专业：计算机科学与技术
姓名：龙永奇
学号：3220105907
日期：2023/10/26

浙江大学实验报告

课程名称：图像信息处理 指导老师：宋明黎 成绩：

实验名称：bmp 灰度图像二值化并及形态学操作

一、实验目的和要求

1. 熟悉二值图像，了解二值图像的用途和结构。
2. 掌握构建二值图像的算法：大津算法，并用代码实现算法以实际操作图像。
3. 基于二值化图像，实现形态学操作。包括图形腐蚀操作，膨胀操作，开操作和闭操作。

二、实验内容和原理

实验内容：

1. 图像二值化操作
2. 图像腐蚀操作
3. 图像膨胀操作
4. 图像开操作
5. 图像闭操作

原理

1. 图像的二值化

传统的机器视觉通常包括两个关键步骤：预处理和物体检测。这两个步骤之间的桥梁是图像分割，它通过简化或改变图像的表示形式，使得图像更易于分析。其中，图像二值化是图像分割中的一种重要方法。

图像二值化的目标是将图像中的像素点的灰度值设置为 0 或 255，以呈现出明显的黑白效果。简而言之，二值图像的每个像素只能取两种值：纯黑或纯白。

由于二值图像的数据表示非常简单，因此许多视觉算法都依赖于它。通过分析二值图像，我们能更好地识别物体的形状和轮廓。此外，二值图像也常用作原始图像的掩模，就像一张部分镂空的纸，用来遮盖我们不感兴趣的区域。二值化有多种方法，其中最常见的是使用阈值法（Thresholding），这也是本次实验中采用的方法。

阈值法包括全局阈值（Global Method）和局部阈值（Local Method），又称为自适应阈值（Adaptive

Thresholding）。本次实验主要使用大津算法来实现全局阈值，除大津算法外，还有其他阈值选择算法，如平均值法和双峰法。

2. 大津算法 OTSU

大津算法是一种图像二值化算法，作用是确定将图像分成黑白两个部分的阈值。对于不同的图像，这个阈值可能不同，需要有一种算法来根据图像的信息自适应地确定这个阈值。其大体思路如下：

首先，需要将图像转换成灰度图像，255 个灰度等级。可以将图像理解成 255 个图层，每一层分布了不同的像素，这些像素垂直叠加合成了一张完整的灰度图。大津算法的目的就是找到一个合适的灰度值，大于这个值为背景（灰度值越大越黑），小于这个值为前景（灰度值越小越白）。

根据统计学方差这一概念，方差越大，相关性越低，黑白越分明。可以求出每一个灰度值对应的像素的方差，找出方差最大情况下对应的灰度值，即可得二值化分隔阈值。

N_0 为前景像素，N_1 为后景像素，N 为总像素，W_0 与W_1 对应两者占比：

$$W_0 = \frac{N_0}{N} \qquad W_1 = \frac{N_1}{N}$$

U_0 和U_1 对应前后景平均灰度：

$$U_0 = \frac{\sum \text{前景灰度}}{N_0} \qquad U_1 = \frac{\sum \text{后景灰度}}{N_1}$$

U 为总平均灰度：

$$U = \frac{\sum \text{前景灰度} + \sum \text{后景灰度}}{N}$$

定义G 为类间方差：

$$G = W_0(U_0 - U)^2 + W_1(U_1 - U)^2 = W_0W_1(U_0 - U_1)^2$$

找出最大G所对应的灰度值，即可作为阈值

在找出阈值后，将灰度小于阈值的像素点设为 0，灰度大于阈值的设为 1 即可完成图像的二值化。

3. 形态学操作

结构元(SE,Structure Element)最直接的理解就是卷积操作中的卷积核，或者是空间域滤波中提到

的滤波器。虽然形态学操作中结构元的形状可以是任意的，但是由于在图像操作中，为了方便计算，通常要求结构元是矩形的阵列，对于任意形状的结构元，如果不满足矩形的要求，则用 0 将其填充为矩形即可。

