图像边缘检测

电子信息工程系

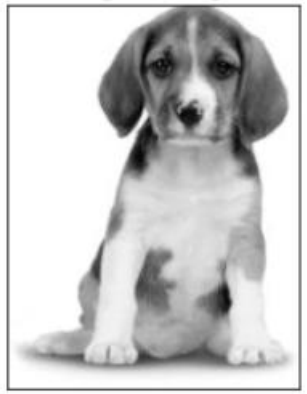
袁羽

目录

CONTENTS

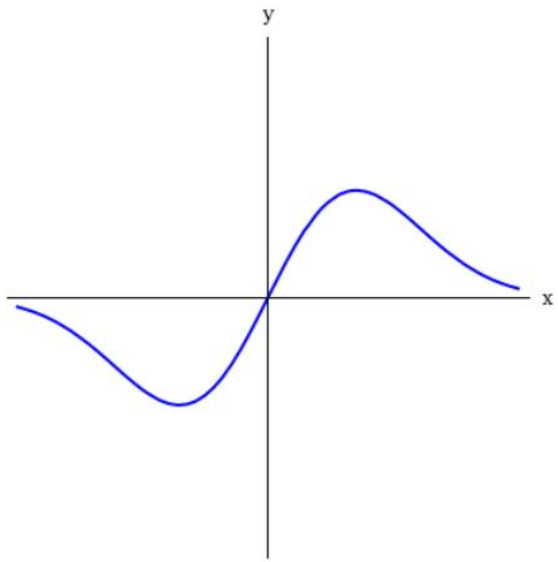
一 图像边缘检测

边缘检测目的是检测识别出图像中亮度变化剧烈的像素点构成的集合。图像边缘的正确检测对于分析图像中的内容、实现图像中物体的分割、定位等具有重要的作用。边缘检测大大减少了源图像的数据量，剔除了与目标不相干的信息，保留了图像重要的结构属性。



一 图像边缘检测

图像的边缘指的是图像中像素灰度值突然发生变化的区域，如果将图像的每一行像素和每一列像素都描述成一个关于灰度值的函数，那么图像的边缘对应在灰度值函数中是函数值突然变大的区域。函数值的变化趋势可以用函数的导数描述。当函数值突然变大时，导数也必然会变大，而函数值变化较为平缓区域，导数值也比较小，因此可以通过寻找导数值较大的区域去寻找函数中突然变化的区域，进而确定图像中的边缘位置。



一 图像边缘检测

图像梯度，可以简单地理解为像素的变化程度。如果几个连续的像素，其像素值跨度越大，则梯度值越大。 图像梯度可以把图像看成二维离散函数，图像梯度其实就是这个二维离散函数的求导。图像梯度运算公式： G(x,y) = dx(i,j) + dy(i,j);

dx(i,j) = I(i+1,j) - I(i,j);

dy(i,j) = I(i,j+1) - I(i,j);

其中，I是图像像素的值(如：RGB值)，(i,j)为像素的坐标。

图像梯度一般也可以用中值差分：

dx(i,j) = [I(i+1,j) - I(i-1,j)]/2;

dy(i,j) = [I(i,j+1) - I(i,j-1)]/2;

