天津理工大学

计算机科学与工程学院

实验报告

2017 至 2018 学年 第 二 学期

实验七 图像的形态学处理

课程名称	数字图像处理				
学号	20152180	学生姓名	王帆	年级	2015
专业	计算机 科学与技术	教学班号	2	实验地点	7-212
实验时间	2018 年 5 月 7 日 第 7 节 至 第 8 节				
主讲教师	杨淑莹				

实验成绩

软件运行	效果	算法分析	流程设计	报告成绩	总成绩

实验（七）	实验名称	图像的形态学处理
软件环境	Windows Visual Studio 2017	
硬件环境	PC	

实验目的

实现图像的形态学处理

实验内容（应包括实验题目、实验要求、实验任务等）

设计并实现两种新的图像形态学处理方法

要求：了解图像的形态学处理基本原理，实现图像的形态学处理。
说明：图像的形态学处理
任务：
（1）在左视图中打开一幅位图。
（2）制作一个【图像 xx 形态学处理】菜单，将消息映射到右视图中，在右视图中实现图像的图像形态学处理变换。

实验过程与实验结果

1.实现目标图像的腐蚀效果

原理：

在数学形态中，设 A 为（x,y）平面上一目标区域，S 为指定大小和形状的结构元素，定义位于坐标（x,y）上的结构元素 S 所表示的区域为 S（x,y）,那么对于 A 的腐蚀结果可以表示为：

$$\left\{(x,y)\middle|(x,y)\in A,\frac{S(x,y)}{A}=\emptyset\right\}$$

在图像处理中上式可理解为定义一个结构元素 S，从图像的左上角开始，按顺序移动结构元素的位置，当结构元素位于某坐标上时，且此时结构元素完全处于目标区域内部，则保留此坐标上的像素点，否则删除此坐标上的像素点。

在实际操作中，可以将结构元素用模板的形式定义，矩形模板中属于结构元素的部分用 1 表示，不属于结构元素的部分可以标记为 0，在腐蚀处理过程中，只要对图像的每个位置都用模板进行检验，如果对于所有模板中为 1 的位置源图像上都存在指定像素，那么就保留模板所在位置的像素点，否则就清除模板所在位置的像素点。下图给出了两种不同的模板，以及它们分别对应的结构元素形状。

对于相同的图像，用不同的结构元素进行腐蚀操作的结果是不同的，因此结构元素的形状和大小往往直接决定了腐蚀操作的性能和效果。

代码：

```
//选项：形态学处理-腐蚀效果
private void ToolStripMenuItem_Corrosion_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap bitmap = COMUtil.Corrosion(objBitmap);
        curBitmap = new Bitmap(bitmap);
        bitmap.Dispose();
        this.pictureBox_new.Image = curBitmap;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
public static Bitmap Corrosion(Bitmap bitmap)
{
    int width = bitmap.Width;
    int height = bitmap.Height;
    Bitmap newBitmap = new Bitmap(width, height);

    for (int i = 1; i < width - 1; i++)
    {
        for (int j = 1; j < height - 1; j++)
        {
            int a = Binaryzation(bitmap.GetPixel(i, j));
            int x1 = Binaryzation(bitmap.GetPixel(i - 1, j));
            int x2 = Binaryzation(bitmap.GetPixel(i, j + 1));
            int x3 = Binaryzation(bitmap.GetPixel(i + 1, j));
            int x4 = Binaryzation(bitmap.GetPixel(i, j - 1));
            if (x1 == 255 || x2 == 255 || x3 == 255 || x4 == 255 && a == 0)
            {
                newBitmap.SetPixel(i, j, Color.FromArgb(255, 255, 255));
            }
            else
            {
                newBitmap.SetPixel(i, j, Color.FromArgb(0, 0, 0));
            }
        }
    }
    return newBitmap;
}
```