另外，结构元内部的有效元素不像滤波器那样有权值，通常结构元中只分为两种元素，就是 0 和 1，不会出现其他数值的系数。(当然对有些算法来说也有例外)。结构元对图像进行的操作也和卷积非常类似，就是由结构元的中心依次滑过图像，然后进行设计好的操作即可。在本次实验中，主要针对二值图像进行形态学操作，包括腐蚀、膨胀、开与闭操作，其中结构元即为二值模板。

a) 膨胀操作

膨胀是将与物体“接触”的所有背景点合并到该物体中，使边界向外部扩张的过程。可以用来填补物体中的空洞。（其中“接触”的含义由结构元描述）

$$A \oplus B = \{z|(B)\ z \cap A/ = \varnothing\}$$

b) 腐蚀操作

腐蚀是一种消除边界点，使边界向内部收缩的过程。可以用来消除小且无意义的物体。

$$A \ominus B = \{(x, y)|(B)\ xy \subseteq A\}$$

c) 开操作

先腐蚀，后膨胀。用来消除小物体、在纤细点处分离物体、平滑较大物体的边界的同时并不 明显改变其面积。

$$A \circ B = (A \ominus B) \oplus B$$

d) 闭操作

闭运算：先膨胀，后腐蚀。用来填充物体内细小空洞、连接邻近物体、平滑其边界的同时并不明显改 变其面积。

$$A \bullet B = (A \oplus B) \ominus B$$

三、实验步骤与分析

1. 图像的二值化

图像文件头、导入图像和实验一的方法相同，先转为灰度图，再进行大津算法可分为四个步骤：

a) 遍历灰度化后的整个图像，找到最大、最小像素值

b) 从 min+1 开始，到 max 遍历整个灰度范围，每一次求出方差 G，如果当前 G 比 maxG 大，则将 threshold 替换为当前灰度值，最后遍历完即可得到阈值

c) 对图像进行二值化，像素值大于等于阈值设为 255，小于阈值设为 0

源代码如下：

```
void OTSU(int BMPheight, int BMPwidth){
    // 获取最大和最小像素
    int byte_row = (BMPwidth + 3) / 4 * 4;
    int now_position, max = 0, min = 255;
    unsigned char now_pix;
    for (int i = 0; i < BMPheight; i++){
        for (int j = 0; j < BMPwidth; j++){
            now_position = i * byte_row + j;
            now_pix = Violet1.pix_val[now_position];
            if (now_pix > max) max = now_pix;
            if (now_pix < min) min = now_pix;
        }
    }
    // 计算阈值 threshold
    int N, N0, N1, avg_N0, avg_N1, g, max_g = 0, grey, threshold;
    for (grey = min + 1; grey <= max; grey++){
        // 灰度值每更新一次就初始化总像素值、前后景像素值、前后景平均灰度值
        N0 = N1 = avg_N0 = avg_N1 =0;
        for (int i = 0; i < BMPheight; i++){
            for (int j = 0; j < BMPwidth; j++, N++){
                now_position = i * byte_row + j;
                now_pix = Violet1.pix_val[now_position];
                if (now_pix >= grey){
                    avg_N0 += now_pix;
                    N0 ++;
                }
                else{
                    avg_N1 += now_pix;
                    N1 ++;
                }
            }
        }
        g = ((N0 * N1) / (N * N)) * (avg_N0/N0 - avg_N1/N1) * (avg_N0/N0 -
avg_N1/N1);
        if (g > max_g){
            max_g = g;
            threshold = grey;
        }
    }
    // 图像二值化
    for (int i = 0; i < BMPheight; i++){
```

```
        for (int j = 0; j < BMPwidth; j++){
            now_position = i * byte_row + j;
            if (Violet1.pix_val[now_position] >= threshold)
Violet2.pix_val[now_position] = 255;
            else Violet2.pix_val[now_position] = 0;
        }
    }
    FILE *fp = fopen("Violet_binwife_block.bmp", "wb");
    Output(&Violet2, fp);
}
```

2. **膨胀操作**

对于环绕某个像素点的结构元即为一个 3*3 的矩阵，对于以每一个像素为中心的小矩阵，遍历 9 个位置，找出中心点即可确定结构元的位置，随后判断这个位置是否出界，若未出界，就将像素值赋值为 255，否则为 0，其源代码如下：

```
void Dilation(int BMPheight, int BMPwidth, FILESTRUCT Org_Image, FILESTRUCT
Image){
    int byte_row = (BMPwidth + 3) / 4 * 4;
    int x, y, index;
    int matrix[3][3] = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}};
    for (int i = 1; i < BMPheight - 1; i++){
        for (int j = 1; j < BMPwidth - 1; j++){
            int index = 0;
            for (int p = -1; p <= 1; p++){
                for (int q = -1; q <= 1; q++){
                    if (matrix[p + 1][q + 1] == 1&&Org_Image.pix_val[(i + p)
* byte_row + (j + q)] == 255){
                        index = 1;
                        break;
                    }
                }
                if (index == 0) break;
            }
            if (index == 1) Image.pix_val[i * byte_row + j] = 255;
            else Image.pix_val[i * byte_row + j] = 0;
        }
    }
    if (flag == 216){
        FILE *fp = fopen("Violet_binwife_Open.bmp", "wb");
        Output(&Image, fp);
    }
    if (flag != 217){
        FILE *fp = fopen("Violet_binwife_Dilation.bmp", "wb");
```

```
        Output(&Image, fp);
    }
}
```

