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

实验五 图像的锐化处理

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 4 月 23 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（五）	实验名称	图像的锐化处理
软件环境	Windows Visual Studio 2017	
硬件环境	PC	

实验目的

实现图像的锐化处理。

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锐化处理方法

要求：了解图像锐化基本原理，实现图像锐化。
说明：图像的锐化基本原理
任务：
（1）在左视图中打开一幅位图。
（2）制作两个【图像的锐化】菜单，将消息映射到右视图中，在右视图中实现图像的拉普拉斯锐化。

实验过程与实验结果

1. 拉普拉斯锐化

原理：

拉普拉斯锐化是利用拉普拉斯算子对图像进行边缘增强的一种方法，拉普拉斯算子是以图像邻域内像素灰度差分计算为基础，通过二阶微分推导出的一种图像邻域增强的算法。它的基本思想是：当邻域的中心像素灰度低于它所在邻域内其它像素的平均灰度时，此中心像素的灰度应被进一步降低，当邻域的中心像素灰度高于它所在邻域内其它像素的平均灰度时此中心像素的灰度应被进一步提高，以此实现图像的锐化处理。

拉普拉斯边缘检测算子也是针对图像中 3*3 邻域的检测，它分为正相拉普拉斯算法和反相拉普拉斯算法，他们分别对应两种不同的边缘。有如下公式：

$$\text{正相: } L(C) = (N0 + N1 + N2 + N3 + N4 + N5 + N6 + N7) - 8 * C$$

$$\text{反相: } NL(C) = 8 * C - (N0 + N1 + N2 + N3 + N4 + N5 + N6 + N7)$$

利用拉普拉斯边缘检测算子对图像邻域进行处理得到的结果可能是正数也可能是负数，二者分别反映了不同的边缘，这里分别命名为正边缘和负边缘。对于可能得到的两种边缘，在图像处理中有时只需要关心其中一种，那么就可以通过忽略正数或负数的方法过滤掉无意义的边缘信息。

在算法实现过程中，拉普拉斯锐化算法通过对邻域中心像素的四方向或八方向求梯度，并将梯度和相加来判断中心像素灰度与邻域内其他像素灰度的关系，

并用梯度运算的结果对像素灰度进行调整。

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

图 1-1 四方向模板（左）与八方向模板（右）

通过模板可以发现：当邻域内像素灰度相同时，模板的卷积运算结果为 0；当中心像素的灰度高于邻域内其他像素的平均灰度时，模板的卷积运算结果为正数，当中心像素的灰度低于邻域其他像素的平均灰度时，模板的卷积运算结果为负数。对卷积运算结果用适当的衰减因子处理并加在原始中心像素上，就可以实现图像的锐化处理。当然，除了 3x3 邻域，图像的拉普拉斯锐化算法还可以扩展到其他大小邻域的情况。

实现步骤：

1.获取原图像的 Bitmap 对象 objBitmap，以及其大小参量；
2.构造一个新 Bitmap 对象 bitmap，初始化 Laplace 模板；
3.将该点像素值与 Laplace 模板进行卷积运算，并迭代赋值到 bitmap 内；
4.对运算结果溢出（超过 255 或小于 0）的值进行处理；
5.使用 bitmap 构造全局变量 curBitmap，销毁 bitmap 对象，使用 curBitmap 初始化右侧显示框。

代码：

```
//选项：平滑处理-锐化处理-Laplace锐化
private void ToolStripMenuItem_sharpen_laplace_Click(object sender, EventArgs
e)
{
    try
    {
        int base_value = 3;
        int[] template = new int[9] { -1, -1, -1, -1, 9, -1, -1, -1, -1 };
        Bitmap bitmap = COMUtil.Sharpen_operation(base_value, template,
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);
    }
```

```
}
// namespace COMUtil;
// 锐化相关方法
public static Color Convolution(int[] template, int base_value, int x, int y,
Bitmap bitmap)
{
    int r = 0, g = 0, b = 0;
    int Index = 0;
    Color pixel;
    for (int col = -base_value / 2; col <= base_value / 2; col++)
        for (int row = -base_value / 2; row <= base_value / 2; row++)
        {
            pixel = bitmap.GetPixel(x + row, y + col);
            r += pixel.R * template[Index];
            g += pixel.G * template[Index];
            b += pixel.B * template[Index];
            Index++;
        }
    //处理颜色值溢出
    r = r > 255 ? 255 : r;
    r = r < 0 ? 0 : r;
    g = g > 255 ? 255 : g;
    g = g < 0 ? 0 : g;
    b = b > 255 ? 255 : b;
    b = b < 0 ? 0 : b;
    return Color.FromArgb(r, g, b);
}
public static Bitmap Sharpen_operation(int base_value, int[] template, Bitmap
bitmap)
{
    int Height = bitmap.Height;
    int Width = bitmap.Width;
    Bitmap newBitmap = new Bitmap(Width, Height);
    for (int x = 1; x < Width - 1; x++)
        for (int y = 1; y < Height - 1; y++)
        {
            newBitmap.SetPixel(x - 1, y - 1, Convolution(template, base_value,
x, y, bitmap));
        }
    return newBitmap;
}
```

