重 庆 交 通 大 学

学生实验报告

课 程 名 称：数 字 图 像 处 理 .

开 课实验室：软 件 实 验 中 心 .

学 院：信息学院年级物联网工程专业 2 班

学 生 姓 名：李骏飞学 号 632109160602

指 导 教 师：蓝 章 礼 .

开 课 时 间：2023 至 2024 学年第 二 学期

成 绩	
教师签名	

实验项目名称		图像的锐化			
姓名	李骏飞	学号	**632109160602**	实验日期	**2024.4.17**

教师评阅：

1:实验目的明确□A□B□C□D

2:内容与原理□A□B□C□D

3:实验报告规范□A□B□C□D

4:实验主要代码与效果展示□A□B□C□D

5:实验分析总结全面□A□B□C□D

实验记录

一、实验目的

完成图像的锐化操作的程序编写。

二、实验主要内容及原理

边缘检测是图像处理和计算机视觉中的基本问题，边缘检测的目的是标识数字图像中亮度变化明显的点。图像属性中的显著变化通常反映了属性的重要事件和变化，包括深度不连续、表面方向不连续、物质属性变化和场景照明变化。边缘检测特征是提取中的一个研究领域。图像边缘检测大幅度地减少了数据量，并且剔除了可以认为不相关的信息，保留了图像重要的结构属性。

图像锐化算法是一种用于增强图像中细节和边缘的技术。这些方法都可以用于图像锐化。 简言之，锐化就是增强边缘上的差异，来突出边缘周围像素间颜色亮度值。

（1）Laplacian（拉普拉斯）算子：

拉普拉斯算子是图像邻域内像素灰度差分计算的基础，通过二阶微分推导出的一种图像邻域增强算法。它的基本思想是当邻域的中心像素灰度

低于它所在邻域内的其他像素的平均灰度时，此中心像素的灰度应该进一步降低；当高于时进一步提高中心像素的灰度，从而实现图像锐化处理。

在算法实现过程中，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。

对于数字图像，拉普拉斯算子可以简化为：

$$g(i,j) = 4f(i,j) - f(i+1,j) - f(i-1,j) - f(i,j+1) - f(i,j-1)$$

其中 K=1，I=1 时 H（r,s）取下式，四方面模板：

$$H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

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内其他像素的平均灰度时，模板的卷积的负数。对卷积运算的结果用适当的衰弱因子处理并加在原中心像素上，就可以实现图像的锐化处理。

其他常用的拉普拉斯核 H 如下：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 2 & -8 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

（2）Sobel 算子

采用梯度微分锐化图像，会让噪声、条纹得到增强，Sobel 算子在一定程度上解决了这个问题:

$$\begin{aligned} S_x &= [f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1)] \\ &- [f(i-1,j-1) + 2f(i-1,j) + f(i-1,j+1)] \\ S_y &= [f(i-1,j+1) + 2f(i,j+1) + f(i+1,j+1)] \\ &- [f(i-1,j-1) + 2f(i,j-1) + f(i+1,j-1)] \end{aligned}$$

从这个式子中，可以得到两个性质，

(1) Sobel 引入了平均的因素，因此对噪声有一定的平滑作用

(2) Sobel 算子的操作就是相隔两个行（列）的差分，所以边缘两侧元素的得到了增强，因此边缘显得粗而亮。

Sobel 算子表示形式为：

$$H_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

（3）Prewitt 算子

Prewitt 算子是一种图像边缘检测的微分算子，其原理是利用特定区域内像素灰度值产生的差分实现边缘检测。由于 Prewitt 算子采用 3*3 模板对区域内的像素值进行计算，而 Robert 算子的模板为 22，故 Prewitt 算子的边缘检测结果在水平方向和垂直方向均比 Robert 算子更加明显。Prewitt 算子适合用来识别噪声较多、灰度渐变的图像，其计算公式如下所示：

$$H_X = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_Y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

（4）Roberts 算子

Roberts 算子又称为交叉微分算法，它是基于交叉差分的梯度算法，通过局部差分计算检测边缘线条。常用来处理具有陡峭的低噪声图像，当图像边缘接近于正 45 度或负 45 度时，该算法处理效果更理想。其缺点是对边缘的定位不太准确，提取的边缘线条较粗。

Roberts 算子的模板分为水平方向和垂直方向，如下式所示，从其模板可以看出，Roberts 算子能较好的增强正负 45 度的图像边缘。

$$d_X = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad d_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

（5）Krisch 算子和 Robinson 算子

Krisch 算子由以下 8 个卷积核组成。图像与每一个核进行卷积，然后取绝对值作为对应方向上的边缘强度的量化。对 8 个卷积结果取绝对值，然后在对应值位置取最大值作为最后输出的边缘强度。

$$\mathbf{k}_1=\begin{pmatrix}5&5&5\\-3&0&-3\\-3&-3&-3\end{pmatrix},\mathbf{k}_2=\begin{pmatrix}-3&-3&-3\\-3&0&-3\\5&5&5\end{pmatrix},\mathbf{k}_3=\begin{pmatrix}-3&5&5\\-3&0&5\\-3&-3&-3\end{pmatrix},\mathbf{k}_4=\begin{pmatrix}-3&-3&-3\\5&0&-3\\5&5&-3\end{pmatrix},$$

$$\mathbf{k}_5=\begin{pmatrix}-3&-3&5\\-3&0&5\\-3&-3&5\end{pmatrix},\mathbf{k}_6=\begin{pmatrix}5&-3&-3\\5&0&-3\\5&-3&-3\end{pmatrix},\mathbf{k}_7=\begin{pmatrix}-3&-3&-3\\-3&0&5\\-3&5&5\end{pmatrix},\mathbf{k}_8=\begin{pmatrix}5&5&-3\\5&0&-3\\-3&-3&-3\end{pmatrix},$$

Robinson 算子也由 8 个卷积核组成。

$$\mathbf{r}_1=\begin{pmatrix}1&1&1\\1&-2&1\\-1&-1&-1\end{pmatrix},\mathbf{r}_2=\begin{pmatrix}1&1&1\\-1&-2&1\\-1&-1&1\end{pmatrix},\mathbf{r}_3=\begin{pmatrix}-1&1&1\\-1&-2&1\\-1&1&1\end{pmatrix},\mathbf{r}_4=\begin{pmatrix}-1&-1&1\\-1&0&1\\1&1&1\end{pmatrix},$$

$$\mathbf{r}_5=\begin{pmatrix}-1&-1&-1\\1&-2&1\\1&1&1\end{pmatrix},\mathbf{r}_6=\begin{pmatrix}1&-1&-1\\1&-2&-1\\1&1&1\end{pmatrix},\mathbf{r}_7=\begin{pmatrix}1&1&-1\\1&-2&-1\\1&1&-1\end{pmatrix},\mathbf{r}_8=\begin{pmatrix}1&1&1\\1&0&-1\\1&-1&-1\end{pmatrix},$$

