

该系列文章是讲解Python OpenCV图像处理知识，前期主要讲解图像入门、OpenCV基础用法，中期讲解图像处理的各种算法，包括图像锐化算子、图像增强技术、图像分割等，后期结合深度学习研究图像识别、图像分类应用。希望文章对您有所帮助，如果有不足之处，还请海涵~

该系列在github所有源代码：<https://github.com/eastmountyxz/ImageProcessing-Python>
PS：请求帮忙点个Star，哈哈，第一次使用Github，以后会分享更多代码，一起加油。

同时推荐作者的C++图像系列知识：

[数字图像处理] 一.MFC详解显示BMP格式图片

[数字图像处理] 二.MFC单文档分割窗口显示图片

[数字图像处理] 三.MFC实现图像灰度、采样和量化功能详解

[数字图像处理] 四.MFC对话框绘制灰度直方图

[数字图像处理] 五.MFC图像点运算之灰度线性变化、灰度非线性变化、阈值化和均衡化处理详解

[数字图像处理] 六.MFC空间几何变换之图像平移、镜像、旋转、缩放详解

[数字图像处理] 七.MFC图像增强之图像普通平滑、高斯平滑、Laplacian、Sobel、Prewitt锐化详解

前文参考：

[Python图像处理] 一.图像处理基础知识及OpenCV入门函数

[Python图像处理] 二.OpenCV+Numpy库读取与修改像素

[Python图像处理] 三.获取图像属性、兴趣ROI区域及通道处理

[Python图像处理] 四.图像平滑之均值滤波、方框滤波、高斯滤波及中值滤波

[Python图像处理] 五.图像融合、加法运算及图像类型转换

[Python图像处理] 六.图像缩放、图像旋转、图像翻转与图像平移

[Python图像处理] 七.图像阈值化处理及算法对比

[Python图像处理] 八.图像腐蚀与图像膨胀

[Python图像处理] 九.形态学之图像开运算、闭运算、梯度运算

[Python图像处理] 十.形态学之图像顶帽运算和黑帽运算

[Python图像处理] 十一.灰度直方图概念及OpenCV绘制直方图

[Python图像处理] 十二.图像几何变换之图像仿射变换、图像透视变换和图像校正

[Python图像处理] 十三.基于灰度三维图的图像顶帽运算和黑帽运算

[Python图像处理] 十四.基于OpenCV和像素处理的图像灰度化处理

[Python图像处理] 十五.图像的灰度线性变换

[Python图像处理] 十六.图像的灰度非线性变换之对数变换、伽马变换

[Python图像处理] 十七.图像锐化与边缘检测之Roberts算子、Prewitt算子、Sobel算子和Laplacian算子

由于收集图像数据的器件或传输数图像的通道的一些质量缺陷，文物图像时间久远，或者受一些其他外界因素、动态不稳定抓取图像的影响，使得图像存在模糊和有噪

声的情况，从而影响到图像识别工作的开展。这时需要开展图像锐化和边缘检测处理，加强原图像的高频部分，锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰。

图像锐化和边缘提取技术可以消除图像中的噪声，提取图像信息中用来表征图像的一些变量，为图像识别提供基础。通常使用灰度差分法对图像的边缘、轮廓进行处理，将其凸显。前文分别采用Laplacian算子、Robert算子、Prewitt算子和Sobel算子进行图像锐化边缘处理实验，本文将继续讲解Scharr算子、Canny算子和LOG算子。

本文主要讲解灰度线性变换，基础性知识希望对您有所帮助。

1.Scharr算子

2.Canny算子

3.LOG算子

4.总结代码

注意：该部分知识均为自己查阅资料撰写，转载请署名CSDN+杨秀璋及原地址出处，谢谢！！

PS：文章参考自己以前系列图像处理文章及OpenCV库函数，同时参考如下文献：

杨秀璋等. 基于苗族服饰的图像锐化和边缘提取技术研究[J]. 现代计算机, 2018(10).

《数字图像处理》（第3版），冈萨雷斯著，阮秋琦译，电子工业出版社，2013年.

《数字图像处理学》（第3版），阮秋琦，电子工业出版社，2008年，北京.

《OpenCV3编程入门》，毛星云，冷雪飞，电子工业出版社，2015.