效果图：

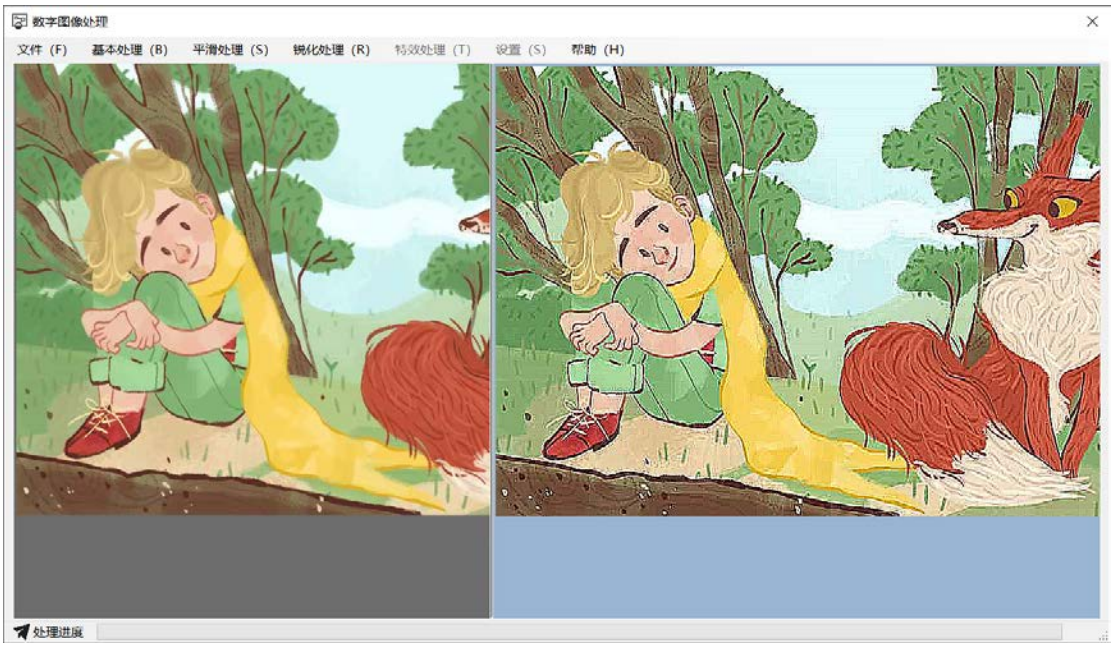

图 1-2 拉普拉斯锐化

2. Sobel 边缘细化

原理：

Sobel 算子是针对图像 3*3 邻域的处理，它的原理是先后在水平和垂直方向上对邻域灰度求差分，然后取两个差分的平均值或其中较大者，通常我们使用两个差分的较大者。

有如下公式：

$$S(C) = \max\begin{Bmatrix}|(N0 + N1 * 2 + N2) - (N4 + N5 * 2 + N6)|\\|(N0 + N7 * 2 + N6) - (N2 + N3 * 2 + N4)|\end{Bmatrix}$$

或：

$$S(C) = \frac{\begin{bmatrix}|(N0 + N1 * 2 + N2) - (N4 + N5 * 2 + N6)|\\|(N0 + N7 * 2 + N6) - (N2 + N3 * 2 + N4)|\end{bmatrix}}{2}$$

与 2*2 邻域相似，在 3*3 邻域中，若已知 C 像素的偏移位置为 n. 那么很容易得到邻域中的其他像素的偏移位置，对应关系如图所示：

N_0 $(n - wx4 - 4)$	N_1 $(n - wx4)$	N_2 $(n - wx4 + 4)$
N_7 $(n - 4)$	C (n)	N_3 $(n + 4)$
N_6 $(n + wx4 - 4)$	N_5 $(n + wx4)$	N_4 $(n + wx4 + 4)$

图 2-1 Sobel 邻域偏移量

实现步骤：

1.获取原图像的 Bitmap 对象 objBitmap，以及其大小参量；
2.构造一个新 Bitmap 对象 bitmap，初始化 Sobel 模板；
3.将该点像素值与 Sobel 模板进行卷积运算，并迭代赋值到 bitmap 内；
4.对运算结果溢出（超过 255 或小于 0）的值进行处理；
5.使用 bitmap 构造全局变量 curBitmap，销毁 bitmap 对象，使用 curBitmap 初始化右侧显示框。

