**cv2.threshold()、OTSU和cv2.adaptiveThreshold()简要介绍**

**定义**

* cv2.threshold(src, thresh, maxval, type)
* cv2.adaptiveThreshold(src,maxvalue,method,type1,size,c)

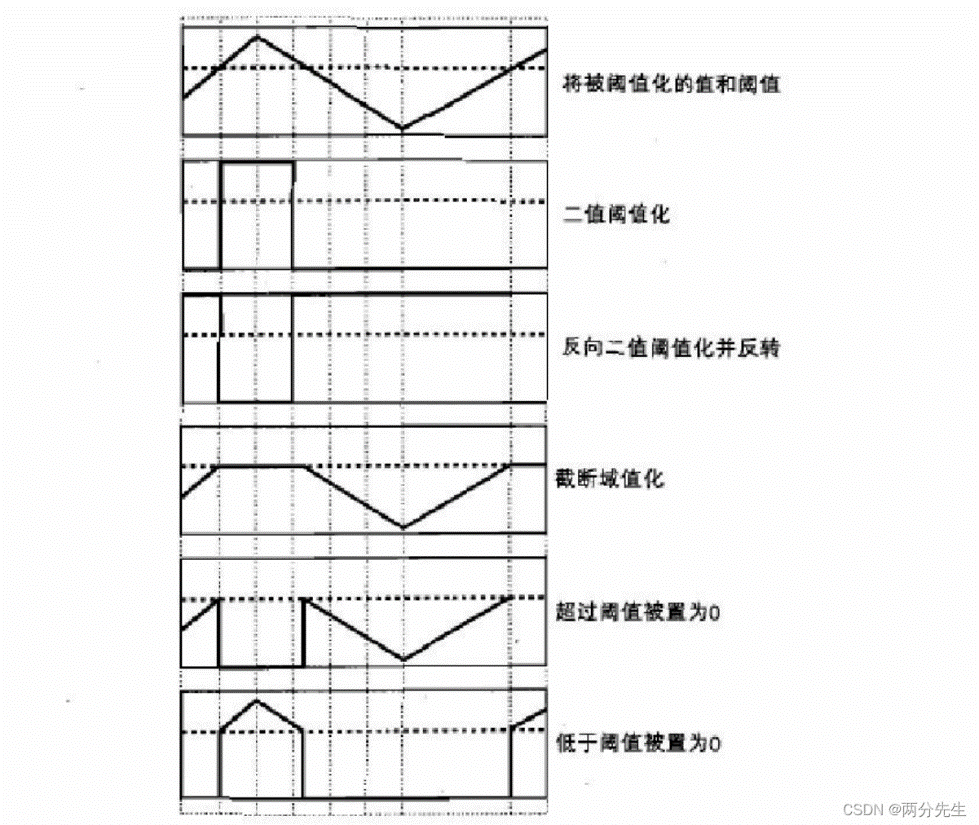
**用法**

|  |  |  |  |
| --- | --- | --- | --- |
| cv2.threshold(src, thresh, maxval, type) | | | |
| 函数作用 | | 1.若为普通[阈值](https://so.csdn.net/so/search?q=%E9%98%88%E5%80%BC&spm=1001.2101.3001.7020)分割，则设定一个阈值，当像素值高于阈值时，我们给这个像素赋予一个新的值（可能是白色），否则我们给它赋予另外一个新值（也许是黑色）。有两个返回值，第一个是得到的阈值值，第二个就是阈值化后的图像  2.若为OTSU阈值分割，要把阈值设为 0。 此时的函数 cv2.threshold()会自动寻找最优阈值，并将返回该阈值。 | |
| src | | 输入图像（只能输入灰度图） | |
| thresh | | 设定的阈值，若使用的OTSU大津阈值,则该值为0 | |
| maxval | | 当像素值高于（或者小于）阀值时，应该被赋予新的像素值 | |
| type | | [二值化](https://so.csdn.net/so/search?q=%E4%BA%8C%E5%80%BC%E5%8C%96&spm=1001.2101.3001.7020)操作的类型，共有五种类型（见下述）  若使用OTSU阈值处理，则在其后+cv2.THRESH\_OTSU  （例如：cv2.THRESH\_BINARY+cv2.THRESH\_OTSU) | |
| cv2.adaptiveThreshold(src,maxvalue,method,type1,size,c) | | |
| 函数作用 | 以每个像素点作为中心取一定的区域，计算这个区域的阈值，决定这个像素点变0还是变255 | |
| src | 输入图像（只能输入灰度图） | |
| maxvalue | 当像素值高于（或者小于）阀值时，应该被赋予新的像素值 | |
| method | 自适应阈值算法，ADAPTIVE\_THRESH\_MEAN\_C（阈值取自相邻区域的平均值）或ADAPTIVE\_THRESH\_GAUSSIAN\_C（阈值取相邻区域的加权和，权重为一个高斯窗口） | |
| type | 只能选择THRESH\_BINARY或者THRESH\_BINARY\_INV | |
| size | 领域大小，高宽一般取奇数 | |
| c | 每个邻域计算出的值需要再减去c得到阈值 | |

