实验三　数字图像的形态学处理及边缘检测

一、实验目的

基于 opencv-python 实现基于形态学运算和微分算子的数字图像处理，强化对课堂讲授内容如膨胀、腐蚀、开、闭、梯度运算等知识及微分算子应用的深入理解与灵活应用。

二、实验内容

（1）用 OpenCV 实现数字图像的形态学运算，提取图像分量信息最本质的形状特征，不同运算操作的能实现的目的；

（2）用 OpenCV 编写一个程序实现基于微分算子的边缘检测，并比较各微分检测算子的优劣；

（3）要求给出膨胀、腐蚀、开、闭、梯度等运算操作的效果图。

三、实验过程

形态学操作主要包括：腐蚀、膨胀、开运算、闭运算、形态学梯度运算、顶帽运算（礼帽运算）、黑帽运算等操作。腐蚀操作和膨胀操作是形态学的基础，将腐蚀和膨胀操作进行结合，就可以实现开运算、闭运算、形态学梯度运算、顶帽运算、黑帽运算等不同形式的运算。

1. 腐蚀和膨胀运算

① 腐蚀是最基本的形态学操作之一，它能够将图像的边界点消除，使图像沿着边界向内收缩，也可以将小于指定结构体元素的部分去除。

腐蚀用来“收缩”或者“细化”二值图像中的前景，借此实现去除噪声、元素分割等功能。

语法结构：

```
dst = cv2.erode( src, kernel [, anchor[, iterations [, borderType [, borderValue ] ] ] ] )
```

参数解释：

dst： 腐蚀后输出的目标图像；src： 是要进行腐蚀的原始图像。

kernel： 腐蚀操作时所采用的结构类型，可以自定义，也可以通过函数 cv2.getStructringElement()生成；anchor： 是 element 结构中锚点位置默认

为(-1，-1)，在核的中心位置；iterations： 腐蚀操作迭代的次数，默认值为1，即只进行一次腐蚀操作

② 膨胀和腐蚀操作的作用是相反的，能对图像的边界进行扩张，将与当前对象（前景）接触到的背景点合并到当前对象内，从而实现将图像的边界点向外扩张。

语法结构：

```
dst=cv2.dilate(src,kernel[,anchor[,iterations[,borderType[,borderValue]]]])
```

代码演示：

```python
#encoding:utf-8
import cv2
import numpy as np

#读取图片
src1 = cv2.imread('D:/Pic/test01.jpg', -1)
src2 = cv2.imread('D:/Pic/test02.png', -1)

#设置卷积核
kernel = np.ones((5,5), np.uint8)

#图像腐蚀处理
erosion1 = cv2.erode(src1, kernel)
#图像膨胀处理
dilatation = cv2.dilate(src2, kernel)


#显示图像
cv2.imshow("e-src", src1)
cv2.imshow("erosion", erosion1)

cv2.imshow("d-src", src2)
cv2.imshow("dilatation", dilatation)

#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. 通用形态学函数

通用形态学： 将腐蚀和膨胀操作进行组合，就可以实现开运算、闭运算（关运算）、形态学梯度（Morphological Gradient）运算、礼帽运算（顶帽运算）、黑帽运算、击中击不中等多种不同形式的运算。

OpenCv 提供了 cv2.morphologyEx

```
dst=cv2.morphologyEx
（src,op,kernel[,anchor[,iterations[,borderType[,borderValue]]]]）
```

通过 cv2.morphologyEx，可实现下列运算操作：

- ✓ 开运算操作：先将图像腐蚀，在对腐蚀的结果进行膨胀，可达到去噪、计数的目的；
- ✓ 闭运算操作：先将图像膨胀，再对膨胀的结果进行腐蚀，可达到关闭前景物体内部的小孔；去除物体上的小黑点；将不同的前景图像进行连接等目的；
- ✓ 梯度运算操作：用图像的膨胀图像减腐蚀图像的操作，可获取原始图像中前景图像的边缘；
- ✓ 顶帽运算操作：用原始图像减去其开运算图像的操作，能够获取图像的噪声信息；得到比原始图像的边缘更亮的边缘信息；
- ✓ 黑帽操作运算：用闭运算图像减去原始图像的操作，可获取图像内部的小孔；前景色中的小黑点；比原始图像的边缘更暗的边缘部分。

代码演示：

```python
#encoding:utf-8
import cv2
import numpy as np

#读取图片
src = cv2.imread('D:/Pic/aaa.png', cv2.IMREAD_UNCHANGED)

#设置卷积核
kernel = np.ones((5,5), np.uint8)

#图像开运算
result_open = cv2.morphologyEx(src, cv2.MORPH_OPEN, kernel)

#图像闭运算
result_close = cv2.morphologyEx(src, cv2.MORPH_CLOSE, kernel)

#图像梯度运算
result_grid = cv2.morphologyEx(src, cv2.MORPH_GRADIENT, kernel)