这两种算子在保持细节和抗噪声方面都有较好的效果。

三、实验环境

Windows11

Visual Studio2021

C#语言

四、实验主要代码与效果展示

Laplacian 算子

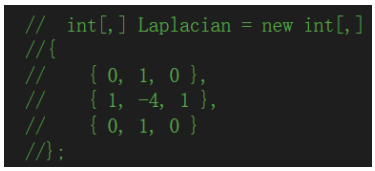
➢ 算法描述：

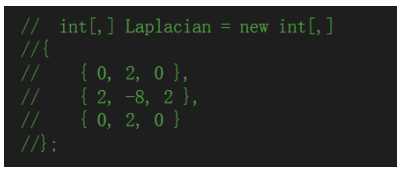
Laplacian 算子通过计算像素点周围像素的差异来检测图像中的边缘。算子模板中心的权重值为负值，而周围的权重值为正值。通过对图像进行卷积运算，将像素点与模板进行相乘并求和，得到边缘图像的像素值。

因此我定义一个 Laplacian 算子模板，该模板是一个 3x3 的矩阵，用于计算像素点周围像素的差异。然后遍历图像的每个像素点，另外在循环中定义变量 sumr、sumg、sumb 用于累加像素点周围像素与 Laplacian 算子模板的乘积，对于像素点周围的每个像素，获取其颜色值，并获取对应位

置的 Laplacian 算子模板的权重值，将当前像素的 RGB 值与对应的模板权重值相乘并累加到 sumr、sumg、sumb 中，在循环结束后创建一个新的颜色对象 edgeColor，使用范围限制后的 sumr、sumg、sumb 值创建该颜色对象。

通过以上步骤就完成了基于拉普拉斯算子的图像边缘化提取，同时拉普拉斯的算子可以任意定义，这里为了测试方便，我没有额外定义选取算子的组件，而是简单的在代码中通过注释选择，常见的拉普拉斯算子模板如下：



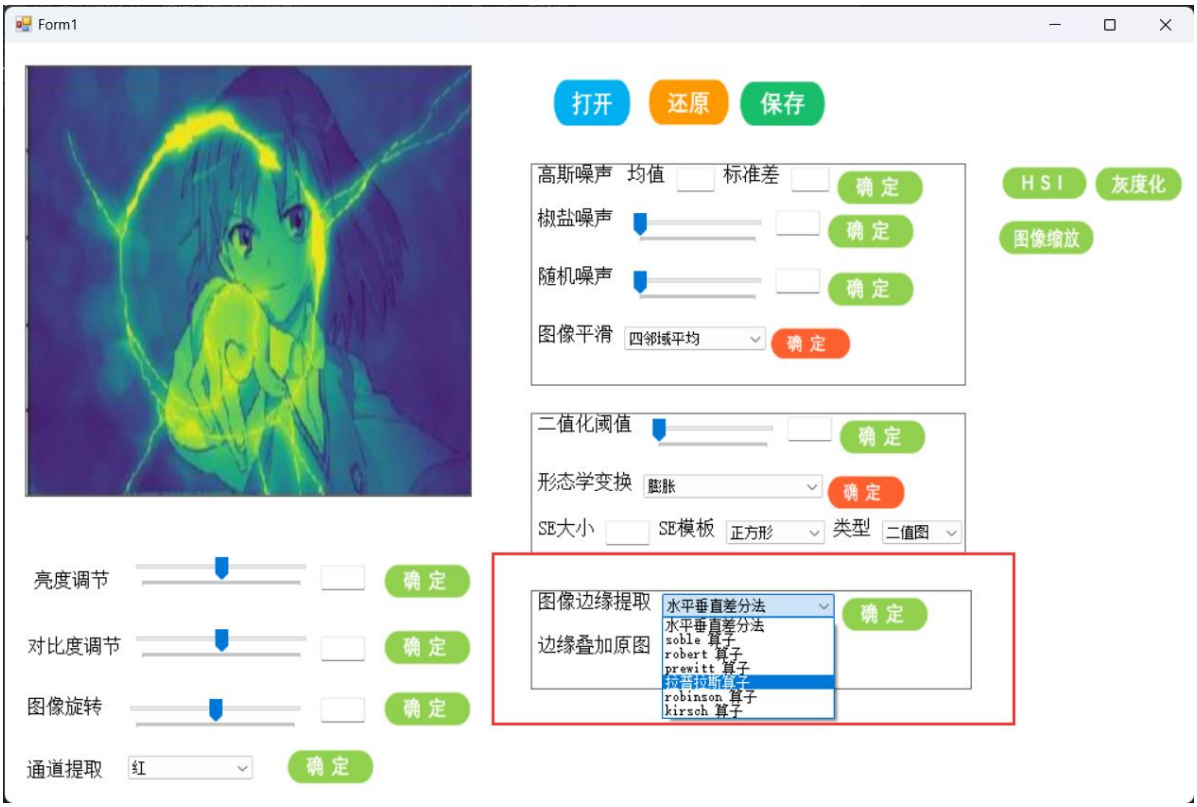



```
1.  private void image_sharpening_拉普拉斯算子(object sender, EventArgs e)
2.  {
3.      Bitmap bt1 = new Bitmap(original_image.Image);
4.      Bitmap sharpened = new Bitmap(pictureBox_show.Image);
5.
6.      // 创建 Laplacian 算子滤波器
7.      int[,] Laplacian = new int[,]
8.      {
9.        { 1, 1, 1 },
10.       { 1, -8, 1 },
11.       { 1, 1, 1 }
12.     };
13.
14.     //  int[,] Laplacian = new int[,]
15.     //{
16.     //    { 0, 1, 0 },
17.     //    { 1, -4, 1 },
18.     //    { 0, 1, 0 }
19.     //};
20.
21.     //  int[,] Laplacian = new int[,]
22.     //{
23.     //    { 0, 2, 0 },
24.     //    { 2, -8, 2 },
25.     //    { 0, 2, 0 }
26.     //};
27.
28.     for (int i = 1; i < bt1.Width - 1; i++)
29.     {
30.         for (int j = 1; j < bt1.Height - 1; j++)
31.         {
32.             int sumr = 0, sumg = 0, sumb = 0;
33.             Color pixelColor;
34.             for (int k = -1; k <= 1; k++)
35.             {
36.                 for (int l = -1; l <= 1; l++)
```

```
                {
                    pixelColor = bt1.GetPixel(i + k, j + l);
                    int LaplacianValue = Laplacian[k + 1, l + 1];

                    sumr += pixelColor.R * LaplacianValue;
                    sumg += pixelColor.G * LaplacianValue;
                    sumb += pixelColor.B * LaplacianValue;
                }
            }
            int resultR = Math.Max(0, Math.Min(255, sumr));
            int resultG = Math.Max(0, Math.Min(255, sumg));
            int resultB = Math.Max(0, Math.Min(255, sumb));

            Color edgeColor = Color.FromArgb(resultR, resultG, resultB);
            sharpened.SetPixel(i, j, edgeColor);
            //pictureBox_show.Refresh();
        }
    }
    pictureBox_show.Image = sharpened;
}
```