其中，type包含以下5种类型：

* cv2.THRESH\_BINARY
* cv2.THRESH\_BINARY\_INV
* cv2.THRESH\_TRUNC
* cv2.THRESH\_TOZERO
* cv2.THRESH\_TOZERO\_INV

5种类型图解如下：



**示例**

**cv2.threshold()**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**img = cv2.imread('D:/4.png',0)**

**ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)**

**ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)**

**ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)**

**ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)**

**ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)**

**titles = ['Original Image','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']**

**images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]**

**for i in range(6):**

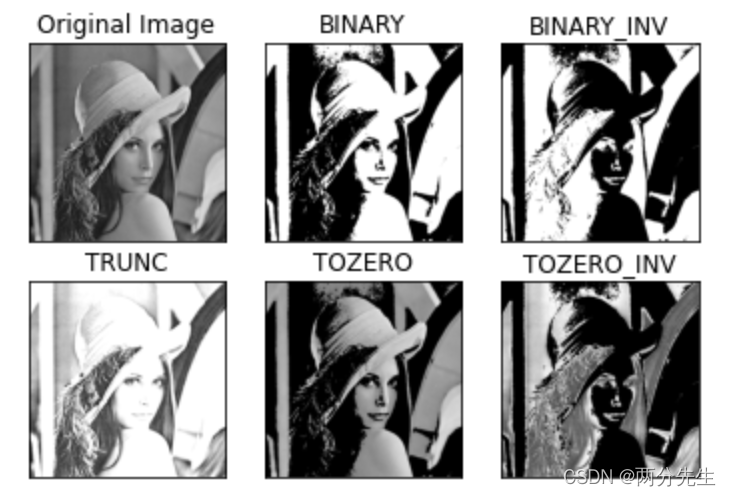
**plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')**

**plt.title(titles[i])**

**plt.xticks([]),plt.yticks([])**

**plt.show()**

效果图



**cv2.adaptiveThreshold()**

**import numpy as np**

**from matplotlib import pyplot as plt**

**import cv2**

**img = cv2.imread('D:/Images/Lena.jpg',0)**

**img = cv2.medianBlur(img,5)**

**ret , th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)**

**th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C , cv2.THRESH\_BINARY,11,2 )**

**th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C , cv2.THRESH\_BINARY,11,2)**

**titles = ['original image' , 'global thresholding (v=127)','Adaptive mean thres holding', 'adaptive gaussian thresholding']**