#图像顶帽运算
result_tophat = cv2.morphologyEx(src, cv2.MORPH_TOPHAT, kernel)

#图像黑帽运算
result_blackhat = cv2.morphologyEx(src, cv2.MORPH_BLACKHAT, kernel)

#显示图像
cv2.imshow("src", src)
cv2.imshow("result_open", result_open)
cv2.imshow("result_close", result_close)
cv2.imshow("result_grid", result_grid)
cv2.imshow("result_tophat", result_tophat)
cv2.imshow("result_blackhat", result_blackhat)
#等待显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. 基于微分算子的图像边缘检测

图像锐化和边缘提取技术可以消除图像中的噪声，提取图像信息中心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。本次实验要求使用 Laplacian 算子、Robert 算子、Prewitt 算子和 Sobel 算子进行图像退化边缘处理。

- Robert 算子基于交叉查分的梯度算法，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。其缺点是对边缘定位不太准确，提取的边缘线条较粗；

Robert 算子主要通过 Numpy 定义模板，再调用 OpenCV 的 filter2D（）函数实现边缘提取，该函数主要利用内核实现对图像的卷积运算。

dst = filter2D(src, ddepth, kernel[, dst[, anchor[, 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
- kernel表示卷积核，一个单通道浮点型矩阵
- anchor表示内核的基准点，其默认值为（-1，-1），位于中心位置
- delta表示在储存目标图像前可选的添加到像素的值，默认值为0
- borderType表示边框模式

- Prewitt 算子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 算子的模板是 2*2，故 Prewitt 算子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；

Prewitt 算子实现过程类似于 Robert 算子。通过 Numpy 定义模板，再调用 OpenCV 的 filter2D()函数实现对图像的卷积运算，最终通过 convertScaleAbs()和 addWeighted()函数实现边缘提取。

- Sobel 算子是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导，其在 Prewitt 算子的基础上增加了权重的概念，认为相邻点的距离远近对当前像素点的影响是不同的，距离越近的像素点对应当前像素的影响越大，从而实现图像锐化并突出边缘轮廓；

Sobel 算子通过如下函数实现：

dst = Sobel(src, ddepth, dx, dy[, dst[, ksize[, 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
- ksize表示Sobel算子的大小，其值必须是正数和奇数
- 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

注意，在进行Sobel算子处理之后，还需要调用convertScaleAbs()函数计算绝对值，并将图像转换为8位图进行显示。其算法原型如下：

dst = convertScaleAbs(src[, dst[, alpha[, beta]]])

- src表示原数组
- dst表示输出数组，深度为8位
- alpha表示比例因子
- beta表示原数组元素按比例缩放后添加的值

- Laplacian 算子是二阶微分算子，常用于图像增强领域和边缘提取。它通过灰度差分计算邻域内的像素，基本流程是：判断图像中心像素灰度值和它周围其他像素的灰度值，如果中心像素的灰度值更高，则提升中心像素的灰度；反之降低中心像素的灰度，从而实现图像锐化操作。

Laplacian 算子通过如下函数实现：

dst = Laplacian(src, ddepth[, dst[, ksize[, 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
- ksize表示用于计算二阶导数的滤波器的孔径大小，其值必须是正数和奇数，且默认值为1，更多详细信息查阅getDerivKernels
- scale表示计算拉普拉斯算子值的可选比例因子。默认值为1，更多详细信息查阅getDerivKernels
- 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，默认值为0
- borderType表示边框模式，更多详细信息查阅BorderTypes

代码实现：

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

#读取图像
img = cv2.imread('D:/Pic/lena.bmp')
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)

#Roberts算子
kernelx = np.array([[-1,0],[0,1]], dtype=int)
kernely = np.array([[0,-1],[1,0]], dtype=int)
x = cv2.filter2D(binary, cv2.CV_16S, kernelx)
y = cv2.filter2D(binary, cv2.CV_16S, kernely)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#Prewitt算子
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]], dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]], dtype=int)
x = cv2.filter2D(binary, cv2.CV_16S, kernelx)
y = cv2.filter2D(binary, cv2.CV_16S, kernely)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Prewitt = cv2.addWeighted(absX,0.5,absY,0.5,0)

#Sobel算子
x = cv2.Sobel(binary, cv2.CV_16S, 1, 0)
y = cv2.Sobel(binary, cv2.CV_16S, 0, 1)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#拉普拉斯算法
dst = cv2.Laplacian(binary, cv2.CV_16S, ksize = 3)
Laplacian = cv2.convertScaleAbs(dst)
```

```
#效果图
titles = ['Source Image', 'Binary Image', 'Roberts Image',
          'Prewitt Image','Sobel Image', 'Laplacian Image']
images = [lenna_img, binary, Roberts, Prewitt, Sobel, Laplacian]
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()
```