效果图：

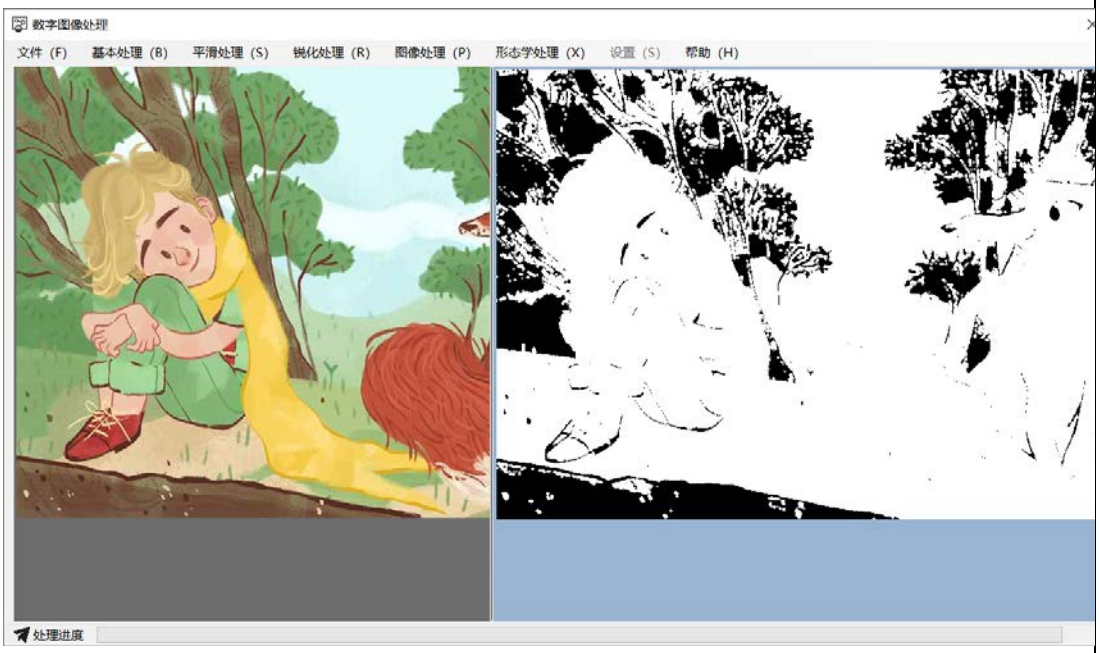

图 1 腐蚀效果

2.实现目标图像的膨胀效果

原理：

膨胀在数学形态学运算中的作用是扩展物体的边界点，在数字图像处理中，对于确定的结构元素，通过膨胀运算可以使一些相临距离较短的区域进行连接。

在数学形态中，设 A 为(x,y)平面上一目标区域，S 为指定大小和形状的结构元素，定义位于坐标(x,y)上的结构元素 S 所表示的区域为 S(x,y),那么对于 A 的膨胀结果可以表示为:

$$\{(x,y)\,|\,(x,y) \in A, S(x,y) \cap A \neq \varnothing\}$$

在图像处理中上式可以理解为定义一个结构元素 S，从图像左上角开始，按顺序移动结构元素的位置，当结构元素位于某坐标上时，且此时结构元素与目标图像存在交集，则保留此坐标上的像素点，否则删除此坐标上的像素点。

图像的膨胀处理同样可以借助模板来实现，像图像腐蚀一样，用模板表示结构元素的形状，则图像的膨胀处理可以描述为：移动模板使之遍历目标图像的每一个像素，当模板处于坐标(x,y)处之时，若模板中任意为 1 的位置对应的像素存在，则膨胀结果中坐标（x,y）对应位置的像素存在，否则坐标（x,y）对应位置的像素不存在。

在解决二值图像的膨胀处理时，用数字逻辑学中集合的运算思想就完全可以解决所有问题，但对于彩色图像的膨胀处理，纯粹的集合运算是远远不够的。对于彩色图像的腐蚀处理，可以通过最小值描述法来解决，那么对于彩色图像的膨胀处理，同样可以利用取最大值的办法实现。

代码：

```
//选项：形态学处理-膨胀效果
private void ToolStripMenuItem_expansion_Click(object sender, EventArgs e)
```

```
{
    try
    {
        Bitmap bitmap = COMUtil.Expansion(objBitmap);
        curBitmap = new Bitmap(bitmap);
        bitmap.Dispose();
        this.pictureBox_new.Image = curBitmap;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
public static Bitmap Expansion(Bitmap bitmap)
{
    int width = bitmap.Width;
    int height = bitmap.Height;
    Bitmap newBitmap = new Bitmap(width, height);

    for (int i = 1; i < width - 1; i++)
    {
        for (int j = 1; j < height - 1; j++)
        {
            int a = Binaryzation(bitmap.GetPixel(i, j));
            int x1 = Binaryzation(bitmap.GetPixel(i - 1, j));
            int x2 = Binaryzation(bitmap.GetPixel(i, j + 1));
            int x3 = Binaryzation(bitmap.GetPixel(i + 1, j));
            int x4 = Binaryzation(bitmap.GetPixel(i, j - 1));
            if (x1 == 0 || x2 == 0 || x3 == 0 || x4 == 0 && a == 0)
            {
                newBitmap.SetPixel(i, j, Color.FromArgb(0, 0, 0));
            }
            else
            {
                newBitmap.SetPixel(i, j, Color.FromArgb(255, 255, 255));
            }

        }
    }
    return newBitmap;
}
```