代码：

```
//选项：平滑处理-锐化处理-Sobel边缘细化-水平边缘
private void ToolStripMenuItem_sharpen_sobel_horizontal_Click(object sender,
EventArgs e)
{
    try
    {
        int base_value = 3;
        int[] template = new int[9] { -1, -2, -1, 0, 0, 0, 1, 2, 1 };
        Bitmap bitmap = COMUtil.Sharpen_copy(base_value, template, 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);
    }
}


//选项：平滑处理-锐化处理-Sobel边缘细化-垂直边缘
private void ToolStripMenuItem_sharpen_sobel_vertical_Click(object sender,
EventArgs e)
{
    try
    {
        int base_value = 3;
        int[] template = new int[9] { 1, 0, -1, 2, 0, -2, 1, 0, -1 };
        Bitmap bitmap = COMUtil.Sharpen_copy(base_value, template, 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 Sharpen_copy(int base_value, int[] template, Bitmap
bitmap)
{
    int Height = bitmap.Height;
    int Width = bitmap.Width;
    Bitmap newBitmap = new Bitmap(Width, Height);
    for (int x = 1; x < Width - 1; x++)
        for (int y = 1; y < Height - 1; y++)
        {
            newBitmap.SetPixel(x - 1, y - 1, Convolution(template, base_value,
x, y, bitmap));
        }
    return newBitmap;

}
```