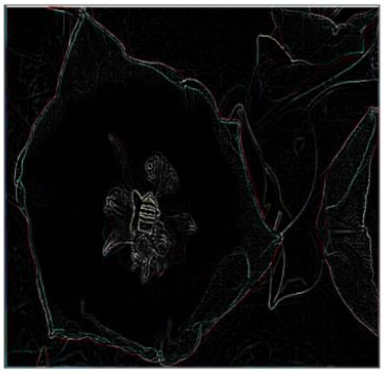
➢ **演示效果：**

可视化展示：



不同的图像边缘化对比：

原图



Laplacian算子边缘提取

叠加原图实现锐化效果

应用的滤波器模板：

```
// 创建Laplacian算子滤波器
int[,] Laplacian = new int[,]
{
    { 1, 1, 1 },
    { 1, -8, 1 },
    { 1, 1, 1 }
};
```

原图

Laplacian算子边缘提取

叠加原图实现锐化效果

应用的滤波器模板：

```
int[,] Laplacian = new int[,]
{
    { 0, 2, 0 },
    { 2, -8, 2 },
    { 0, 2, 0 }
};
```

Sobel 算子

➢ 算法描述：

Sobel 算子使用两个 3x3 的模板（一个用于检测水平边缘，一个用于检测垂直边缘），分别对图像进行卷积运算，然后将两个方向上的边缘值进行平方和开方得到最终的边缘强度。

因此我需要定义两个 Sobel 算子模板 sobelX 和 sobelY，分别用于检测水平和垂直边缘，通常有两种 Sobel 算子滤波器：

```
// 创建Sobel算子滤波器
//  int[,] sobelX = new int[,]
//{
//    { -1, 0, 1 },
//    { -2, 0, 2 },
//    { -1, 0, 1 }
//};
//  int[,] sobelY = new int[,]
//  {
//    { -1, -2, -1 },
//    { 0, 0, 0 },
//    { 1, 2, 1 }
//  };
```

```
int[,] sobelX = new int[,]
{
{ -3, 0, 3 },
{ -10, 0, 10 },
{ -3, 0, 3 }
};

int[,] sobelY = new int[,]
{
{ -3, -10, -3 },
{ 0, 0, 0 },
{ 3, 10, 3 }
};
```

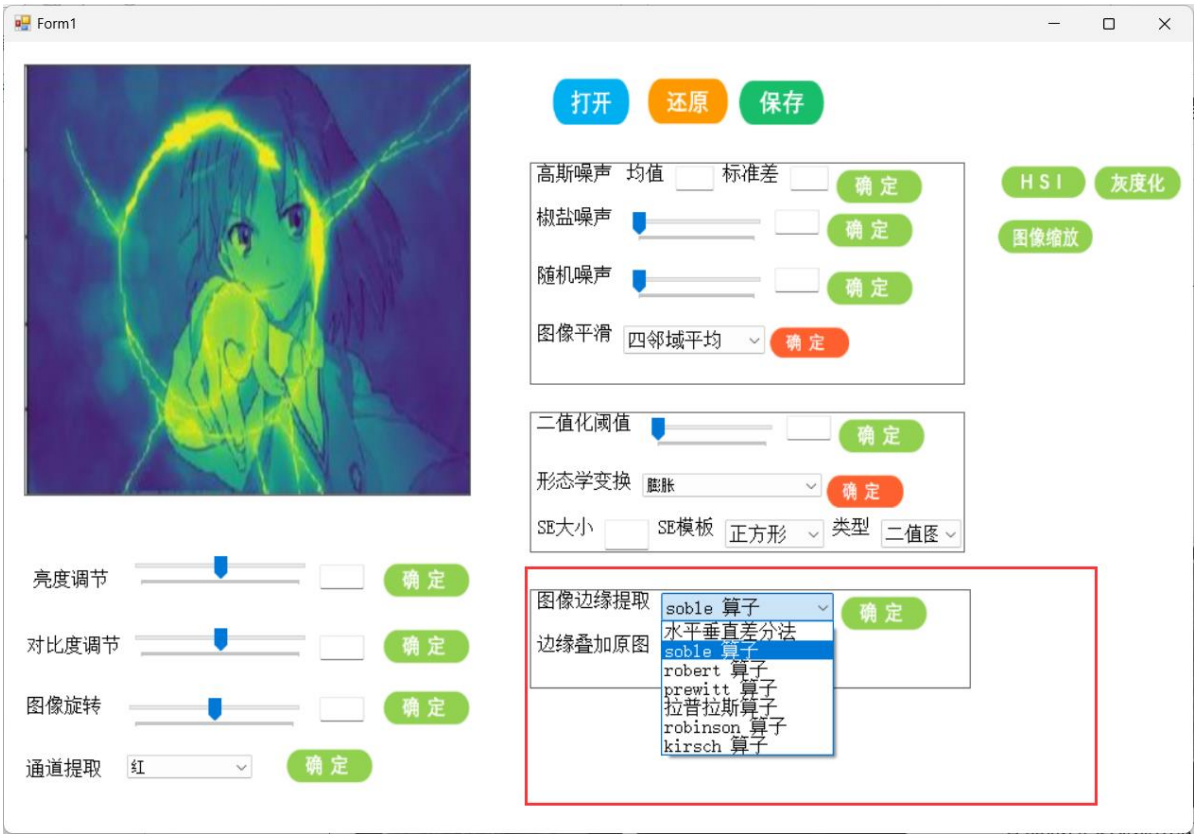
之后就同拉普拉斯算子边缘化的处理方式，对于像素点周围的每个像素，获取其颜色值，并获取对应位置的 Sobel 算子模板的权重值，再计算边缘强度，使用绝对值的方式计算水平和垂直方向上的边缘值，并将两个方向上的边缘值相加得到最终的边缘强度。

```
1.  private void image_sharpening_soble 算子(object sender, EventArgs e)
2.  {
3.      Bitmap bt1 = new Bitmap(pictureBox_show.Image);
4.      Bitmap sharpened = new Bitmap(pictureBox_show.Image);
5.
6.      // 创建 Sobel 算子滤波器
7.      //  int[,] sobelX = new int[,]
8.      //{
9.      //    { -1, 0, 1 },
10.     //    { -2, 0, 2 },
11.     //    { -1, 0, 1 }
12.     //};
13.     //  int[,] sobelY = new int[,]
14.     //  {
15.     //    { -1, -2, -1 },
16.     //    { 0, 0, 0 },
17.     //    { 1, 2, 1 }
18.     //  };
19.
20.     int[,] sobelX = new int[,]
21.     {
22.     { -3, 0, 3 },
23.     { -10, 0, 10 },
24.     { -3, 0, 3 }
25.     };
26.
27.     int[,] sobelY = new int[,]
28.     {
29.     { -3, -10, -3 },
30.     { 0, 0, 0 },
31.     { 3, 10, 3 }
32.     };
33.
34.     for (int i = 1; i < bt1.Width - 1; i++)
35.     {
```