效果图：

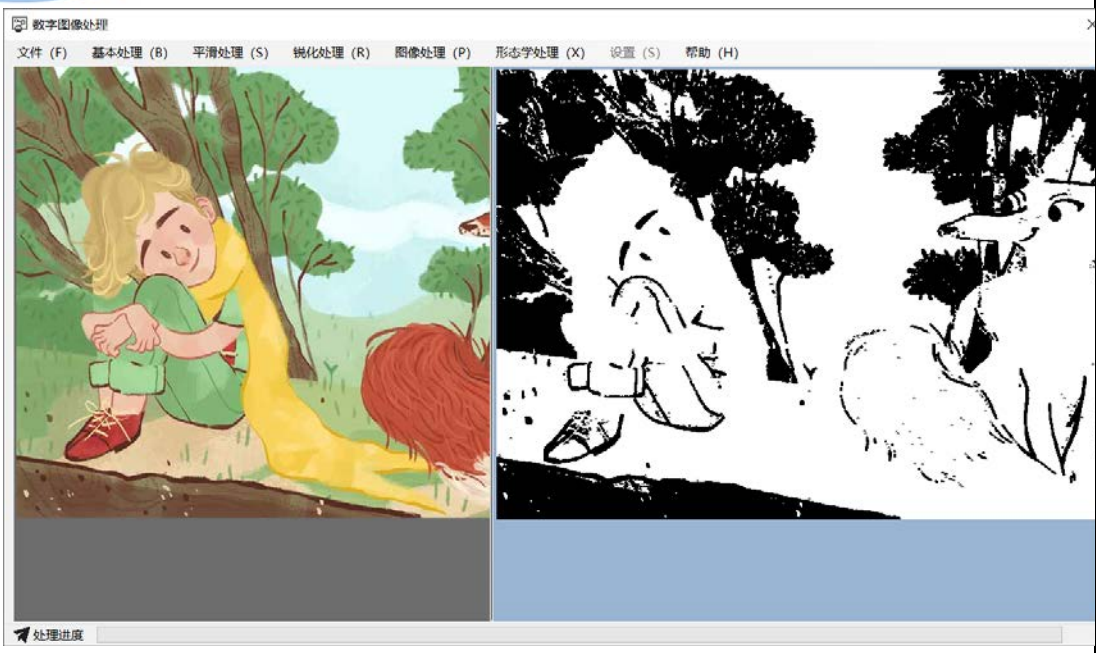

图 2 膨胀效果

2.实现目标图像的开闭运算效果

原理：

腐蚀和膨胀是数字图像形态学中最基本的两个算子，通过对他们的组合及配合集合的运算，可以构造出形态运算族来实现更复杂的功能。开运算和闭运算就是两个最常用的形态运算族，对于目标图像 A 和结构元素 S，用 A+S 表示利用 S 对 A 进行膨胀，用 A-S 表示利用 S 对 A 腐蚀，定义

$$A \circ S = （A - S）+ S$$

$$A \bullet S = （A - S）+ S$$

可见对于目标图像 A 的开运算可以分解为先对 A 进行腐蚀处理，再对腐蚀结果进行膨胀处理；而 A 的闭运算可以分解为先对 A 进行膨胀处理，再对膨胀结果进行腐蚀处理。开运算和闭运算在图像处理中常用来对目标图像进行过滤或修补。