**images = [img,th1,th2,th3]**

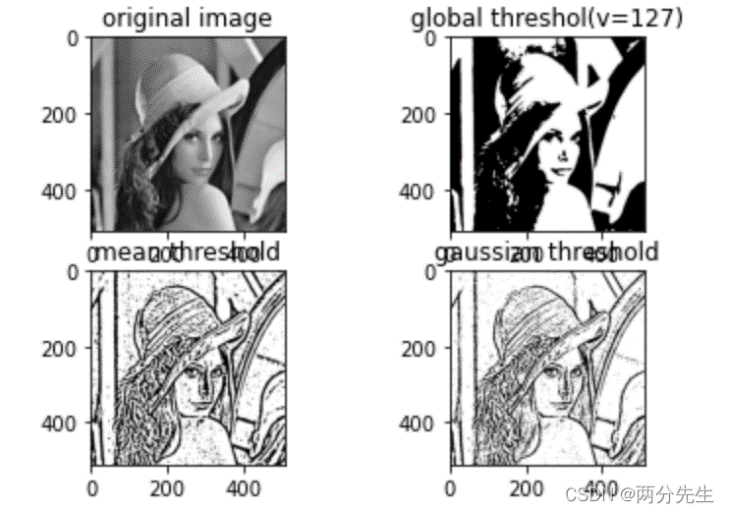
**for i in range(4):**

**plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')**

**plt.title(titles[i])**

**plt.show()**

效果图



Opencv-Python | 图像阈值处理及代码实现

        图像的二值化或阈值化 (Binarization) 旨在提取图像中的目标物体，将背景以及噪声区分开来。通常会设定一个阈值T，通过阈值将图像的像素划分为两类：大于阈值的像素群和小于阈值的像素群。 灰度转换处理后的图像中，每个像素都只有一个灰度值，其大小表示明暗程度。二值化处理可以将图像中的像素划分为两类颜色

1、简单阈值处理

**（1）函数介绍**

**ret, dst = cv2.threshold(src, thresh, maxval, type)**  
        **src**： 输入图，只能输入单通道图像，通常来说为灰度图  
        **dst**： 输出图  
        **thresh**： 阈值  
        **maxval**： 当像素值超过了阈值（或者小于阈值，根据type来决定），所赋予的值  
        **type**：二值化操作的类型，包含以下5种类型： cv2.THRESH\_BINARY；         cv2.THRESH\_BINARY\_INV； cv2.THRESH\_TRUNC；         cv2.THRESH\_TOZERO； cv2.THRESH\_TOZERO\_INV  
       **cv2.THRESH\_BINARY** 超过阈值部分取maxval（最大值），否则取0  
        **cv2.THRESH\_BINARY\_INV** THRESH\_BINARY的反转  
        **cv2.THRESH\_TRUNC** 大于阈值部分设为阈值，否则不变  
        **cv2.THRESH\_TOZERO** 大于阈值部分不改变，否则设为0  
        **cv2.THRESH\_TOZERO\_INV**THRESH\_TOZERO的反转

（2）代码实现

import cv2 #opencv读取的格式是BGR

import numpy as np

import matplotlib.pyplot as plt#Matplotlib是RGB

def img\_show(name, img): # 展示图像函数

cv2.imshow(name, img)

cv2.waitKey(0)

cv2.destroyAllWindows()

x = np.zeros([255,255])

for i in np.arange(0,255):

x[:,i] = i

img\_x = np.array(x,dtype='uint8')

img\_show('img\_x',img\_x)

ret, thresh1 = cv2.threshold(img\_x, 127, 255, cv2.THRESH\_BINARY)

ret, thresh2 = cv2.threshold(img\_x, 127, 255, cv2.THRESH\_BINARY\_INV)

ret, thresh3 = cv2.threshold(img\_x, 127, 255, cv2.THRESH\_TRUNC)

ret, thresh4 = cv2.threshold(img\_x, 127, 255, cv2.THRESH\_TOZERO)

ret, thresh5 = cv2.threshold(img\_x, 127, 255, cv2.THRESH\_TOZERO\_INV)

titles = ['Original Image', 'BINARY', 'BINARY\_INV', 'TRUNC', 'TOZERO', 'TOZERO\_INV']

images = [img\_x, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):

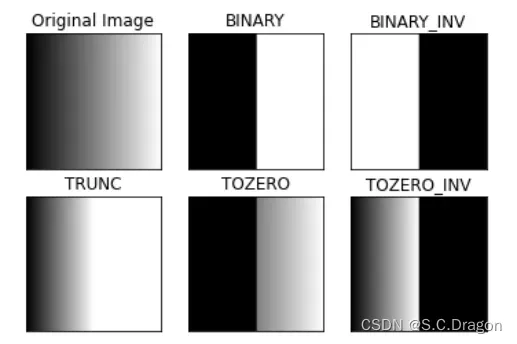
plt.subplot(2, 3, i + 1), plt.imshow(images[i], 'gray')

plt.title(titles[i])