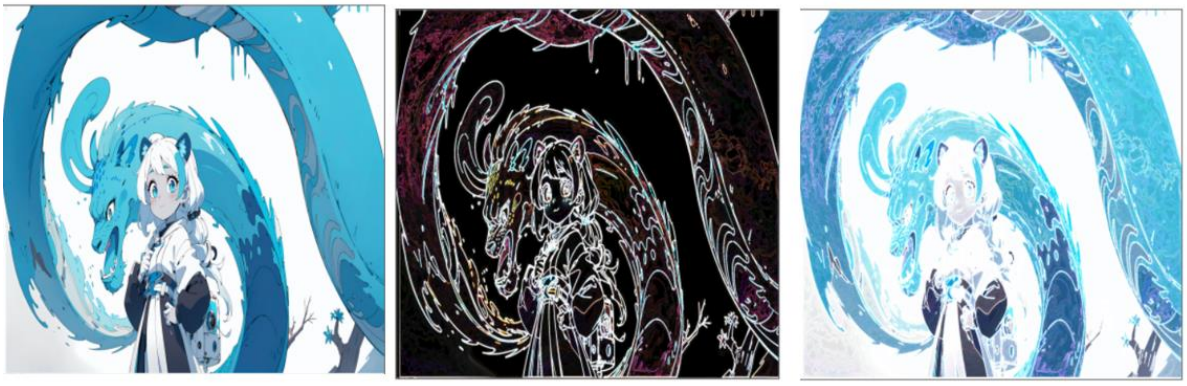
```
        for (int j = 1; j < bt1.Height - 1; j++)
        {
            int sumXr = 0, sumXg = 0, sumXb = 0;
            int sumYr = 0, sumYg = 0, sumYb = 0;

            for (int k = -1; k <= 1; k++)
            {
                for (int l = -1; l <= 1; l++)
                {
                    Color pixelColor = bt1.GetPixel(i + k, j + l);
                    int sobelXValue = sobelX[k + 1, l + 1];
                    int sobelYValue = sobelY[k + 1, l + 1];

                    sumXr += pixelColor.R * sobelXValue;
                    sumXg += pixelColor.G * sobelXValue;
                    sumXb += pixelColor.B * sobelXValue;

                    sumYr += pixelColor.R * sobelYValue;
                    sumYg += pixelColor.G * sobelYValue;
                    sumYb += pixelColor.B * sobelYValue;
                }
            }
            //double gradientR = Math.Sqrt(sumXr * sumXr + sumYr * sumYr);
            //double gradientG = Math.Sqrt(sumXg * sumXg + sumYg * sumYg);
            //double gradientB = Math.Sqrt(sumXb * sumXb + sumYb * sumYb);

            int gradientR = Math.Max(0, Math.Min(255, Math.Abs(sumXr) + Math.Abs(s
umYr)));
            int gradientG = Math.Max(0, Math.Min(255, Math.Abs(sumXg) + Math.Abs(s
umYg)));
            int gradientB = Math.Max(0, Math.Min(255, Math.Abs(sumXb) + Math.Abs(s
umYb)));

            Color edgeColor = Color.FromArgb(gradientR, gradientG, gradientB);
            sharpened.SetPixel(i, j, edgeColor);
            //pictureBox_show.Refresh();
        }
    }
    pictureBox_show.Image = sharpened;
}
```

- **演示效果：**

可视化展示：



测试图：



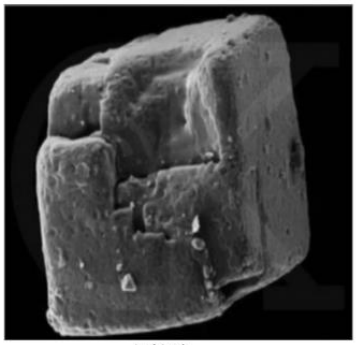
原图　　Sobel算子边缘提取　　叠加原图实现锐化效果

应用的滤波器模板：

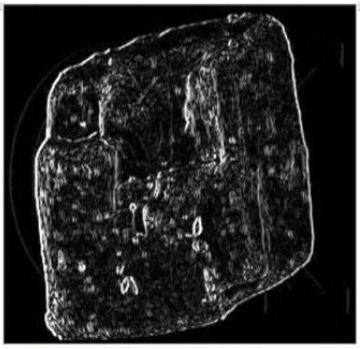
```
// 创建Sobel算子滤波器
int[,] sobelX = new int[,]
{
    { -1,  0,  1 },
    { -2,  0,  2 },
    { -1,  0,  1 }
};
```

```
int[,] sobelY = new int[,]
{
    { -1, -2, -1 },
    {  0,  0,  0 },
    {  1,  2,  1 }
};
```

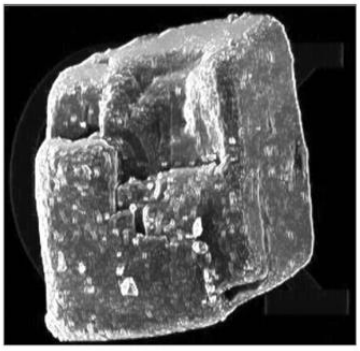
对于显微镜下的食盐晶体，用 **Sobel 算子**和**拉普拉斯算子**锐化图像的结果对比图如下：



