. sum = new int\* [row1];
412. for (int i = 0; i < row1; i++) {
413. sum[i] = new int[col1];
414. for (int j = 0; j < col1; j++) {
415. sum[i][j] = matrix1[i][j] \* matrix2[i][j];
416. }
417. }
418. cout << "矩阵1和矩阵2Hardmard乘积的结果是：" << endl;
419. for (int i = 0; i < row1; i++) {
420. for (int j = 0; j < col1; j++) {
421. cout << sum[i][j] << " ";
422. }
423. cout << endl;
424. }
425. for (int i = 0; i < row1; i++) {
426. delete[] sum[i];
427. }
428. delete[] sum;
429. }
430. else {
431. cout << "矩阵1和矩阵2无法做Hardmard乘积！" << endl;
432. }
433. for (int i = 0; i < row1; i++) {
434. delete[] matrix1[i];
435. }
436. delete[] matrix1;
437. for (int i = 0; i < row2; i++) {
438. delete[] matrix2[i];
439. }
440. delete[] matrix2;
441. wait\_for\_enter();
442. }
443. void conv() {
444. system("cls"); // 清屏函数
445. const int kernel\_size = 3;
446. const int padding = 1;
447. const int stride = 1;
448. int rows, cols;
449. cout << "请输入原矩阵的行和列数：";
450. cin >> rows >> cols;
451. int\*\* input = new int\* [rows]();
452. for (int i = 0; i < rows; i++) {
453. input[i] = new int[cols]();
454. }
455. cout << "请输入原矩阵的元素（以空格分隔）：" << endl;
456. for (int i = 0; i < rows; i++) {
457. for (int j = 0; j < cols; j++) {
458. cin >> input[i][j];
459. }
460. }
461. int\*\* kernel = new int\* [kernel\_size]();
462. for (int i = 0; i < kernel\_size; i++) {
463. kernel[i] = new int[kernel\_size]();
464. }
465. cout << "请输入卷积核的元素（以空格分隔）：" << endl;
466. for (int i = 0; i < kernel\_size; i++) {
467. for (int j = 0; j < kernel\_size; j++) {
468. cin >> kernel[i][j];
469. }
470. }
471. int output\_rows = rows + 2 \* padding - kernel\_size + 1;
472. int output\_cols = cols + 2 \* padding - kernel\_size + 1;
473. int\*\* output = new int\* [output\_rows]();
474. for (int i = 0; i < output\_rows; i++) {
475. output[i] = new int[output\_cols]();
476. }
477. convolve(input, output, kernel, rows, cols);
478. cout << "卷积所得矩阵为 ：" << endl;
479. for (int i = 0; i < output\_rows; i++) {
480. for (int j = 0; j < output\_cols; j++) {
481. cout << output[i][j] << " ";
482. }
483. cout << endl;
484. }
485. for (int i = 0; i < rows; i++) {
486. delete[] input[i];
487. }
488. delete[] input;
489. for (int i = 0; i < kernel\_size; i++) {
490. delete[] kernel[i];
491. }
492. delete[] kernel;
493. for (int i = 0; i < output\_rows; i++) {
494. delete[] output[i];
495. }
496. delete[] output;
497. wait\_for\_enter();
498. }

#include <conio.h>

#include <iostream>

#include <opencv2/opencv.hpp>

using namespace cv;

using namespace std;

void convolve\_(int\*\* input, int\*\* output, int\*\* kernel, int rows, int cols, int kernel\_size = 3, int padding = 1, int stride = 1) {

int output\_rows = (rows - kernel\_size + 2 \* padding) / stride + 1;

int output\_cols = (cols - kernel\_size + 2 \* padding) / stride + 1;

int kernel\_sum = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel\_sum += kernel[i][j];

}

}

if (kernel\_sum == 0)

kernel\_sum = 1;

int\*\* padded\_input = new int\* [rows + 2 \* padding]();

for (int i = 0; i < rows + 2 \* padding; i++) {

padded\_input[i] = new int[cols + 2 \* padding]();

}

for (int i = 0; i < rows; i++) {

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

padded\_input[i + padding][j + padding] = input[i][j];

}

}

for (int i = 0; i < output\_rows; i++) {

for (int j = 0; j < output\_cols; j++) {

int sum = 0;

for (int m = 0; m < kernel\_size; m++) {

for (int n = 0; n < kernel\_size; n++) {

sum += padded\_input[i \* stride + m][j \* stride + n] \* kernel[m][n];

}

}

output[i][j] = sum / kernel\_sum;

}

}

for (int i = 0; i < rows + 2 \* padding; i++) {

delete[] padded\_input[i];

}

delete[] padded\_input;

}

void application(Mat& ima, int i) {

const int kernel\_size = 3;

const int padding = 1;

const int stride = 1;

int kernel\_1[3][3] = { 1,1,1,1,1,1,1,1,1 };

int kernel\_2[3][3] = { -1,-2,-1,0,0,0,1,2,1 };

int kernel\_3[3][3] = { -1,0,1,-2,0,2,-1,0,1 };

int kernel\_4[3][3] = { -1 ,-1 ,-1,-1 ,9,-1,-1,-1,-1 };

int kernel\_5[3][3] = { -1,-1,0,-1,0,1,0,1,1 };

int kernel\_6[3][3] = { 1,2,1,2,4,2,1,2,1 };

int rows = ima.rows;

int cols = ima.cols;

int\*\* input = new int\* [rows]();

for (int i = 0; i < rows; i++) {

input[i] = new int[cols]();