图像边缘一般都是通过对图像进行梯度运算来实现的。

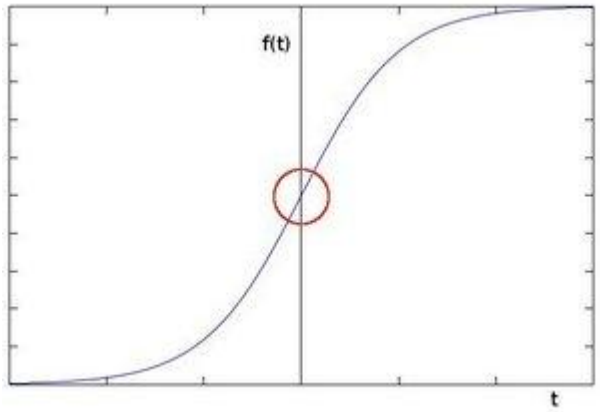
二 一阶微分算子

基于搜索的边缘检测方法：首先计算边缘强度，通常用一阶导数表示，例如梯度模，然后，计算估计边缘的局部方向，通常采用梯度的方向，并利用此方向找到局部梯度模的最大值。

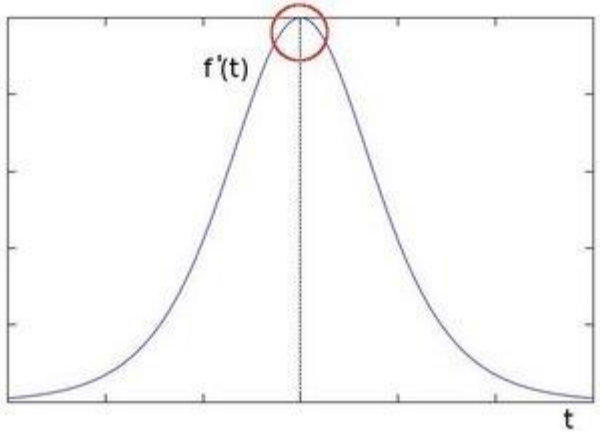
一阶微分为基础的边缘检测，通过计算图像的梯度值来检测图像的边缘，代表算法是Sobel算子和Scharr算子。

二 一阶微分算子

1.Sobel算子：Sobel算子是高斯平滑与微分操作的结合体，所以其抗噪声能力很好。



像素随着变量t而发生变化



求取亮度变化的一阶导数

一阶微分算子 利用图像在边缘处的阶跃性，即图像梯度在边缘处取得极大值的特性来进行边缘检测。对于一幅二维的数字图像而言，它需要完成x与y两个方向上的微分，分别求出x和y两个方向上的偏微分，最终得到的梯度是一个矢量，具有方向与模。

二 一阶微分算子

1.Sobel算子：

Sobel算子使用3×3的卷积核来获得近似导数，使用一个卷积核检测在X方向（水平方向）的像素亮度突变， 使用另外一个卷积核检测Y方向（垂直方向）的亮度突变 。

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

用于检测X方向的卷积核

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

用于检测Y方向的卷积核

二 一阶微分算子

1.Sobel算子：进行sobel边缘检测的的方法：

dst = cv2.Sobel(src, ddepth, dx, dy, ksize, scale, delta, borderType)

参数：

src：应用Sobel算子的图像数据，可以为多通道彩色图像，但实际应用中往往使用单通道灰度图像作为输入检测物体边缘；

ddepth：图像深度，往往使用-1，表示输出图像数据类型与输入图像一致；

dx和dy: 指求导的阶数，0表示这个方向上没有求导，取值为0、1；

ksize：必须为大于1的奇数，默认值为3，表示Sobel卷积核的大小。该值越大，对越细的边缘产生响应，3往往足够使用；

scale：double类型，默认值为1，对卷积计算的结果进行缩放的倍数；

delta：double类型，默认值为0，卷积计算结果会再加上该值作为最后的输出值；

borderType：边界外推类型，默认值为cv2.BORDER_DEFAULT，实质上就是镜像边界

二 一阶微分算子

1.Sobel算子：

注意：

- 不能让dx和dy同时为0，应该分两步分别求两个方向的导数。

- 函数求完导数后会有负值，还有会大于255的值。而原图像是uint8，即8位无符号数，所以Sobel建立的图像位数不够，会有截断。因此要将图像深度设置的更大一点，如cv2.CV_16S，cv2.CV_32F，cv2.CV_64F等，处理完图像后，再使用cv2.convertScaleAbs()函数将其转回原来的uint8格式，否则图像无法显示。

```
Scale_abs = cv2.convertScaleAbs(x) # 格式转换函数
```

- Sobel算子是在两个方向计算的，最后还需要将其组合起来。

```
result = cv2.addWeighted(src1, alpha, src2, beta) # 图像混合
```