3. **腐蚀操作**

腐蚀操作选取结构元和膨胀一样，如果存在结构元，图像当前像素点像素值为 0，则舍弃此中心点。对于边界外的点将其视为 0，即原图像四周全为 0.最后遍历每个中心点，未被舍弃则赋值为 255，否则赋值为 0，其源代码如下：

```
void Dilation(int BMPheight, int BMPwidth, FILESTRUCT Org_Image, FILESTRUCT
Image){
    int byte_row = (BMPwidth + 3) / 4 * 4;
    int x, y, index;
    int matrix[3][3] = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}};
    for (int i = 1; i < BMPheight - 1; i++){
        for (int j = 1; j < BMPwidth - 1; j++){
            int index = 0;
            for (int p = -1; p <= 1; p++){
                for (int q = -1; q <= 1; q++){
                    if (matrix[p + 1][q + 1] == 1&&Org_Image.pix_val[(i + p)
* byte_row + (j + q)] == 255){
                        index = 1;
                        break;
                    }
                }
                if (index == 0) break;
            }
            if (index == 1) Image.pix_val[i * byte_row + j] = 255;
            else Image.pix_val[i * byte_row + j] = 0;
        }
    }
    if (flag == 216){
        FILE *fp = fopen("Violet_binwife_Open.bmp", "wb");
        Output(&Image, fp);
    }
    if (flag != 217){
        FILE *fp = fopen("Violet_binwife_Dilation.bmp", "wb");
        Output(&Image, fp);
    }
}
```

4. **开操作、闭操作**

开操作为先腐蚀再膨胀，闭操作反之，直接调用前面的腐蚀膨胀操作即可，源代码如下：

```
void Open(int BMPheight, int BMPwidth){
```

```
    flag = 216;
    Dilation(BMPheight, BMPwidth, Violet3, Violet5);
}
void Close(int BMPheight, int BMPwidth){
    flag = 217;
    Erosion(BMPheight, BMPwidth, Violet4, Violet6);
}
```

四、实验环境及运行方法

1. 实验环境

系统 Windows11 编译器 gcc 10.3.0x86_mingw32

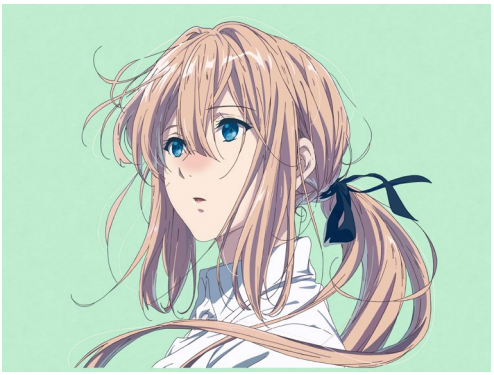
2. 运行方法

在文件夹中，Lab2.c 为源文件，运行代码，如果出现“薇尔莉特在这里:)”说明读取图片文件成功，否则会出现“薇尔莉特不见了:(”

随后程序会分别输出二值化图像：Violet_binwife.bmp，膨胀图像：Violet_binwife_Dilation.bmp 腐蚀图像：Violet_binwife_Erosion.bmp，开操作图像：Violet_binwife_Open.bmp 以及闭操作图像：Violet_binwife_Close.bmp

五、实验结果展示

1. 原图像与大津算法二值化图像



2. 膨胀与腐蚀图像

3. 开操作与闭操作图像

六、心得体会

本次实验通过对大津算法的学习与实践，完成了对灰度图像的二值化，并在此基础上进行了图像形态学操作中的膨胀、腐蚀、开操作与闭操作。在完成实验的过程中遇到了像素值指针越界、中心元选取等问题，在助教老师的帮助下终于完成修复。本次实验收获满满，对图像信息处理有了进一步的认识，期望在后续的课程和实验中进一步学习更多的图像操作与算法。