}

for (int i = 0; i < rows; i++) {

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

input[i][j] = ima.at<Vec3b>(i, j)[0];/\*灰度图只需要读一个通道即可\*/

}

}

int\*\* kernel = new int\* [kernel\_size]();

for (int i = 0; i < kernel\_size; i++) {

kernel[i] = new int[kernel\_size]();

}

if (i == 1) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel[i][j] = kernel\_1[i][j];

}

}

}

else if (i == 2) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel[i][j] = kernel\_2[i][j];

}

}

}

else if (i == 3) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel[i][j] = kernel\_3[i][j];

}

}

}

else if (i == 4) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel[i][j] = kernel\_4[i][j];

}

}

}

else if (i == 5) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel[i][j] = kernel\_5[i][j];

}

}

}

else {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

kernel[i][j] = kernel\_6[i][j];

}

}

}

int output\_rows = rows;

int output\_cols = cols;

int\*\* output = new int\* [output\_rows]();

for (int i = 0; i < output\_rows; i++) {

output[i] = new int[output\_cols]();

}

convolve\_(input, output, kernel, rows, cols);

for (int i = 0; i < output\_rows; i++) {

for (int j = 0; j < output\_cols; j++) {

if (output[i][j] > 255)

output[i][j] = 255;

if (output[i][j] < 0)

output[i][j] = 0;

ima.at<Vec3b>(i, j)[0] = output[i][j];

ima.at<Vec3b>(i, j)[1] = output[i][j];

ima.at<Vec3b>(i, j)[2] = output[i][j];

}

}

for (int i = 0; i < rows; i++) {

delete[] input[i];

}

delete[] input;

for (int i = 0; i < kernel\_size; i++) {

delete[] kernel[i];

}

delete[] kernel;

for (int i = 0; i < output\_rows; i++) {

delete[] output[i];

}

delete[] output;

1. }

#include <conio.h>

#include <iostream>

#include <opencv2/opencv.hpp>

using namespace cv;

using namespace std;

void OTSU(int\*\*in,int\*\*out,int rows,int cols) {

int threshold;

long sum0 = 0, sum1 = 0; //存储前景的灰度总和和背景灰度总和

long cnt0 = 0, cnt1 = 0; //前景的总个数和背景的总个数

double w0 = 0, w1 = 0; //前景和背景所占整幅图像的比例

double u0 = 0, u1 = 0; //前景和背景的平均灰度

double variance = 0; //最大类间方差

int i;

double u = 0;

double maxVariance = 0;

for (i = 1; i < 256; i++) //一次遍历每个像素

{

sum0 = 0;

sum1 = 0;

cnt0 = 0;

cnt1 = 0;

w0 = 0;

w1 = 0;

for (int m = 0; m < rows; m++) {

for (int n = 0; n < cols; n++) {

if (in[m][n] < i) {

cnt0++;

sum0 += in[m][n];

}

}

}

for (int m = 0; m < rows; m++) {

for (int n = 0; n < cols; n++) {

if (in[m][n] >= i) {

cnt1++;

sum1 += in[m][n];

}

}

}

u0 = (double)sum0 / cnt0;

w0 = (double)cnt0 /(rows\*cols);

u1 = (double)sum1 / cnt1;

w1 = 1 - w0; // (double)cnt1 / size;

u = u0 \* w0 + u1 \* w1; //图像的平均灰度

//variance = w0 \* pow((u0 - u), 2) + w1 \* pow((u1 - u), 2);

variance = w0 \* w1 \* (u0 - u1) \* (u0 - u1);

if (variance > maxVariance)

{

maxVariance = variance;

threshold = i;

}

}

for (int m = 0; m < rows; m++) {

for (int n = 0; n < cols; n++) {

if (in[m][n] > threshold) {

out[m][n] = in[m][n];

}

else

out[m][n] =0;

}

}

}

void application\_(Mat& ima) {

int rows = ima.rows;

int cols = ima.cols;

int\*\* input = new int\* [rows]();

for (int i = 0; i < rows; i++) {

input[i] = new int[cols]();

}

for (int i = 0; i < rows; i++) {

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

input[i][j] = ima.at<Vec3b>(i, j)[0];/\*灰度图只需要读一个通道即可\*/

}

}

int output\_rows = rows;

int output\_cols = cols;

int\*\* output = new int\* [output\_rows]();

for (int i = 0; i < output\_rows; i++) {

output[i] = new int[output\_cols]();

}

OTSU(input, output, rows, cols);

for (int i = 0; i < output\_rows; i++) {

for (int j = 0; j < output\_cols; j++) {

ima.at<Vec3b>(i, j)[0] = output[i][j];

ima.at<Vec3b>(i, j)[1] = output[i][j];

ima.at<Vec3b>(i, j)[2] = output[i][j];

}

}

for (int i = 0; i < rows; i++) {

delete[] input[i];

}

delete[] input;

for (int i = 0; i < output\_rows; i++) {

delete[] output[i];

}

delete[] output;

1. }