**实验报告**

专业：计算机科学与技术

姓名：薛柔

学号：3220104854

日期：2023/10/29

课程名称： 图像信息处理 指导老师： 宋明黎 成绩：

实验名称： 图像二值化和形态学操作

**一、实验目的和要求**

1. 学会对图像进行二值化。

2. 基于二值化图像，实现形态学操作：腐蚀操作，膨胀操作，开操作和闭操作。

**二、实验内容和原理**

实验内容：

1. 图像二值化；

2. 二值图像腐蚀操作；

3. 二值图像膨胀操作；

4. 二值图像开操作；

5. 二值图像闭操作。

实验原理：

1. 二值图像

二值图像是指在图像中，灰度等级只有两种，也就是说，图像中的任何像素点的灰度值均为0或者255，分别代表黑色和白色。二值图像易于计算机处理。例如，在文本识别、边缘检测、图像分割等应用中，二值图像可以帮助算法更准确地识别图像中的边缘和特征。此外，图像二值化还可以帮助减小图像的文件大小，这对于存储和传输图像是很有帮助的。

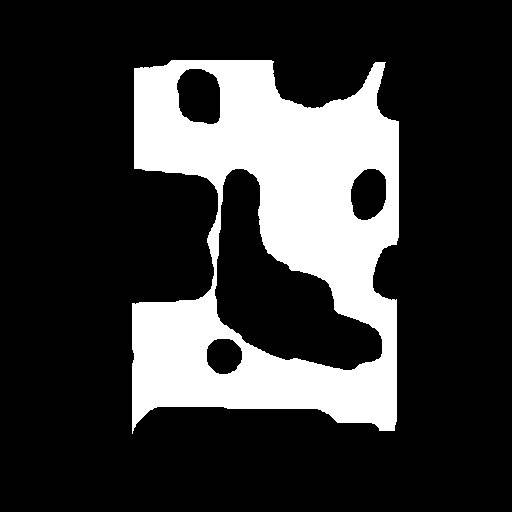


图2.1 二值图像概述图

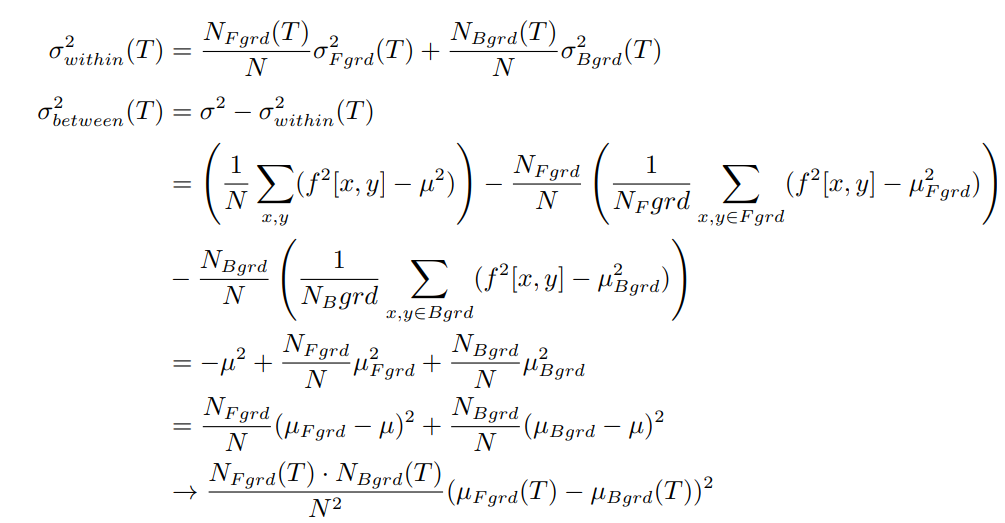
2. 图像二值化

图像二值化的过程，简单概括就是找到一个Threshold值，令灰度大于等于Threshold的像素灰度值均为255（即白色），小于Threshold的均为0（即黑色），得到一个黑白的二值图像。