张小洪，杨丹，刘亚威. 基于Canny算子的改进型边缘检测算法[J]. 计算机工程与应用, 2003

[数字图像处理] 七.MFC图像增强之图像普通平滑、高斯平滑、Laplacian、Sobel、Prewitt锐化详解

图像边缘检测——一阶微分算子 Roberts、Sobel、Prewitt、Kirsch、Robinson (Matlab实现)

[OpenCV图像处理入门学习教程四] 基于LoG算子的图像边缘检测 primetong

https://en.wikipedia.org/wiki/Canny_edge_detector

一.Scharr算子

由于Sobel算子在计算相对较小的核的时候，其近似计算导数的精度比较低，比如一个33的Sobel算子，当梯度角度接近水平或垂直方向时，其不精确性就越发明显。Scharr算子同Sobel算子的速度一样快，但是准确率更高，尤其是计算较小核的情景，所以利用33滤波器实现图像边缘提取更推荐使用Scharr算子。

Scharr算子又称为Scharr滤波器，也是计算x或y方向上的图像差分，在OpenCV中主要是配合Sobel算子的运算而存在的，其滤波器的滤波系数如下：

$$d_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad d_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

Scharr算子的函数原型如下所示，和Sobel算子几乎一致，只是没有ksize参数，其函数原型如下所示：

dst = Scharr(src, ddepth, dx, dy[, dst[, scale[, delta[, borderType]]]])

- src表示输入图像
- dst表示输出的边缘图，其大小和通道数与输入图像相同
- ddepth表示目标图像所需的深度，针对不同的输入图像，输出目标图像有不同的深度
- dx表示x方向上的差分阶数，取值1或0
- dy表示y方向上的差分阶数，取值1或0
- scale表示缩放导数的比例常数，默认情况下没有伸缩系数
- delta表示将结果存入目标图像之前，添加到结果中的可选增量值
- borderType表示边框模式，更多详细信息查阅BorderTypes

Python实现代码如下所示：

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('lena.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Scharr算子
x = cv2.Scharr(grayImage, cv2.CV_32F, 1, 0) #X方向
y = cv2.Scharr(grayImage, cv2.CV_32F, 0, 1) #Y方向
absX = cv2.convertScaleAbs(x)
```

```
absY = cv2.convertScaleAbs(y)
Scharr = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = [u'原始图像', u'Scharr算子']
images = [lenna_img, Scharr]
for i in xrange(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))
plt.show()
```

运行结果如下图所示：



二.Canny算子

John F.Canny于1986年发明了一个多级边缘检测算法——Canny边缘检测算子，并创立了边缘检测计算理论（Computational theory of edge detection），该理论有效地解释了这项技术的工作理论。

边缘检测通常是在保留原有图像属性的情况下，对图像数据规模进行缩减，提取图像边缘轮廓的处理方式。

Canny算法是一种被广泛应用于边缘检测的标准算法，其目标是找到一个最优的边缘检测解或找寻一幅图像中灰度强度变化最强的位置。最优边缘检测主要通过低错误率、高定位性和最小响应三个标准进行评价。Canny算子的实现步骤如下：

1.使用高斯平滑（如公式所示）去除噪声。

$$K(5,5) = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

2.按照Sobel滤波器步骤计算梯度幅值和方向，寻找图像的强度梯度。先将卷积模板分别作用x和y方向，再计算梯度幅值和方向，其公式如下所示。梯度方向一般取0度、45度、90度和135度四个方向。

$$d_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad d_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

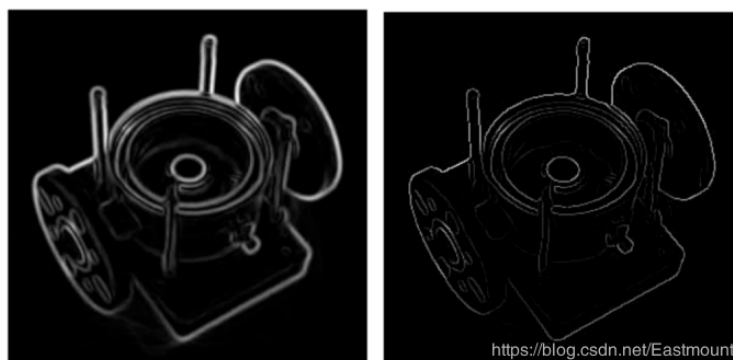
$$S = \sqrt{(d_x(i, j))^2 + (d_y(i, j))^2}$$

$$\theta = \arctan\left(\frac{d_y}{d_x}\right)$$

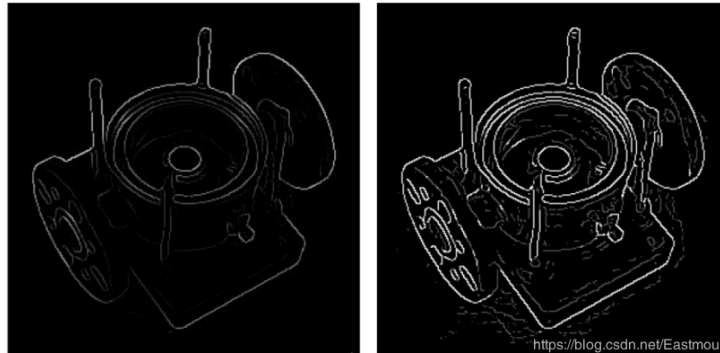
3.通过非极大值抑制 (Non-maximum Suppression) 过滤掉非边缘像素，将模糊的边界变得清晰。该过程保留了每个像素点上梯度强度的极大值，过滤掉其他的值。

对于每个像素点，它进行如下操作：

- 1) 将其梯度方向近似为以下值中的一个，包括0、45、90、135、180、225、270和315，即表示上下左右和45度方向。
- 2) 比较该像素点和其梯度正负方向的像素点的梯度强度，如果该像素点梯度强度最大则保留，否则抑制（删除，即置为0）。其处理后效果如下图所示，左边表示梯度值，右边表示非极大值抑制处理后的边缘。



4.利用双阈值方法来确定潜在的边界。经过非极大抑制后图像中仍然有很多噪声点，此时需要通过双阈值技术处理，即设定一个阈值上界和阈值下界。图像中的像素点如果大于阈值上界则认为必然是边界（称为强边界，strong edge），小于阈值下界则认为必然不是边界，两者之间的则认为是候选项（称为弱边界，weak edge）。经过双阈值处理的图像如下图所示，左边为非极大值抑制处理后的边缘，右边为双阈值技术处理的效果图。



5.利用滞后技术来跟踪边界。若某一像素位置和强边界相连的弱边界认为是边界，其他的弱边界则被删除。

在OpenCV中，Canny()函数原型如下所示：