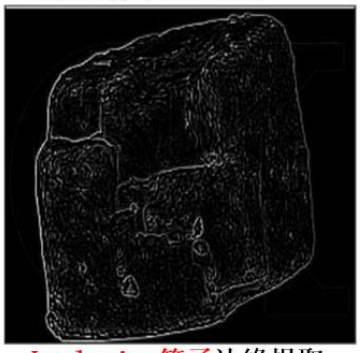
原图



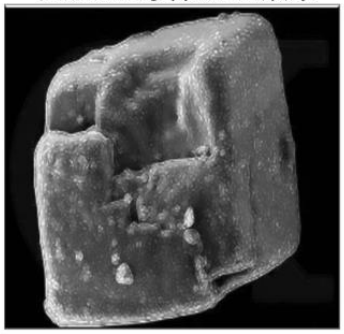
Sobel算子边缘提取



叠加原图实现锐化效果



Laplacian算子边缘提取

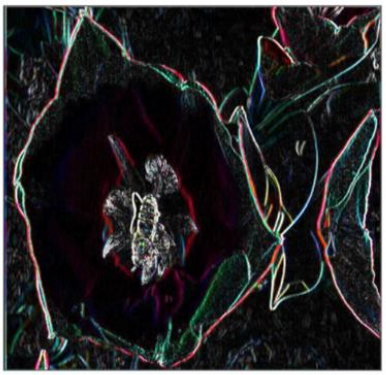


当内核大小为 3 时,Sobel 内核可能产生比较明显的误差，为解决这一问题，我们使用 Scharr 函数，但该函数仅作用于大小为 3 的内核。该函数的运算与 Sobel 函数一样快，但结果却更加精确，其计算方法为:

$$H_X = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \qquad H_Y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

scharr 算子和 sobel 的原理一致，就是 Gx 和 Gy 参数的大小不同，也就是卷积核中各元素的权不同，其他都一样，scharr 算子对于边界的梯度计算效果更精确。如下对比图:

原图



Sobel算子边缘提取

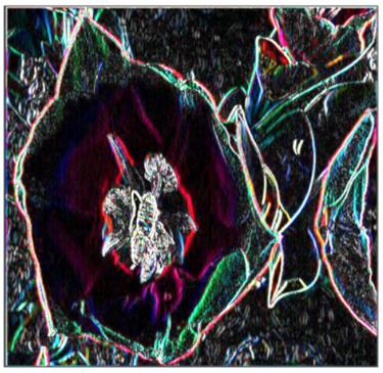
叠加原图实现锐化效果

应用的滤波器模板：

```
// 创建Sobel算子滤波器
int[,] sobelX = new int[,]
{
    { -1,  0,  1 },
    { -2,  0,  2 },
    { -1,  0,  1 }
};
```

```
int[,] sobelY = new int[,]
{
    { -1, -2, -1 },
    {  0,  0,  0 },
    {  1,  2,  1 }
};
```

原图



Sobel算子边缘提取

叠加原图实现锐化效果

应用的滤波器模板：

```
int[,] sobelX = new int[,]
{
    { -3,  0,  3 },
    { -10,  0,  10 },
    { -3,  0,  3 }
};
```

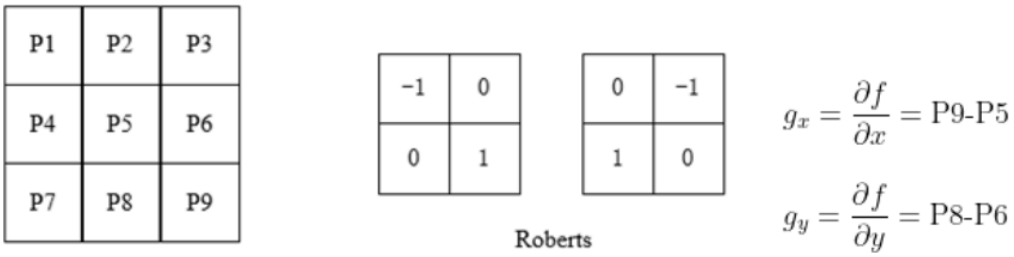
```
int[,] sobelY = new int[,]
{
    { -3, -10, -3 },
    {  0,  0,  0 },
    {  3,  10,  3 }
};
```

Roberts 算子

➢ 算法描述：

Roberts 算子又称为交叉微分算法，它是基于交叉差分的梯度算法，通过局部差分计算检测边缘线条。常用来处理具有陡峭的低噪声图像，当图像边缘接近于正 45 度或负 45 度时，该算法处理效果更理想。其缺点是对边缘的定位不太准确，提取的边缘线条较粗。

Roberts 算子的模板分为水平方向和垂直方向，如下式所示，从其模板可以看出，Roberts 算子能较好的增强正负 45 度的图像边缘。



该算子是 2*2 的模板，因此可以直接在循环中遍历当前像素点的右下方三个像素，即根据算子模板得到当前像素点及其相邻像素的颜色值（**像素 5、像素 6、像素 8、像素 9**）。然后计算**水平**方向上的颜色差异（像素 9 与像素 5 的颜色差值）和**垂直**方向上的颜色差异（像素 8 与像素 6 的颜色差值），并将两个方向上的边缘值相加得到最终的边缘强度。

```
1.    private void image_sharpening_robert 算子(object sender, EventArgs e)
2.    {
3.        Bitmap originalBt = new Bitmap(pictureBox_show.Image);
4.        Bitmap edgeBt = new Bitmap(originalBt.Width, originalBt.Height);
5.
6.        int[,] robertsX = { { -1, 0 }, { 0, 1 } }; // Roberts 算子水平方向模板
7.        int[,] robertsY = { { 0, -1 }, { 1, 0 } }; // Roberts 算子垂直方向模板
8.
9.        int diffXr = 0, diffXg = 0, diffXb = 0;
10.       int diffYr = 0, diffYg = 0, diffYb = 0;
11.       for (int i = 0; i < originalBt.Width - 1; i++)
12.       {
13.           for (int j = 0; j < originalBt.Height - 1; j++)
14.           {
15.               Color pixel5 = originalBt.GetPixel(i, j);
16.               Color pixel6 = originalBt.GetPixel(i, j + 1);
17.               Color pixel8 = originalBt.GetPixel(i + 1, j);
18.               Color pixel9 = originalBt.GetPixel(i + 1, j + 1);
19.
20.               diffXr = pixel9.R - pixel5.R;
21.               diffXg = pixel9.G - pixel5.G;
22.               diffXb = pixel9.B - pixel5.B;
23.
24.               diffYr = pixel8.R - pixel6.R;
25.               diffYg = pixel8.G - pixel6.G;
26.               diffYb = pixel8.B - pixel6.B;
27.
28.
29.               int gradientR = Math.Max(0, Math.Min(255, Math.Abs(diffXr)+ Math.Abs
(diffYr)));
30.               int gradientG = Math.Max(0, Math.Min(255, Math.Abs(diffXr)+ Math.Abs
(diffYg)));
31.               int gradientB = Math.Max(0, Math.Min(255, Math.Abs(diffXr)+ Math.Abs
(diffYb)));
32.
33.               edgeBt.SetPixel(i, j, Color.FromArgb(gradientR, gradientG, gradientB
));
34.           }
```

```
35.         }
36.         pictureBox_show.Refresh(); // 刷新图片框
37.         pictureBox_show.Image = edgeBt;
38.     }
```

➢ **实现效果：**

可视化组件展示：

原图

Sobel算子边缘提取

叠加原图实现锐化效果

Prewitt 算子

➢ 算法描述：

由于 Prewitt 算子采用 3×3 模板对区域内的像素值进行计算，而 Robert 算子的模板为 2×2，故 Prewitt 算子的边缘检测结果在水平方向和垂直方向均比 Robert 算子更加明显。Prewitt 算子适合用来识别噪声较多、灰度渐变的图像，其计算公式如下所示：