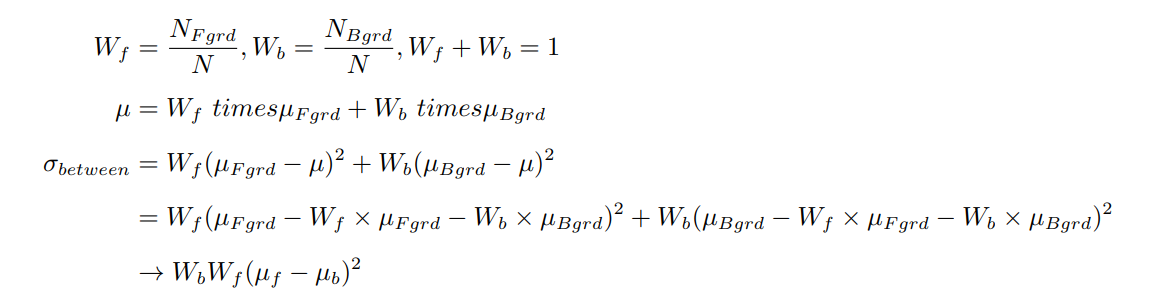
如何找Threshold的值，这里介绍三种方法：

一是直接手动设置一个固定的Threshold；

二是全局运用大津算法：将二值化得到的二值图像视为两部分，一部分对应前景（Foreground），另一部分对应背景（Background）。遍历最大灰度和最小灰度之间的 threshold 使得到的前景和背景的内部方差最小，而它们之间的方差最大。



简化后：



其中，W­­b、Wf分别为为后景、前景像素数占总像素数的比例，μ为灰度平均值。

第三种方法是滑动窗口的大津算法，在一个小窗口中局部运用大津算法，滑动这个窗口遍历整个图像。可调参数是窗口的大小和窗口滑动后重叠部分的大小。

3. 形态学操作

这里我们需要用到一个二值图像，记为A，和一个结构元，记为B。常见的结构元有方形、十字形等。用图形描述，二值图像可看成一个0和1组成的棋盘，形态学操作就是将结构元（如图灰色部分）放在棋盘上上进行与、或操作。

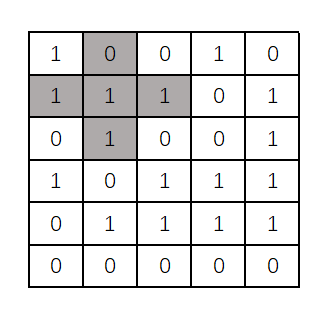


图2.2 形态学操作描述图

3.1 腐蚀操作

当十字形中的所有值均为1时，十字中心的像素才为1，否则为0，可以收缩边界。用公式描述则为：

A ⊖ B = {(x, y)|(B)xy ⊆ A}

3.2 膨胀操作

当十字形中的所有值均为0时，十字中心的像素才为0，否则为1，可以扩张边界。用公式描述则为：

A ⊕ B = {z|(B)z ∩ A ̸= ∅}

3.3 开操作

先腐蚀后膨胀就是开操作，可以消除连接薄弱的部分。

A ◦ B = (A ⊖ B) ⊕ B

3.4 闭操作

先膨胀后腐蚀就是闭操作，可以加强连接薄弱的部分。

A • B = (A ⊕ B) ⊖ B

**三、实验步骤与分析**

1. 图像二值化；

先用实验1的方式读取到图像数据并灰度化，这里不多做赘述。下面用原理中提到的三种方式实现二值化：

方法一：直接手动设置一个固定的Threshold，这里设为100；

1. **int** threshold=100;
3. PIXEL new\_pixel[pixel\_num];//建立数组存储二值图数据
4. **for**(**int** i=0;i<pixel\_num;i++){
5. //令灰度大于等于Threshold的像素灰度值均为255（即白色）
6. //小于Threshold的均为0（即黑色）
7. **if**(b[i].Y<threshold){
8. new\_pixel[i].red=new\_pixel[i].green=new\_pixel[i].blue=0;
9. }**else**{
10. new\_pixel[i].red=new\_pixel[i].green=new\_pixel[i].blue=255;
11. }
12. }