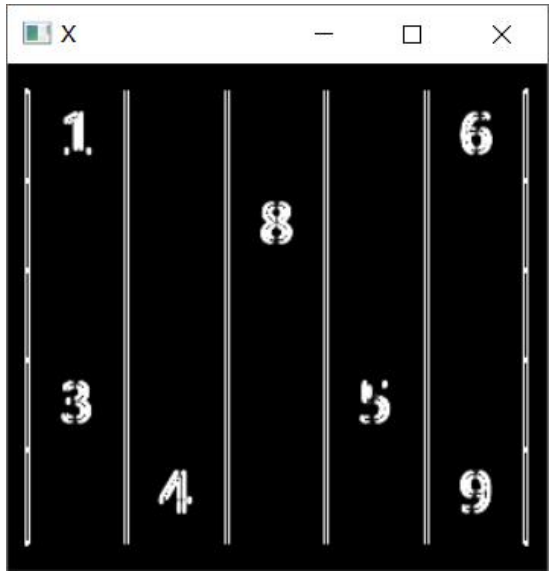
或 result = cv2.add（src1，src2）

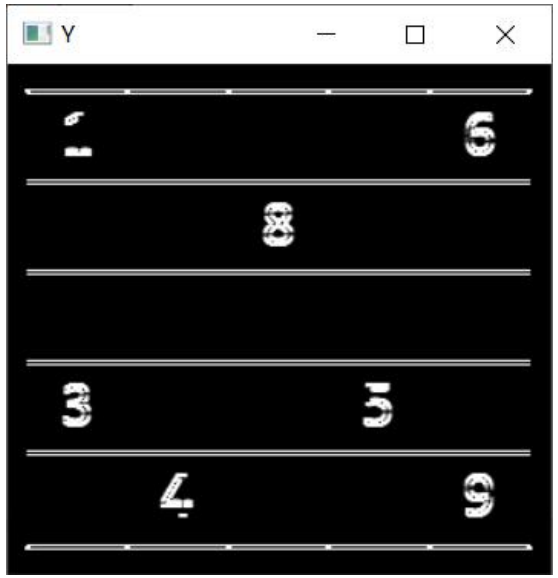
例　利用Sobel算子检测图像边缘。

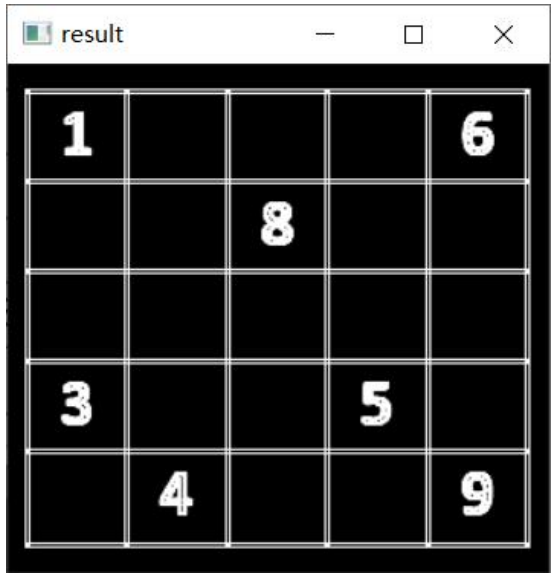
1				6
		8		
3			5	
	4			9

例　利用Sobel算子检测图像边缘。

1				6
		8		
3			5	
	4			9



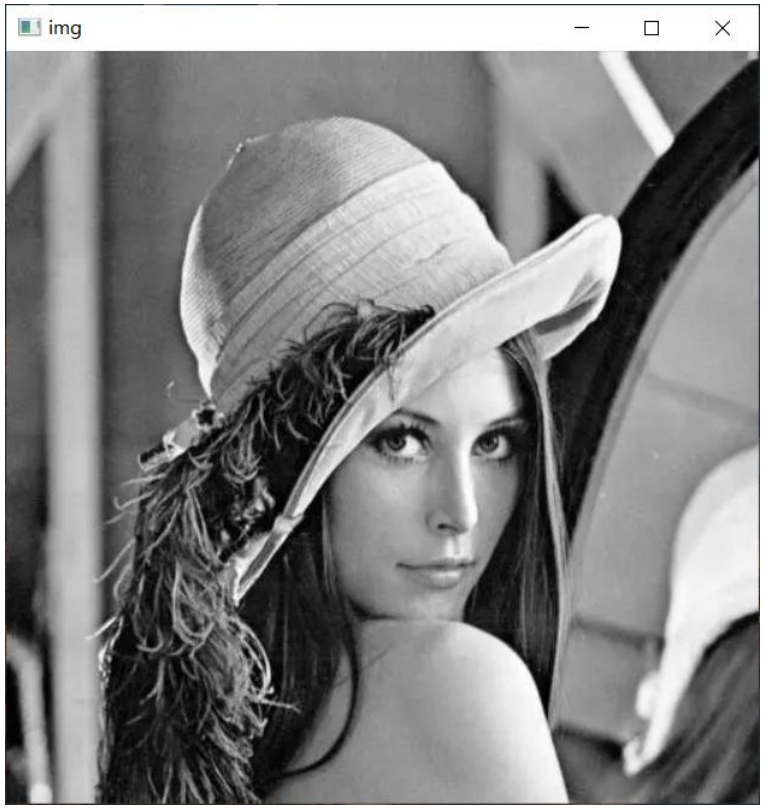


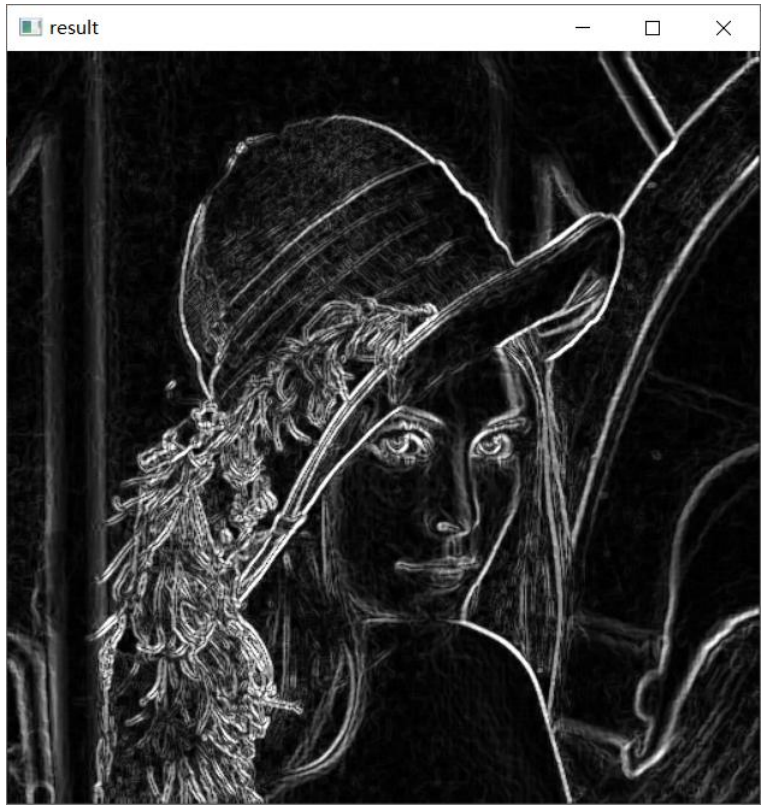


例 利用Sobel算子检测图像边缘。

```
import cv2
import numpy as np
# 1 读取图像
img = cv2.imread('table.png',0)
# 2 计算Sobel卷积结果
x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
y = cv2.Sobel(img, cv2.CV_16S, 0, 1)
# 3 将数据进行转换
X = cv2.convertScaleAbs(x)
Y = cv2.convertScaleAbs(y)
# 4 结果合成
result = cv2.add(X,Y)
# result = cv2.addWeighted(X, 0.5, Y, 0.5, 0)
cv2.imshow('img',img)
cv2.imshow("X",X)
cv2.imshow("Y",Y)
cv2.imshow('result',result)
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

例　利用Sobel算子检测图像边缘。





例 利用Sobel算子检测图像边缘。