P1	P2	P3
P4	P5	P6
P7	P8	P9

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

$$g_x = \frac{\partial f}{\partial x} = (P7+P8+P9)-(P1+P2+P3)$$

$$g_y = \frac{\partial f}{\partial y} = (P3+P6+P9)-(P1+P4+P7)$$

算法编程中，Prewitt 算子的实现过程与 Roberts 算子比较相似，由于其使用了 3*3 的模板，因此我仿照 Sobel 算子定义了算子模板如下，其具

体实现过程也和上述算子类似：

```
    //prewitt算子滤波器模板
    int[,] prewittY = new int[,]
    {
{ -1, 0, 1 },
{ -1, 0, 1 },
{ -1, 0, 1 }
    };
```

```
    int[,] prewittX = new int[,]
    {
{ -1, -1, -1 },
{ 0, 0, 0 },
{ 1, 1, 1 }
    };
```

```
1.  private void image_sharpening_prewitt 算子(object sender, EventArgs e)
2.  {
3.      Bitmap bt1 = new Bitmap(pictureBox_show.Image);
4.      Bitmap sharpened = new Bitmap(pictureBox_show.Image);
5.
6.      //prewitt 算子滤波器模板
7.      int[,] prewittY = new int[,]
8.      {
9.  { -1, 0, 1 },
10. { -1, 0, 1 },
11. { -1, 0, 1 }
12.     };
13.
14.     int[,] prewittX = new int[,]
15.     {
16. { -1, -1, -1 },
17. { 0, 0, 0 },
18. { 1, 1, 1 }
19.     };
20.
21.     for (int i = 1; i < bt1.Width - 1; i++)
22.     {
23.         for (int j = 1; j < bt1.Height - 1; j++)
24.         {
25.             int sumXr = 0, sumXg = 0, sumXb = 0;
26.             int sumYr = 0, sumYg = 0, sumYb = 0;
27.
28.             for (int k = -1; k <= 1; k++)
29.             {
30.                 for (int l = -1; l <= 1; l++)
31.                 {
32.                     Color pixelColor = bt1.GetPixel(i + k, j + l);
33.                     int prewittXValue = prewittX[k + 1, l + 1];
34.                     int prewittYValue = prewittY[k + 1, l + 1];
35.
36.                     sumXr += pixelColor.R * prewittXValue;
37.                     sumXg += pixelColor.G * prewittXValue;
38.                     sumXb += pixelColor.B * prewittXValue;
39.
40.                     sumYr += pixelColor.R * prewittYValue;
41.                     sumYg += pixelColor.G * prewittYValue;
42.                     sumYb += pixelColor.B * prewittYValue;
43.                 }
44.             }
45.
46.             int gradientR = Math.Max(0, Math.Min(255, Math.Abs(sumXr) + Math.Abs(s
    umYr)));
```

```
47.             int gradientG = Math.Max(0, Math.Min(255, Math.Abs(sumXg) + Math.Abs(s
    umYg)));
48.             int gradientB = Math.Max(0, Math.Min(255, Math.Abs(sumXb) + Math.Abs(s
    umYb)));
49.
50.             Color edgeColor = Color.FromArgb(gradientR, gradientG, gradientB);
51.             sharpened.SetPixel(i, j, edgeColor);
52.         }
53.     }
54.     pictureBox_show.Image = sharpened;
55.
56. }
57.
```

- **实现效果：**

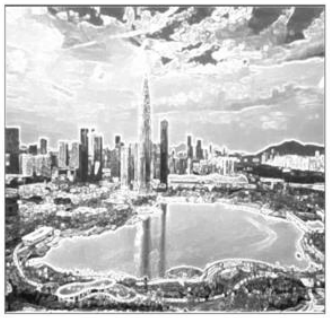
由下图可以看出 **Prewitt 算子**的边缘检测结果在**水平**方向和**垂直**方向均比 **Robert 算子**更加明显：

原图

Robert算子边缘提取

叠加原图实现锐化效果

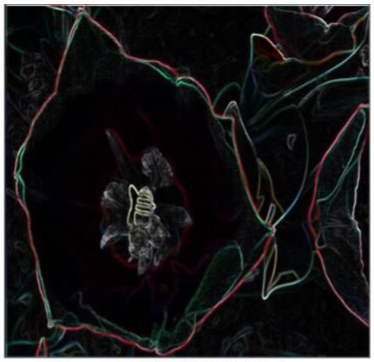
Prewitt算子边缘提取



叠加原图实现锐化效果

再测试几组：

原图



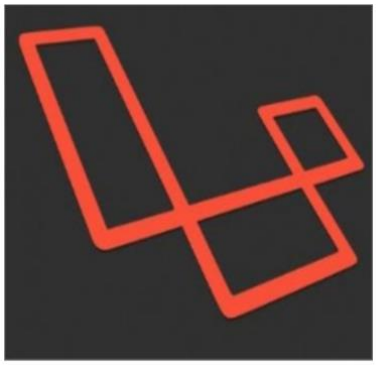
Prewitt算子边缘提取

叠加原图实现锐化效果

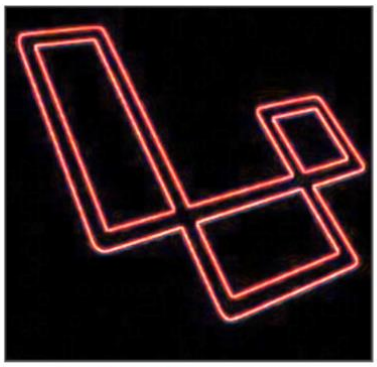
应用的滤波器模板：

```
//prewitt算子滤波器模板
int[,] prewittX = new int[,]
{
{ -1, 0, 1 },
{ -1, 0, 1 },
{ -1, 0, 1 }
};
```

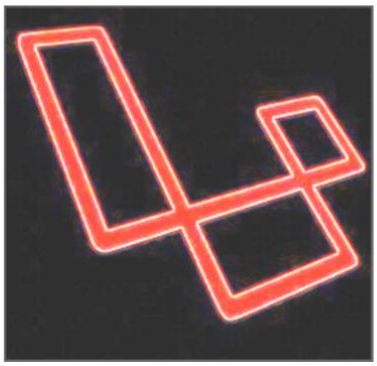
```
int[,] prewittY = new int[,]
{
{ -1, -1, -1 },
{ 0, 0, 0 },
{ 1, 1, 1 }
};
```