方法二：全局的大津算法

1. //RGB->YUV
2. **int** min=255;
3. **int** max=0;
4. YUV b[pixel\_num];
5. **for**(**int** i=0;i<pixel\_num;i++){
6. b[i].Y=a[i].red\*0.299+a[i].green\*0.587+a[i].blue\*0.114;
7. b[i].U=a[i].red\*-0.147+a[i].green\*-0.289+a[i].blue\*0.435;
8. b[i].V=a[i].red\*0.615+a[i].green\*-0.515+a[i].blue\*-0.100;
9. //将Y控制在[0,255]
10. **if**(b[i].Y<0)    b[i].Y=0;
11. **if**(b[i].Y>255)  b[i].Y=255;
12. //找到最小、大的Y值
13. **if**(b[i].Y<min)  min=b[i].Y;
14. **if**(b[i].Y>max)  max=b[i].Y;
15. }
17. //大津法算threshold
18. **int** threshold;
19. **double** variance=0;
20. //遍历找到最大方差
21. **for**(**int** i=1;i<max-min;i++){
22. **int** fore\_num=0;//计数前景像素数
23. **double** fore\_Y=0;//统计前景灰度总和
24. **int** back\_num=0;//计数后景像素数
25. **double** back\_Y=0;//统计后景灰度总和
26. **for**(**int** j=0;j<pixel\_num;j++){
27. //Y值<threshold(min+i),前景像素数++，反之后景++
28. **if**(b[j].Y<min+i){
29. back\_num++;
30. back\_Y+=b[j].Y;
31. }**else**{
32. fore\_num++;
33. fore\_Y+=b[j].Y;
34. }
35. }
36. //大津算法计算方差
37. **double** temp=0;
38. temp=fore\_num\*back\_num\*1.0/pixel\_num/pixel\_num\*
39. (fore\_Y/fore\_num-back\_Y/back\_num)\*(fore\_Y/fore\_num-back\_Y/back\_num);
40. //令方差最大时的灰度值为threshold
41. **if**(temp>variance){
42. variance=temp;
43. threshold=min+i;
44. }
45. }