图像的开运算的结果是有损的，也就说进行开运算后的图像并不等于源图像。经过开运算处理后只有那些在附近存在完整结构元素的像素点会被保留，其他的像素点都会被清除。

图像的开运算常常用来对目标图像进行噪音处理，同时，图像的开运算可以选择性的保留目标图像中符合结构元素几何性质的部分，而过滤掉相对结构元素而言残损的部分。

图像的闭运算的结果常常会比源图像增加一些像素，在闭运算处理中，距离较近的区域可能被连接，离散的杂点通常会被放大，放大的程度和形状由结构元素的形状和大小决定。

图像的闭运算常常用来对目标图像分开的区域进行连接及对图像中的细小缝隙进行填补，通过适当的选择结构元素，图像的闭运算可以令图像的填补结果具有一定的几何特征，适当的对图像进行闭运算有时可以使图像变得更加清晰和连贯，同时可以避免源图像中的线条加粗。

代码：

```
//选项：形态学处理-开闭运算-开运算

private void ToolStripMenuItem_open_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap bitmap = COMUtil.Corrosion(objBitmap);
        Bitmap bitmap_new = COMUtil.Expansion(bitmap);
        curBitmap = new Bitmap(bitmap_new);
        bitmap.Dispose();
        bitmap_new.Dispose();
        this.pictureBox_new.Image = curBitmap;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}

private void ToolStripMenuItem_close_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap bitmap = COMUtil.Expansion(objBitmap);
        Bitmap bitmap_new = COMUtil.Corrosion(bitmap);
        curBitmap = new Bitmap(bitmap_new);
        bitmap.Dispose();
        bitmap_new.Dispose();
        this.pictureBox_new.Image = curBitmap;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

示意图：

图 3 开运算处理

图 4 闭运算处理

2.实现目标图像的细化处理效果

原理：

图像的细化处理是指在保留源图像几何形状的前提下，尽量减少图像所包含的信息量，图像细化的结果被形象的成为图像的骨架，获得图像骨架的方法有很多，下面介绍的图像细化算法是基于形态学处理的思想实现的。

图像细化的操作实际是一个逐渐腐蚀的过程，每次从图像的边缘处腐蚀掉一个像素，直到不能继续腐蚀为止。算法实现的关键是如何在每次腐蚀中判断像素点是否可以清除，根据不同的需要，图像细化的算法和判断条件有很多，

但是最基本的有两条准则首先图像的细化不能缩短图像骨架的长度，其次细化不能将图像分解成不同部分。

这两条准则在判断时可以总结为以下条件：

1. 计算当前像素邻域内 8 个方向的可见像素数目，如果少于两个，则删除此像素会缩短图像骨架长度，若多于 6 个像素，则删除此像素会改变图像骨架几何形状；
2. 计算当前像素周围邻域内的区域数目，如果多余 1 个，那么删除中心像素会目标图像分解成不同部分。

代码：

```
public static int getAshETV(Bitmap bitmap, int x, int y)
{
    Color color = bitmap.GetPixel(x, y);
    int a = (color.R + color.G + color.B) / 3;
    return a;
}
private void ToolStripMenuItem_thin_Click(object sender, EventArgs e)
{
    try
    {
        int a = 0;
        int width = objBitmap.Width;
        int height = objBitmap.Height;
        int t1, t2;
        int count;
        bool finish = false;
        Bitmap bitmap = new Bitmap(width, height);
        int[,] nb = new int[5, 5] { { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0,
0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 } };
        for (int i = 0; i < width; i++)
        {
            for (int j = 0; j < height; j++)
            {
                if (COMUtil.getAshETV(objBitmap, i, j) < 100)
                {
                    bitmap.SetPixel(i, j, Color.FromArgb(a, a, a));
                }
                else
                {
                    bitmap.SetPixel(i, j, Color.FromArgb(255 - a, 255 - a, 255
- a));
                }
            }
        }
        while (!finish)
```