```
edges = Canny(image, threshold1, threshold2[, edges[, apertureSize[,
L2gradient]]])
```

- image表示输入图像
- edges表示输出的边缘图，其大小和类型与输入图像相同
- threshold1表示第一个滞后性阈值
- threshold2表示第二个滞后性阈值
- apertureSize表示应用Sobel算子的孔径大小，其默认值为3
- L2gradient表示一个计算图像梯度幅值的标识，默认值为false

Canny算子的边缘提取实现代码如下所示：

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('lena.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```



```
#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#高斯滤波降噪
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0)

#Canny算子
Canny = cv2.Canny(gaussian, 50, 150)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = [u'原始图像', u'Canny算子']
images = [lenna_img, Canny]
for i in xrange(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))
plt.show()
```

其运行结果如图所示：



三.LOG算子

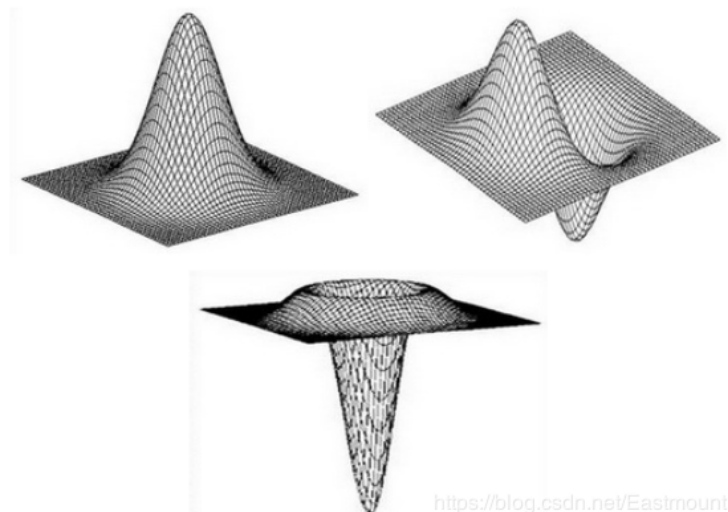
LOG (Laplacian of Gaussian) 边缘检测算子是David Courtnay Marr和Ellen Hildreth在1980年共同提出的，也称为Marr & Hildreth算子，它根据图像的信噪比来求检测边缘的最优滤波器。该算法首先对图像做高斯滤波，然后再求其拉普拉斯 (Laplacian) 二阶导数，根据二阶导数的过零点来检测图像的边界，即通过检测滤波结果的零交叉 (Zero crossings) 来获得图像或物体的边缘。

LOG算子该综合考虑了对噪声的抑制和对边缘的检测两个方面，并且把Gauss平滑滤波器和Laplacian锐化滤波器结合了起来，先平滑掉噪声，再进行边缘检测，所以效果会更好。该算子与视觉生理中的数学模型相似，因此在图像处理领域中得到了广泛的应用。它具有抗干扰能力强，边界定位精度高，边缘连续性好，能有效提取对比度弱的边界等特点。

常见的LOG算子是5*5模板，如下所示：

$$\begin{bmatrix} -2 & -4 & -4 & -4 & -2 \\ -4 & 0 & 8 & 0 & -4 \\ -4 & 8 & 24 & 8 & -4 \\ -4 & 0 & 8 & 0 & -4 \\ -2 & -4 & -4 & -4 & -2 \end{bmatrix}$$

由于LOG算子到中心的距离与位置加权系数的关系曲线像墨西哥草帽的剖面，所以LOG算子也叫墨西哥草帽滤波器，如图所示。



LOG算子的边缘提取实现代码如下所示：