方法三：滑动窗口的大津算法（这里令窗口大小为50\*50像素，overlap即重合为30像素的长度）：

1. //设置小窗口的长宽和重合值
2. **int** width=50;
3. **int** height=50;
4. **int** overlap=30;
5. PIXEL new\_pixel[pixel\_num];
7. **for**(**int** m=0;m<bmih.biHeight;m+=height-overlap){//m表示窗口的第一个像素在第几行，一次循环结束后向下移动窗口
8. **for**(**int** n=0;n<bmih.biWidth;n+=width-overlap){//n表示窗口的第一个像素在第几列  一次循环结束后向右移动窗口
9. **for**(**int** i=1;i<max-min;i++){
10. **int** fore\_num=0;//计数前景像素数
11. **double** fore\_Y=0;//统计前景灰度总和
12. **int** back\_num=0;//计数后景像素数
13. **double** back\_Y=0;//统计后景灰度总和
14. **for**(**int** k=0;k<height;k++){//当前像素在小窗口的第k行
15. **for**(**int** j=0;j<width;j++){//像素在小窗口的第j列
16. //Y值<threshold(min+i),前景像素数++，反之后景++
17. //计算当前像素值的下标
18. **int** subscript=m\*bmih.biWidth+n+j+k\*bmih.biWidth;
19. //判断下标是否合法
20. **if**(subscript<0||subscript>=pixel\_num){
21. **continue**;
22. }
23. **if**(b[subscript].Y<min+i){
24. back\_num++;
25. back\_Y+=b[subscript].Y;
26. }**else**{
27. fore\_num++;
28. fore\_Y+=b[subscript].Y;
29. }
30. }
31. }
32. //大津算法计算方差
33. **double** temp=0;
34. temp=fore\_num\*back\_num\*1.0/width/height/width/height\*
35. (fore\_Y/fore\_num-back\_Y/back\_num)\*(fore\_Y/fore\_num-back\_Y/back\_num);
36. **if**(temp>variance){
37. variance=temp;
38. threshold=min+i;
39. }
40. }
41. **for**(**int** k=0;k<height;k++){//当前像素在小窗口的第k行
42. **for**(**int** j=0;j<width;j++){//像素在小窗口的第j列
43. //Y值<threshold(min+i),前景像素数++，反之后景++
44. //计算当前像素值的下标
45. **int** subscript=m\*bmih.biWidth+n+j+k\*bmih.biWidth;
46. //判断下标是否合法
47. **if**(subscript<0||subscript>=pixel\_num){
48. **continue**;
49. }
50. //二值化
51. **if**(b[subscript].Y<threshold){
52. new\_pixel[subscript].red=new\_pixel[subscript].green=new\_pixel[subscript].blue=0;
53. }**else**{
54. new\_pixel[subscript].red=new\_pixel[subscript].green=new\_pixel[subscript].blue=255;
55. }
56. }
57. }
58. variance=0;//将方差重新置为0
59. }
60. }

2. 二值图像腐蚀操作；

这里尝试了两种结构元实现腐蚀操作，注意遍历每个像素的结构元时要注意下标是否合法。flag为Erosion函数返回值，默认为0（黑色），有一个像素不为黑色，则返回255（白色）。这里的腐蚀函数和膨胀函数都是对下标为i的单个像素进行的。

正方形结构元：

1. **int** Erosion(**int** i,PIXEL a[]){
2. //flag为返回值，默认为0（黑色）
3. **int** flag=0;
4. //正方形结构元
5. **for**(**int** m=-1;m<2;m++){//m为当前像素相对中心像素的行
6. **for**(**int** n=-1;n<2;n++){//m为当前像素相对中心像素的列
7. **int** subscript=i+m\*bmih.biWidth+n;
8. //判断下标是否合法
9. **if**(subscript<0||subscript>=pixel\_num){
10. **continue**;
11. }
12. //有一个像素不为黑色，则返回255（白色）
13. **if**(a[subscript].red==255){
14. flag=255;
15. **break**;
16. }
17. }
18. }
19. **return** flag;
20. }

十字形结构元：

1. **int** Erosion(**int** i,PIXEL a[]){
2. //flag为返回值，默认为0（黑色）
3. //有一个像素不为黑色，则返回255（白色）
4. **int** flag=0;
6. //十字形结构元
7. //中心像素上面的像素
8. **int** subscript=i-bmih.biWidth;
9. **if**(subscript<0||subscript>=pixel\_num){
10. //下标不合法，不做任何事
11. }**else** **if**(a[subscript].red==255){
12. flag=255;
13. }
15. //和中心像素一排的像素
16. **for**(**int** n=-1;n<2;n++){//m为当前像素相对中心像素的列
17. subscript=i+n;
18. //判断下标是否合法
19. **if**(subscript<0||subscript>=pixel\_num){
20. **continue**;
21. }
22. **if**(a[subscript].red==255){
23. flag=255;
24. **break**;
25. }
26. }
28. //中心像素下面的像素
29. subscript=i+bmih.biWidth;
30. **if**(subscript<0||subscript>=pixel\_num){
31. //下标不合法，不做任何事
32. }**else** **if**(a[subscript].red==255){
33. flag=255;
34. }
36. **return** flag;