效果图：

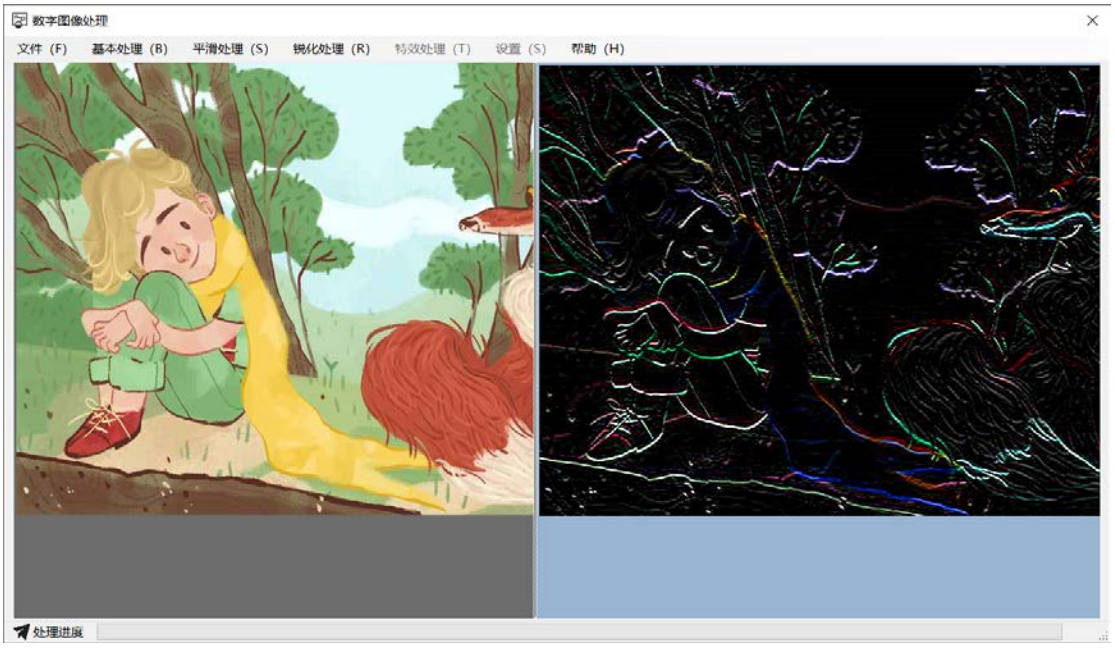

图 2-2 Sobel 边缘细化-水平

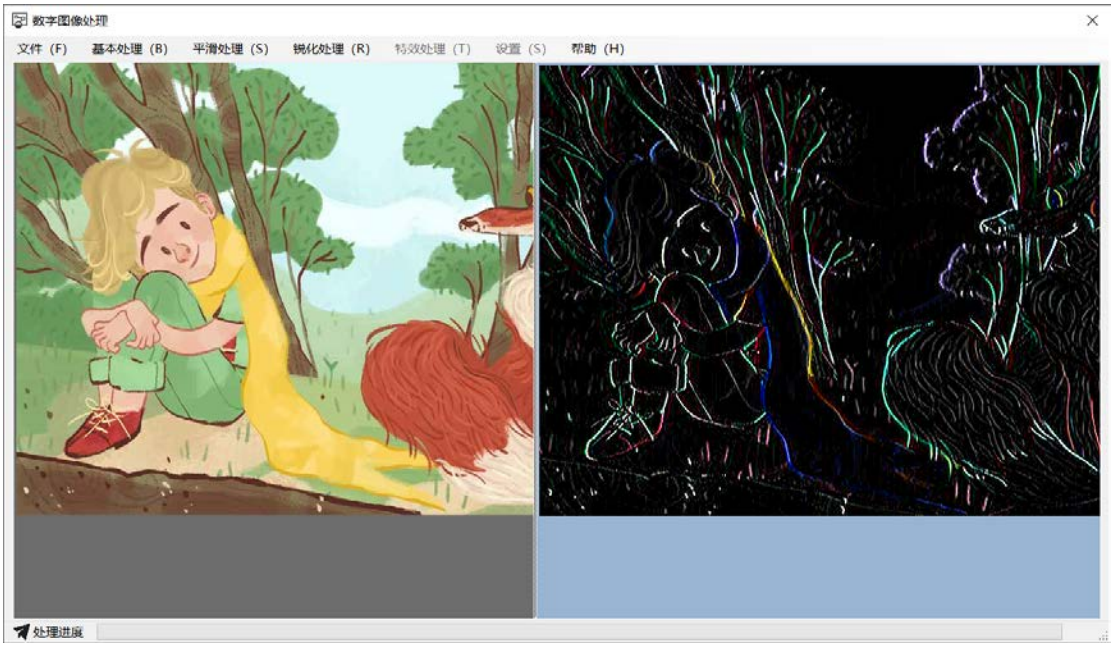

图 2-2 Sobel 边缘细化-垂直

附录

边缘检测

图像的边缘检测是一种通过图像邻域灰度运算提取区域边缘的增强算法，它可以简单理解为提取图像中区域的轮廓。图像中的区域划分以像素灰度为依据，每个区域中的像素灰度大致相同，而区域之间的边界就称为边缘，寻找这些边缘就是图像边缘检测的目的。

边缘检测的算法通常通过对邻域内像素灰度求一阶导数，二阶导数及梯度来实现，这些计算经过简化的结果称为算子，在使用算子进行边缘检测时，我们给定边缘为像素集合：

$$\{x|F(x) > I\},$$

其中，x 为像素编号，$F(x)$为算子计算结果，I为临界值。

从定义中可以看出，边缘检测的过程可以分解为对图像的每个像素分别计算判断的过程。

边缘检测的结果通常用灰度图来表示，原图像中的边缘部分用灰度较高的像素显示，而没有边缘的部分在灰度图中显示为黑色。

灰度图中的像素的灰度可以直接通过算子计算原图像中对应像素的灰度差分得到。在实现时，只用依次对原图像的每个像素进行运算，并把结果保存为灰度图中的像素即可。

带方向的边缘检测同样需要对邻域内像素灰度求差分，与常规边缘检测不同的是，带方向的边缘检测不仅要考虑邻域像素的灰度跃变，还要考虑跃变的方向，这里使用模板来实现。

常用的带方向的边缘检测模板有 3 种，分别是 Prewitt, Robinson 与 Kirsch.

1	1	1	1	1	-1
1	-2	1	1	-2	-1
-1	-1	-1	1	1	-1

1	2	1	1	0	-1
0	0	0	2	0	-2
-1	-2	-1	1	0	-1

5	5	5	5	-3	-3
-3	0	-3	5	0	-3
-3	-3	-3	5	-3	-3

图 3 Prewitt 模板（左）Robinson 模板（中）与 Kirsch 模板（右）

使用模板进行边缘检测时，只需将图像邻域内像素的灰度按照模板中给出的权值相加即可，使用 Prewitt 水平方向的模板和 Kirsch 垂直方向的模板进行边缘检测的算子公式分别为：

$$P(C) = scale * (A0 + A1 + A2 + A3 - A4 - A5 - A6 + A7 - 2 * C)$$

$$K(C) = scale * (5 * A0 - 3 * A1 - 3 * A2 - 3 * A3 - 3 * A4 - 3 * A5 + 5 * A6 + 5 * A7)$$

边缘模板中邻域内各处权值相加的结果为 0，也就是说，当邻域内各像素的灰度相同时，算子的运算结果为 0，而邻域内各像素灰度差别越大，算子的运算结果的差别也越大。

参考资料

1. 拉普拉斯锐化 - le vin - 博客园
https://www.cnblogs.com/DemonEdge/archive/2013/11/12/3419002.html
2. 拉普拉斯(laplacian)滤波实现图像锐化分析 - CSDN 博客
https://blog.csdn.net/scottly1/article/details/44408343
3. C#图像处理之边缘检测(Sobel)的方法 - CodeWeblog.com
http://www.codeweblog.com/c-图像处理之边缘检测-sobel-的方法/