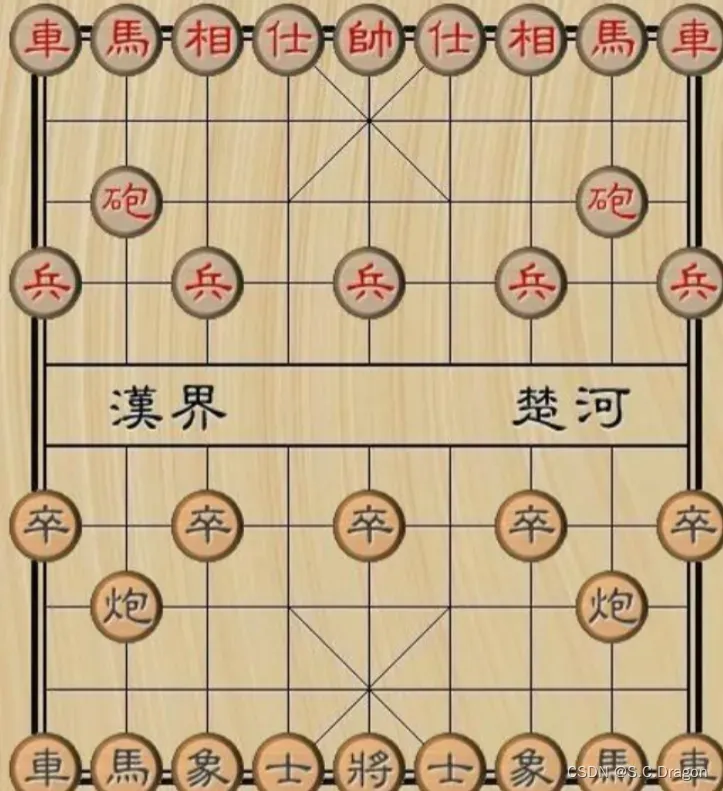
plt.xticks([]), plt.yticks([])

plt.show()

Python



 下面我们用彩色RGB图像进行演示：  
  其中原始RGB三通道图如下图所示：



import cv2 #opencv读取的格式是BGR

import numpy as np

import matplotlib.pyplot as plt#Matplotlib是RGB

def img\_show(name, img):

cv2.imshow(name, img)

cv2.waitKey(0)

cv2.destroyAllWindows()

img=cv2.imread('chess.jpg')

img\_gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

img\_gray

img\_show('chess',img\_gray)

ret, thresh1 = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

ret, thresh2 = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY\_INV)

ret, thresh3 = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_TRUNC)

ret, thresh4 = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_TOZERO)

ret, thresh5 = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_TOZERO\_INV)

titles = ['Original Image', 'BINARY', 'BINARY\_INV', 'TRUNC', 'TOZERO', 'TOZERO\_INV']

images = [img\_gray, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):

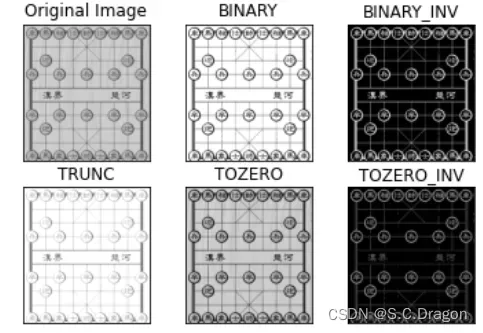
plt.subplot(2, 3, i + 1), plt.imshow(images[i], 'gray')

plt.title(titles[i])

plt.xticks([]), plt.yticks([])

plt.show()

Python



2、 自适应阈值

（1）函数介绍

th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE\_THRESH\_MEAN\_C, cv.THRESH\_BINARY,11,2)