运行后发现十字形结构元的效果较好，因此后面的三个操作均用十字形。

3. 二值图像膨胀操作；

flag为Dilation函数的返回值，默认为255（白色），有一个像素为黑色，则返回0（黑色）。

1. **int** Dilation(**int** i,PIXEL a[]){
2. //flag为返回值，默认为255（白色）
3. //有一个像素为黑色，则返回0（黑色）
4. **int** flag=255;
6. //十字形结构元
7. **int** subscript=i-bmih.biWidth;
8. **if**(subscript<0||subscript>=pixel\_num){
9. //下标不合法，不做任何事
10. }**else** **if**(a[subscript].red==0){
11. flag=0;
12. }
14. **for**(**int** n=-1;n<2;n++){//m为当前像素相对中心像素的列
15. subscript=i+n;
16. //判断下标是否合法
17. **if**(subscript<0||subscript>=pixel\_num){
18. **continue**;
19. }
20. **if**(a[subscript].red==0){
21. flag=0;
22. **break**;
23. }
24. }
25. subscript=i+bmih.biWidth;
26. **if**(subscript<0||subscript>=pixel\_num){
27. //下标不合法，不做任何事
28. }**else** **if**(a[subscript].red==0){
29. flag=0;
30. }
32. **return** flag;
33. }

4. 二值图像开操作；

先调用腐蚀函数后调用膨胀函数就是开操作。

1. //新的数组存储新的位图数据
2. PIXEL new\_pixel[pixel\_num];
3. PIXEL new\_pixel2[pixel\_num];
4. //腐蚀操作
5. **for**(**int** i=0;i<pixel\_num;i++){//遍历每个像素
6. new\_pixel[i].blue=new\_pixel[i].green=new\_pixel[i].red=Erosion(i,a);
7. }
8. //先腐蚀后膨胀——开操作
9. **for**(**int** i=0;i<pixel\_num;i++){//遍历每个像素
10. new\_pixel2[i].blue=new\_pixel2[i].green=new\_pixel2[i].red=Dilation(i,new\_pixel);
11. }

5. 二值图像闭操作。

先调用膨胀函数后调用腐蚀函数就是闭操作。

**四、实验环境及运行方法**

* 实验环境：Windows 11 系统

gcc 10.3.0 (tdm64-1) x86\_64-w64-mingw32

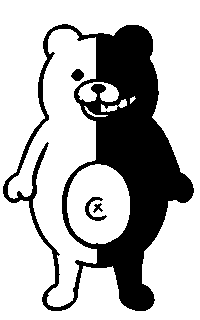
* 运行方法：先将dip\_hw2.exe文件和monokuma.bmp放在同一个目录下，运行dip\_hw2.exe文件，会得到2张新的图像，分别命名为binary.bmp，binary\_local.bmp，即为全局大津法转化后的二值图像、滑动窗口的局部大津法转化后的二值图像。此时再运行同一目录下的dip\_hw2\_morphology（注意顺序），会得到四张图像：erosion.bmp，dilation.bmp，opening.bmp，closing.bmp，分别为相应形态学操作后的二值图像（以binary\_local.bmp为基础）。当终端出现“Successfully open the image”时说明我们成功打开了图像，否则会输出“BMP Image Not Found!”。