```
import cv2
import numpy as np
img = cv2.imread('lenna.jpg',0)
# 2 计算Sobel卷积结果
x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
y = cv2.Sobel(img, cv2.CV_16S, 0, 1)
# 3 将数据进行转换
X = cv2.convertScaleAbs(x)
Y = cv2.convertScaleAbs(y)
# 4 结果合成
result = cv2.addWeighted(X, 0.5, Y, 0.5, 0)
cv2.imshow('img',img)
cv2.imshow('result',result)
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

二 一阶微分算子

2.Scharr算子：

Scharr算子是Sobel算子的改进版，同样是计算图像差分来近似一阶导数的卷积核。大小固定为3 × 3 。相比Sobel算子，Scharr算子可以检测更微弱的边缘。

dst = cv2.Scharr(src, ddepth, dx, dy, scale, delta, borderType)

参数：

src：应用Scharr算子的图像数据，仅可使用单通道灰度图像；

ddepth：图像深度，往往使用-1，表示输出图像数据类型与输入图像一致；

dx：int类型，表示在x方向上计算差分的阶次，往往使用Scharr计算一阶差分；

dy：int类型，表示在y方向上计算差分的阶次，往往使用Scharr计算一阶差分；

scale：double类型，默认值为1，对卷积计算的结果进行缩放的倍数；

delta：double类型，默认值为0，卷积计算结果会再加上该值作为最后的输出值；

borderType：边界外推类型，默认值为cv2.BORDER_DEFAULT；

二 一阶微分算子

2.Scharr算子:

当Sobel算子设定的卷积核的大小为-1时，会默认使用3x3的Scharr滤波器。

例如:

```
x = cv2.Sobel(img, cv2.CV_16S, 1, 0, ksize = -1)
y = cv2.Sobel(img, cv2.CV_16S, 0, 1, ksize = -1)
```

三 Laplacian算子

Laplacian算子主要用于计算图像的二阶梯度。Laplacian算子计算图像二阶梯度时则具有旋转不变性，即不需要考虑方向，调用时可以检测出各个方向上的边缘。

常用算子模块：

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

三 Laplacian算子

OpenCV中提供了Laplacian（）函数，语法如下：

dst = cv2.Laplacian(src, ddepth, ksize, scale, delta, borderType)