* 第一个参数（img）：源图像
* th2：输出图像，与源图像大小一致
* 第三个参数（255）：超过阈值的部分取值是多少（对于cv.THRESH\_BINARY而言）
* 第四个参数（cv.ADAPTIVE\_THRESH\_MEAN\_C）：
* （1）在一个邻域内计算阈值所采用的算法，有两个取值，分别为 ADAPTIVE\_THRESH\_MEAN\_C 和 ADAPTIVE\_THRESH\_GAUSSIAN\_C
* （2）ADAPTIVE\_THRESH\_MEAN\_C的计算方法是计算出领域的平均值再减去第七个参数2的值。
* （3）ADAPTIVE\_THRESH\_GAUSSIAN\_C的计算方法是计算出领域的高斯均值再减去第七个参数2的值
* 第五个参数（cv.THRESH\_BINARY）：这是阈值类型，只有两个取值，分别为 THRESH\_BINARY 和THRESH\_BINARY\_INV
* 第六个参数（11）：adaptiveThreshold的计算单位是像素的邻域块大小选择，这是局部邻域大小，3、5、7等
* 第七个参数（2）：这个参数实际上是一个偏移值调整量，用均值和高斯计算阈值后，再减或加这个值就是最终阈值。

（2）代码实现

img = cv2.imread('chess.jpg',0)

img = cv2.medianBlur(img,5) # 中值滤波

img\_show('1',img)

ret,th1 = cv2.threshold(img,127,255,cv.THRESH\_BINARY)

th2 = cv2.adaptiveThreshold(img,255,cv.ADAPTIVE\_THRESH\_MEAN\_C,\

cv2.THRESH\_BINARY,11,2)

th3 = cv2.adaptiveThreshold(img,255,cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\

cv2.THRESH\_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',

'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']

images = [img, th1, th2, th3]

for i in np.arange(4):

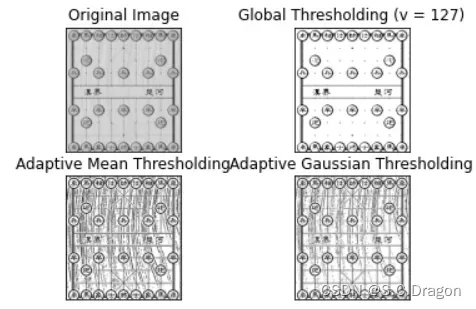
plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

Python



3、Outs二值化处理

（1）基础介绍

在第一部分中，第二个参数retVal（ret）。当我们进行Otsu的二值化时，就会用到它。那是什么？

在全局阈值处理中，我们使用任意值作为阈值，那么，我们如何知道我们选择的值是好还是坏？答案是，试错法。但是考虑一个双峰图像（简单来说，双峰图像是直方图有两个峰值的图像）。对于该图像，我们可以近似地将这些峰值中间的一个值作为阈值，这就是Otus二值化所做的。所以简单来说，它会自动从图像直方图中计算出双峰图像的阈值。（对于非双峰图像，二值化将不准确）

为此，使用了我们的 cv2.threshold() 函数，但传递了一个额外的标志cv2.THRESH\_ OTSU。对于阈值，只需传递 zero。然后算法找到最佳阈值并将您作为第二个输出retVal返回。如果不使用Otsu阈值，则 retVal 与您使用的阈值相同。

看看下面的例子。输入图像是噪声图像。在第一种情况下，我为 127 应用了全局阈值。在第二种情况下，我直接应用了Otsu的阈值。在第三种情况下，我使用 5×5 高斯核过滤图像以去除噪声，然后应用 Otsu 阈值。了解噪声过滤如何改善结果。

（2）代码实现

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('noisy.jpg',0)

# global thresholding

ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

# Otsu's thresholding

ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# Otsu's thresholding after Gaussian filtering

blur = cv2.GaussianBlur(img,(5,5),0)

ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# plot all the images and their histograms

images = [img, 0, th1,

img, 0, th2,

blur, 0, th3]

titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',

'Original Noisy Image','Histogram',"Otsu's Thresholding",

'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in np.arange(3):

plt.subplot(3,3,i\*3+1),plt.imshow(images[i\*3],'gray')

plt.title(titles[i\*3]), plt.xticks([]), plt.yticks([])

plt.subplot(3,3,i\*3+2),plt.hist(images[i\*3].ravel(),256)

plt.title(titles[i\*3+1]), plt.xticks([]), plt.yticks([])

plt.subplot(3,3,i\*3+3),plt.imshow(images[i\*3+2],'gray')

plt.title(titles[i\*3+2]), plt.xticks([]), plt.yticks([])

plt.show()

Python