```
# -*- coding: utf-8 -*-  
import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# 读取图像  
img = cv2.imread('lena.png')  
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```



```
#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#先通过高斯滤波降噪
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0)

#再通过拉普拉斯算子做边缘检测
dst = cv2.Laplacian(gaussian, cv2.CV_16S, ksize = 3)
LOG = cv2.convertScaleAbs(dst)

#用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']

#显示图形
titles = [u'原始图像', u'LOG算子']
images = [lenna_img, LOG]
for i in xrange(2):
    plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))
plt.show()
```

其运行结果如下图所示：



四.总结代码

边缘检测算法主要是基于图像强度的一阶和二阶导数，但导数通常对噪声很敏感，因此需要采用滤波器来过滤噪声，并调用图像增强或阈值化算法进行处理，最后再进行边缘检测。下面是采用高斯滤波去噪和阈值化处理之后，再进行边缘检测的过程，并对比了四种常见的边缘提取算法。

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('lena.png')
lenna_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 高斯滤波
gaussianBlur = cv2.GaussianBlur(grayImage, (3,3), 0)

# 阈值处理
ret, binary = cv2.threshold(gaussianBlur, 127, 255, cv2.THRESH_BINARY)

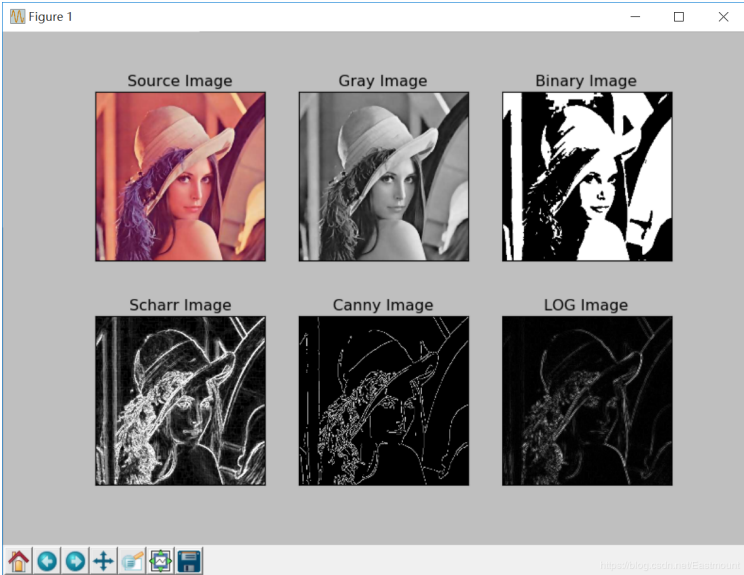
# Scharr 算子
x = cv2.Scharr(grayImage, cv2.CV_32F, 1, 0) #X方向
y = cv2.Scharr(grayImage, cv2.CV_32F, 0, 1) #Y方向
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Scharr = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

# Canny 算子
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0) #高斯滤波降噪
Canny = cv2.Canny(gaussian, 50, 150)


# LOG 算子
gaussian = cv2.GaussianBlur(grayImage, (3,3), 0) #先通过高斯滤波降噪
dst = cv2.Laplacian(gaussian, cv2.CV_16S, ksize = 3) #再通过拉普拉斯算子做边
LOG = cv2.convertScaleAbs(dst)

# 效果图
titles = ['Source Image', 'Gray Image', 'Binary Image',
          'Scharr Image', 'Canny Image', 'LOG Image']
images = [lenna_img, grayImage, binary, Scharr, Canny, LOG]
for i in np.arange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([], plt.yticks([]))
plt.show()
```


输出结果如图所示。



希望这篇基础性文章对您有所帮助，如果有错误 或不足之处，请海涵！一起加油，考博加油。



Eastmount 哈哈，苦心人，天不负。早上花了三个小时终于把滤镜效果搞出来了，纯手工双层像素遍历的Python图像处理(不调包)，继续探索。现在小学生都开始学编程了，不过挺开发大脑的，以后我也教教他编程做游戏🤖🤖
PS: 某个加班人的照片被我玩坏喽，你是像牛麦哈，看书做题了😁



查看全部15张照片

<https://blog.csdn.net/Eastmount>

(By: Eastmount 2019-04-06 下午4点写于贵阳·钟书阁
<https://blog.csdn.net/Eastmount/>)