src：应用Laplacian算子的图像数据，可以为多通道彩色图像，但实际应用中往往使用单通道灰度图像作为输入检测物体边缘；

ddepth：int类型，输出图像的数据类型，往往使用-1，表示输出图像数据类型与输入图像一致；

ksize：int类型，必须为正奇数，默认值为1，该值越大，对越细的边缘产生响应；

scale：double类型，默认值为1，对卷积计算的结果进行缩放的倍数；

delta：double类型，默认值为0，卷积计算结果会再加上该值作为最后的输出值；

borderType：边界外推类型，默认值为cv2.BORDER_DEFAULT，实质上就是镜像边界；

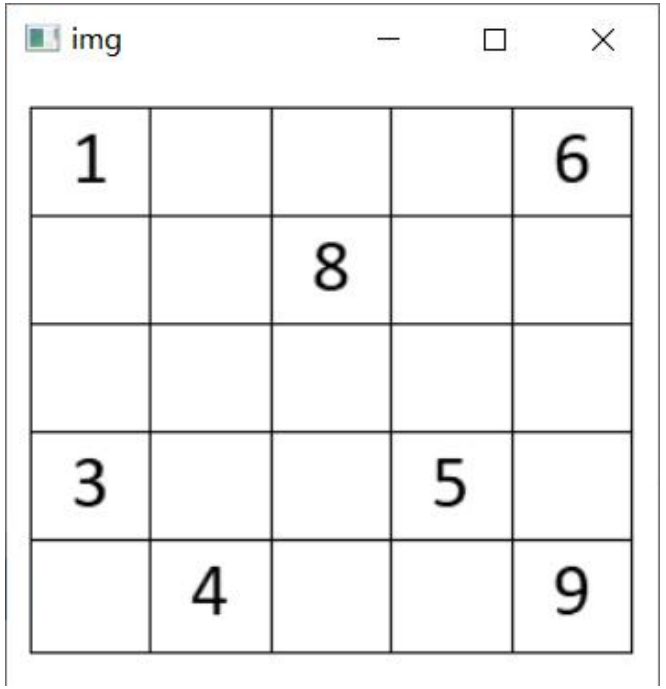
例：利用Laplacian算子对图像进行边缘检测

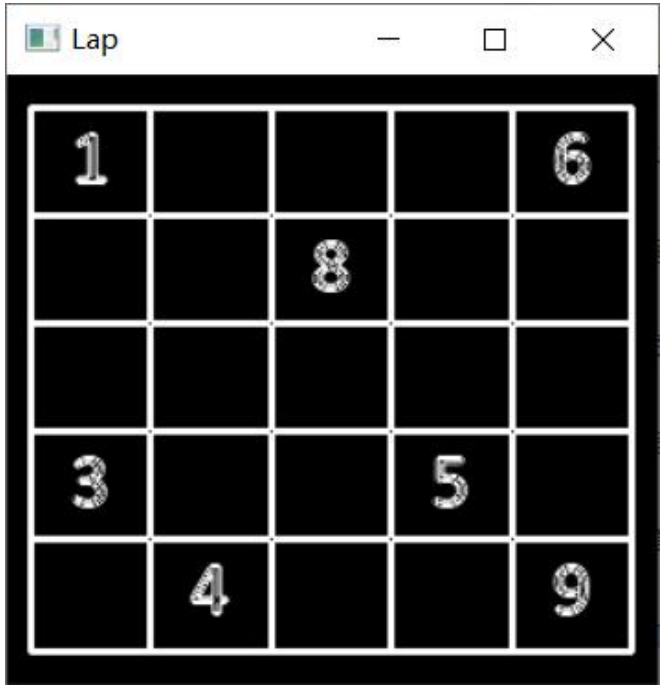
1				6
		8		
3			5	
	4			9

例：利用Laplacian算子对图像进行边缘检测

```
import cv2
img = cv2.imread('table.png',0)
Lap= cv2.Laplacian(img,cv2.CV_64F,ksize=1)
Lap = cv2.convertScaleAbs(Lap)
cv2.imshow("img",img)
cv2.imshow("Lap",Lap)
cv2.waitKey()
cv2.destroyAllWindows()
```

例：利用Laplacian算子对图像进行边缘检测





四 Canny边缘检测

Canny边缘检测是当今最流行的边缘检测算法，这是因为它具有可靠性和灵活性。Canny算法就具有四个处理阶段：

1. 消除噪声；
2. 计算图像的亮度梯度值；
3. 减除虚假边缘；
4. 带有滞回的阈值检测；

四 Canny边缘检测

Canny边缘检测是当今最流行的边缘检测算法，这是因为它具有可靠性和灵活性。Canny算法就具有四个处理阶段：

1. 消除噪声；

原始图像的像素通常会产生边缘噪声，所以Canny边缘检测在计算边缘之前进行噪声去除非常重要。使用高斯模糊化可以去除大部分，或者尽量减少不必要的图像细节产生不必要的边缘。