如果运行不成功，可以将文件夹中的.c源文件重新编译运行。

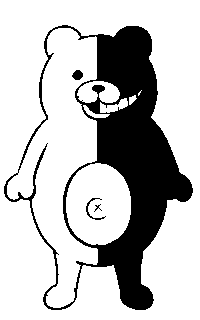
**五、实验结果展示**

****

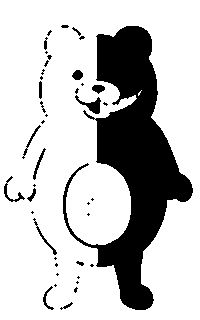
（a）原始图像monokuma.bmp



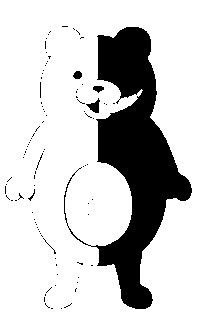
（d）膨胀操作dilation.bmp

****

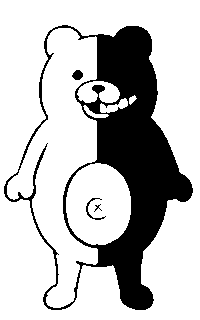
（b）窗口法binary\_local.bmp



（b）开操作opening.bmp



（c）腐蚀操作erosion.bmp



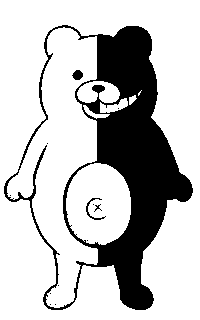
（c）闭操作closing.bmp

后面是一些尝试。

注意到，熊的左下方身体和左手臂上有一些阴影，这在全局大津法的结果上造成了一些小黑点，那么如何去除或者得到阴影呢？

****

（a）原始图像monokuma.bmp

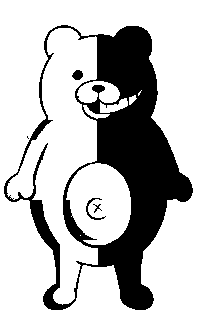
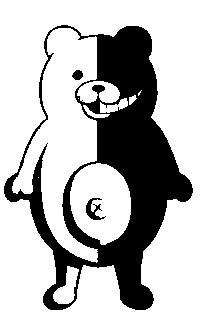
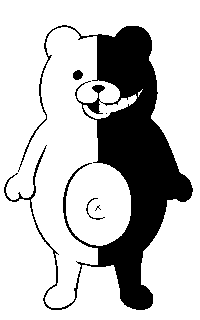




（b）全局法binary.bmp



一开始手动设定Threshold为100的结果图、简单分成四块的结果图、窗口大小为50\*50像素、重叠值为30像素宽度的结果图如下：可以看出，（a）图效果不错，（b）图保留了熊的身体阴影，而（c）图效果不佳，阴影割裂，说明参数没选对。



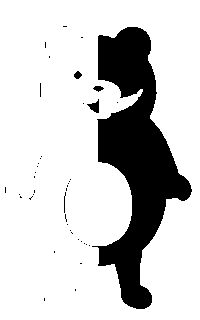
（a） （b） （c）

注意到，如果想消除阴影，需要保证黑色跟阴影在同一方块中，这样才会使阴影像素为白，所以最终设定的窗口参数为：

1. **int** width=199;
2. **int** height=60;
3. **int** overlap=10;

width与图像宽度相同，这样就得到了上上页的结果图像binary\_local.bmp（消除了噪点）。当然，消除噪点也可以用开操作达成。

下图为3\*3像素的正方形结构体作用的腐蚀效果：



可以看出腐蚀效果比十字形强很多。我希望腐蚀后仍能看清大部分轮廓，所以选择了十字形结构元。

**六、心得体会**

这次实验我进行了很多尝试：二值化方法的选取、滑动窗口法参数的选择、结构元的选取等等。几种二值化方式各有各的优势，直接自定义阈值运行成本极小，某些情况下能取得不错的效果（比如我在实验中用到的图）；全局大津法很普适且合理，但在阴影存在的情况下会有噪点，丢失信息（比如课上的案例）；滑动窗口法在确定好参数的情况下效果很好，但计算量很大。我的图片尺寸不大，因此程序均在5s内运行完毕，所以并未考虑减小算法的复杂度，这是我的一个不足之处。

这次让我对参数的选择有了经验，一开始我总是用方形的小窗口，结果割裂感很强，仔细思考、观察图像后改为长方形的小窗口，得到了较好的效果。

总的来说，本次实验进展较为顺利，过程十分有趣。