原图



Prewitt算子边缘提取



叠加原图实现锐化效果

应用的滤波器模板：

```
//prewitt算子滤波器模板
int[,] prewittX = new int[,]
{
{ -1, 0, 1 },
{ -1, 0, 1 },
{ -1, 0, 1 }
};
```

```
int[,] prewittY = new int[,]
{
{ -1, -1, -1 },
{ 0, 0, 0 },
{ 1, 1, 1 }
};
```

Krisch 算子

➢ 算法描述：

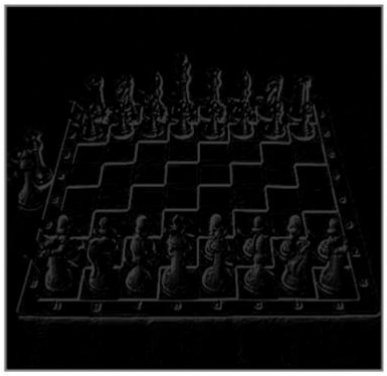
Kirsch 算子是一种基于卷积的边缘检测算法，它采用 8 个 3*3 的模板对图像进行卷积，这 8 个模板代表 8 个方向，并取最大值作为图像的边缘输出，使用了**八个**不同的模板来检测图像中的边缘。由于需要循环八次，我这里为了将每个颜色通道的计算结果限制在 0 到 255 之间，进行了**归一化处理**，经过调参数测试，这里将**结果除以 4** 得到的边缘较为清晰。最后使用归一化后的颜色通道值创建新的 Color 对象 edgeColor 并赋值即可。

```
1.    private void image_sharpening_kirsch 算子(object sender, EventArgs e)
2.        {
3.            Bitmap bt1 = new Bitmap(pictureBox_show.Image);
4.            Bitmap edgeImage = new Bitmap(bt1.Width, bt1.Height);
5.
6.            int[,,] krischTemplates = {
7.      {{-3, -3, 5}, {-3, 0, 5}, {-3, -3, 5}},
8.    {{-3, 5, 5}, {-3, 0, 5}, {-3, -3, -3}},
9.    {{5, 5, 5}, {-3, 0, -3}, {-3, -3, -3}},
10.   {{5, 5, -3}, {5, 0, -3}, {-3, -3, -3}},
11.   {{5, -3, -3}, {5, 0, -3}, {5, -3, -3}},
12.   {{-3, -3, -3}, {5, 0, -3}, {5, 5, -3}},
13.   {{-3, -3, -3}, {-3, 0, -3}, {5, 5, 5}},
14.   {{-3, -3, -3}, {-3, 0, 5}, {-3, 5, 5}}};
15.           // 对每个像素应用 Kirsch 算子
16.           for (int i = 1; i < bt1.Width - 1; i++)
17.           {
18.               for (int j = 1; j < bt1.Height - 1; j++)
19.               {
20.                   int maxGradientR = 0;
21.                   int maxGradientG = 0;
22.                   int maxGradientB = 0;
23.
24.                   // 对每个模板进行卷积计算
25.                   for (int t = 0; t < 8; t++)
26.                   {
27.                       int gradientR = 0;
28.                       int gradientG = 0;
29.                       int gradientB = 0;
30.
31.                       // 计算模板与图像区域的卷积
32.                       for (int m = -1; m <= 1; m++)
33.                       {
34.                           for (int n = -1; n <= 1; n++)
35.                           {
36.                               gradientR += krischTemplates[t, m + 1, n + 1] * bt1.
GetPixel(i + m, j + n).R;
37.                               gradientG += krischTemplates[t, m + 1, n + 1] * bt1.
GetPixel(i + m, j + n).G;
38.                               gradientB += krischTemplates[t, m + 1, n + 1] * bt1.
GetPixel(i + m, j + n).B;
39.                           }
40.                       }
41.
42.                       int SumR = Math.Max(0, Math.Min(255, gradientR));
43.                       int SumG = Math.Max(0, Math.Min(255, gradientG));
44.                       int SumB = Math.Max(0, Math.Min(255, gradientB));
45.                       SumR /= 4; SumG /= 4; SumB /=4;
46.
47.                       Color edgeColor = Color.FromArgb(SumR, SumG, SumB);
48.                       edgeImage.SetPixel(i, j, edgeColor);
49.                   }
50.               }
51.           }
52.           pictureBox_show.Refresh(); // 刷新图片框
```

```
53.             pictureBox_show.Image = edgeImage;
54.         }
```

- 实现效果：

原图



Kirsch算子边缘提取

叠加原图实现锐化效果

原图

Kirsch算子边缘提取

叠加原图实现锐化效果

Robinson 算子

- 算法描述：

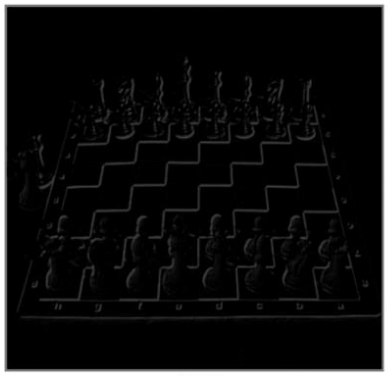
原理同 Krisch 算子。

```
1.   private void image_sharpening_robinson 算子(object sender, EventArgs e)
2.       {
3.           Bitmap bt1 = new Bitmap(pictureBox_show.Image);
4.           Bitmap edgeImage = new Bitmap(bt1.Width, bt1.Height);
5.
6.           int[,,] robinsonTemplates = {
7.  {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}},
8.  {{0, 1, 2}, {-1, 0, 1}, {-2, -1, 0}},
9.  {{1, 2, 1}, {0, 0, 0}, {-1, -2, -1}},
10. {{2, 1, 0}, {1, 0, -1}, {0, -1, -2}},
11. {{1, 0, -1}, {2, 0, -2}, {1, 0, -1}},
12. {{0, -1, -2}, {1, 0, -1}, {2, 1, 0}},
13. {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}},
14. {{-2, -1, 0}, {-1, 0, 1}, {0, 1, 2}}
15.
```