四 Canny边缘检测

Canny边缘检测是当今最流行的边缘检测算法，这是因为它具有可靠性和灵活性。Canny算法就具有四个处理阶段：

2. 计算图像的亮度梯度值；

在图像平滑后，通过调用Sobel水平和垂直卷积核对图像进行滤波。

四 Canny边缘检测

Canny边缘检测是当今最流行的边缘检测算法，这是因为它具有可靠性和灵活性。Canny算法就具有四个处理阶段：

3. 减除虚假边缘；

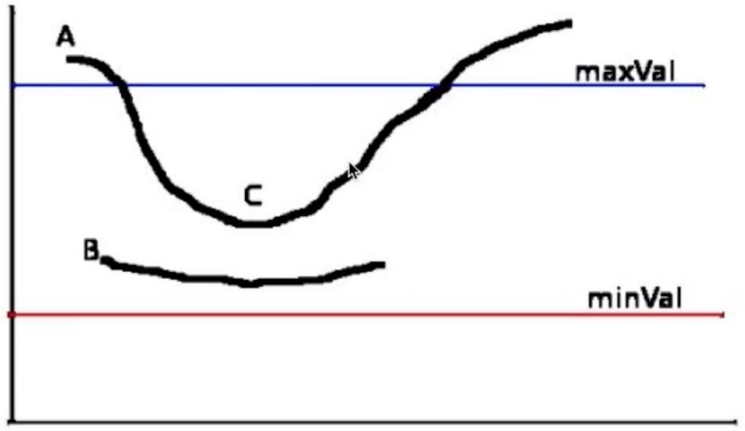
完成了图像中的噪声去除和亮度梯度计算，算法的这个步骤通过使用 non-maximum supperssion （非极大值抑制）来剔除不需要的像素（这些像素不是组成边缘的部分）。 通过比较每个像素与周围像素在水平和垂直两个方向上的梯度值来实现剔除虚假边缘。如果一个像素对应的梯度在局部是最大的，也就是比他的上下左右像素梯度都大，它就保留下来。否则将就该像素置为0。

四 Canny边缘检测

Canny边缘检测是当今最流行的边缘检测算法，这是因为它具有可靠性和灵活性。Canny算法就具有四个处理阶段：

4. 带有滞回的阈值检测；

确定真正的边界，设置两个阈值: minVal 和maxVal。当图像的灰度梯度高于maxVal时被认为是真的边界，低于minVal的边界会被抛弃。对于那些梯度值落在两个给定阈值范围之内的像素他们被标记为弱边缘（也就是作为最终边缘图的候选者），如果弱边缘与强边缘相连，他们最终会保留在边缘图中。



四 Canny边缘检测

OpenCV中Canny边缘检测函数语法：

dst=cv2.Canny(src, threshold1, threshold2)

参数说明：

src:原始图像数据

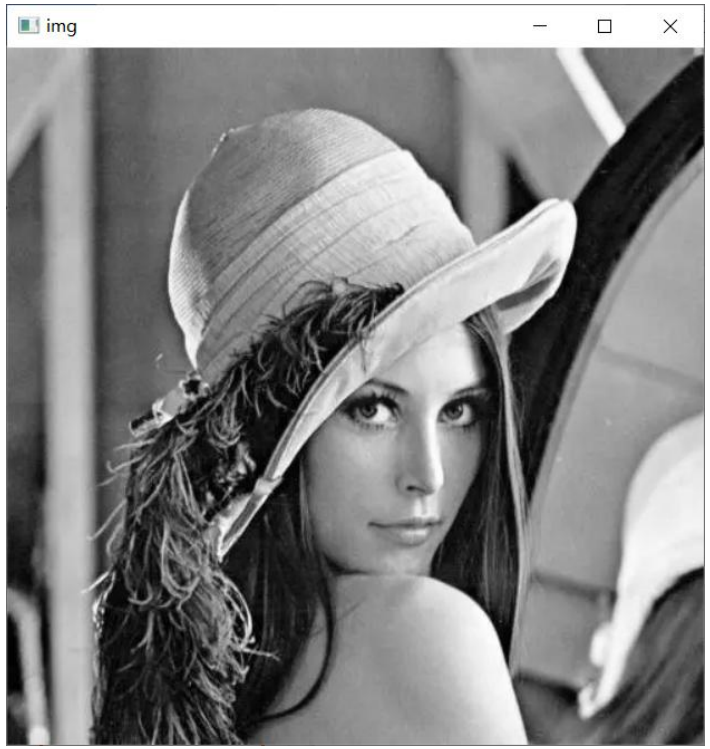
threshold1:较小的阈值minval

threshold2:较大的阈值maxval

例 利用Canny边缘检测函数对图像进行边缘检测

```
import cv2
img = cv2.imread('lenna.jpg',0)
cany= cv2.Canny(img,50,100)
cv2.imshow("img",img)
cv2.imshow("cany",cany)
cv2.waitKey()
cv2.destroyAllWindows()
```

例 利用Canny边缘检测函数对图像进行边缘检测



小结

您的正文已经经简明扼要，字字珠玑，但信息却千丝万缕、错综复杂，需要用更多的文字来表述；但请您尽可能提炼思想的精髓，否则容易造成观者的阅读压力，适得其反。

THANKS