```
        {
            finish = true;
            for (int y = 2; y < height - 2; y++)
            {
                for (int x = 2; x < width - 2; x++)
                {
                    if (COMUtil.getAshETV(bitmap, x, y) == 255 - a)
                        continue;
                    t1 = 0;
                    t2 = 0;
                    for (int j = 0; j < 5; j++)
                    {
                        for (int i = 0; i < 5; i++)
                        {
                            if (COMUtil.getAshETV(bitmap, x + i - 2, y + j - 2)
== a)
                            {
                                nb[j, i] = 1;
                            }
                            else
                            {
                                nb[j, i] = 0;
                            }
                        }
                    }
                    count = nb[1, 1] + nb[1, 2] + nb[1, 3] + nb[2, 1] + nb[2,
3] + nb[3, 1] + nb[3, 2] + nb[3, 3];
                    if (count >= 2 && count <= 6) {t1 = 1; }
                    else{ continue; }
                    count = 0;
                    if (nb[1, 2] == 0 && nb[1, 1] == 1)
                        count++;
                    if (nb[1, 1] == 0 && nb[2, 1] == 1)
                        count++;
                    if (nb[2, 1] == 0 && nb[3, 1] == 1)
                        count++;
                    if (nb[3, 1] == 0 && nb[3, 2] == 1)
                        count++;
                    if (nb[3, 2] == 0 && nb[3, 3] == 1)
                        count++;
                    if (nb[3, 3] == 0 && nb[2, 3] == 1)
                        count++;
                    if (nb[2, 3] == 0 && nb[1, 3] == 1)
                        count++;
```

```
                    if (nb[1, 3] == 0 && nb[1, 2] == 1)
                        count++;
                    if (count == 1) {t2 = 1;}
                    else{ continue; }
                    if (t1 * t2 == 1)
                    {
                bitmap.SetPixel(x, y, Color.FromArgb(255 - a, 255 - a, 255 - a));
                        finish = false;
                    }
                }
            }
        }
        curBitmap = new Bitmap(bitmap);
        bitmap.Dispose();
        this.pictureBox_new.Image = curBitmap;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

示意图：

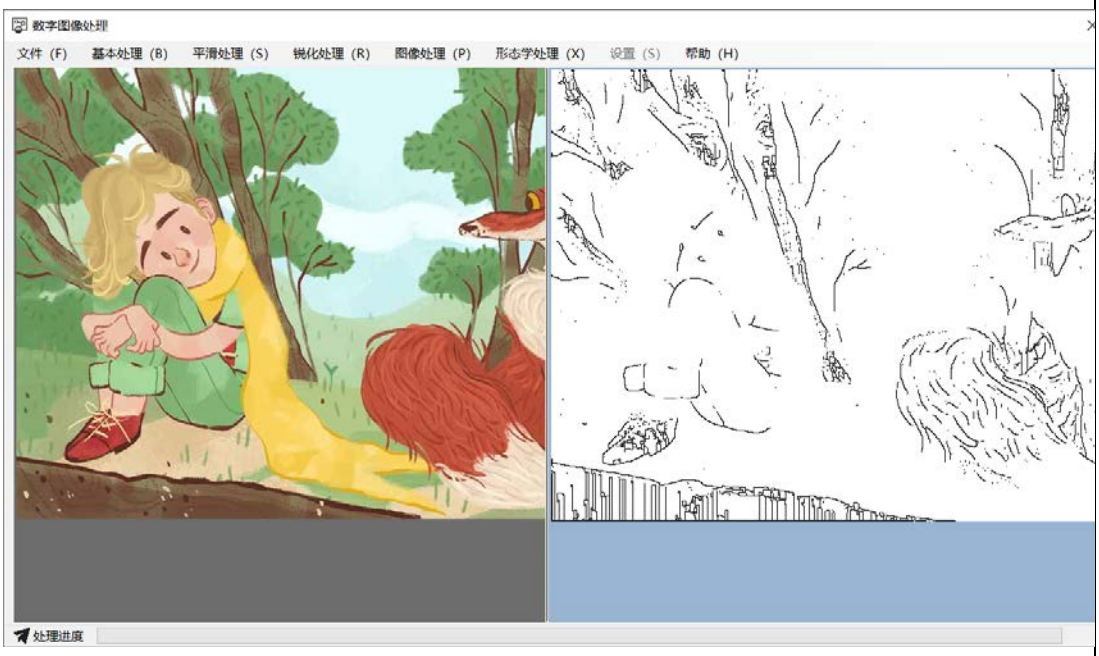

图 5 细化效果

附录

参考资料

1. 形态学在图像处理中的应用 - CSDN 博客
https://blog.csdn.net/sn_gis/article/details/57414029
2. 【C#图像处理】数学形态学 - redrain007@126 的日志 - 网易博客
http://blog.163.com/redrain007@126/blog/static/652683872016097575858/
3. 形态学图像处理 - CSDN 博客
https://blog.csdn.net/wang_juan0205/article/details/49678983