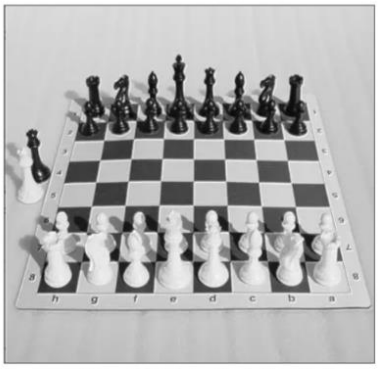
```
        // 对每个像素应用 Kirsch 算子
        for (int i = 1; i < bt1.Width - 1; i++)
        {
            for (int j = 1; j < bt1.Height - 1; j++)
            {
                int maxGradientR = 0;
                int maxGradientG = 0;
                int maxGradientB = 0;

                // 对每个模板进行卷积计算
                for (int t = 0; t < 8; t++)
                {
                    int gradientR = 0;
                    int gradientG = 0;
                    int gradientB = 0;

                    // 计算模板与图像区域的卷积
                    for (int m = -1; m <= 1; m++)
                    {
                        for (int n = -1; n <= 1; n++)
                        {
                            gradientR += robinsonTemplates[t, m + 1, n + 1] * bt
1.GetPixel(i + m, j + n).R;
                            gradientG += robinsonTemplates[t, m + 1, n + 1] * bt
1.GetPixel(i + m, j + n).G;
                            gradientB += robinsonTemplates[t, m + 1, n + 1] * bt
1.GetPixel(i + m, j + n).B;
                        }
                    }

                    int SumR = Math.Max(0, Math.Min(255, gradientR));
                    int SumG = Math.Max(0, Math.Min(255, gradientG));
                    int SumB = Math.Max(0, Math.Min(255, gradientB));
                    SumR /= 4; SumG /= 4; SumB /= 4;

                    Color edgeColor = Color.FromArgb(SumR, SumG, SumB);
                    edgeImage.SetPixel(i, j, edgeColor);
                }
            }
        }
        pictureBox_show.Refresh(); // 刷新图片框
        pictureBox_show.Image = edgeImage;

    }
```

- 实现效果：

原图



Robinson算子边缘提取



叠加原图实现锐化效果

原图

Robinson算子边缘提取

叠加原图实现锐化效果

- **各类算子实验比较**
  - 对比效果:

原图

Prewitt算子边缘提取

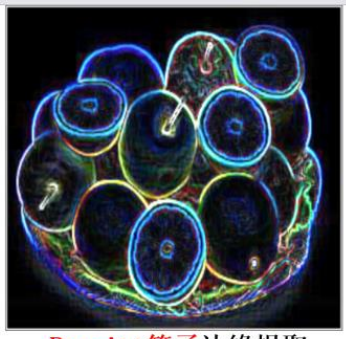
Sobel算子边缘提取

Robert算子边缘提取

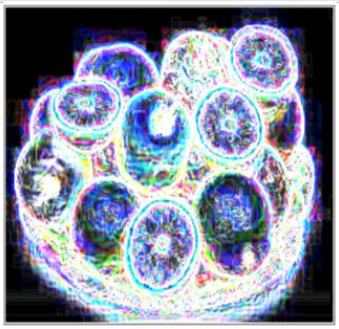
Laplacian算子边缘提取

为了比较不同算子，多测试了几张图像，如下图所示：

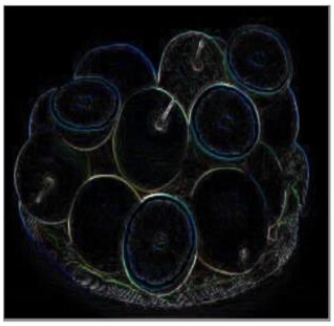
原图



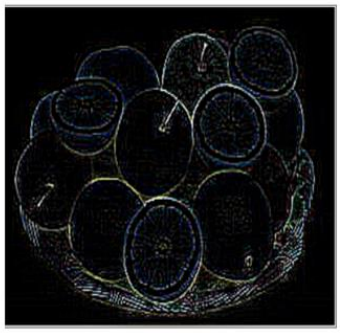
Prewitt算子边缘提取



Sobel算子边缘提取

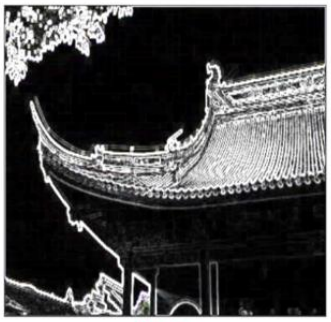


Robert算子边缘提取

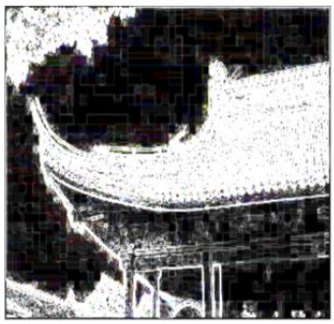


Laplacian算子边缘提取

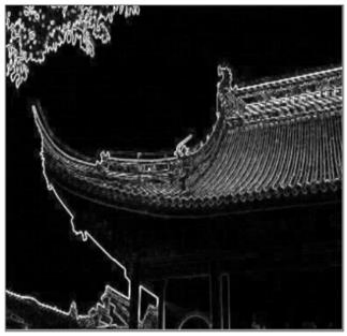
原图



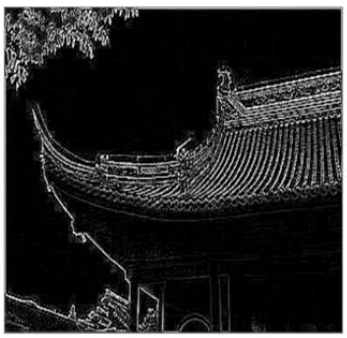
Prewitt算子边缘提取



Sobel算子边缘提取



Robert算子边缘提取



Laplacian算子边缘提取

由上面的结果所示，不同的算子进行了比较。可知：

1）Robert 算子对陡峭的低噪声图像效果较好，尤其是边缘正负 45 度较多的图像，但定位准确率较差；

2）Prewitt 算子对灰度渐变的图像边缘提取效果较好，而没有考虑相邻点的距离远近对当前像素点的影响；

3）Sobel 算子考虑了综合因素，对噪声较多的图像处理效果更好。

4）Laplacian 算子对噪声比较敏感，由于其算法可能会出现双像素边界，常用来判断边缘像素位于图像的明区或暗区，很少用于边缘检测；

五、实验结果及分析(包括心得体会，本部分为重点，不能抄袭复制)

- **完成情况：**

完成了 soble 算子，robert 算子，prewitt 算子，Laplacian 算子，robinson 算子，kirsch 算子对图像的**边缘提取**和叠加原图实现**锐化**，并比较了不同算子之间的差异性和彼此的优缺点

- **实验心得**

在这次图像处理的实验中，我通过研究不同算子的原理和实现方法，完成了对图像的**边缘提取**和叠加原图实现**锐化**的任务。同时我也遇到一些

困难，例如理解**不同算子的原理**、**算法实现的复杂性**、**图像处理的性能**等方面的挑战。并且如果参数不正确，也会导致边缘检测结果不准确、图像锐化效果不理想等。**为了克服困难**，我仔细研究了每个算子的原理和实现细节，参考学习通的资料，根据不同的算子模板编写了自己的边缘提取代码，并逐步优化代码和参数设置。

这次图像锐化虽然是单一的一个部分，但是里面涉及到的预备知识很多，像微积分，梯度相关的知识都很重要。总之这次实验提高了我的图像处理技能，还培养了我解决问题的能力和实验研究的经验，以及更了解数学原理和图像处理该如何结合应用。