**OpenCV学习笔记：图像二值化**

2022年06月02日 21:44 ·  阅读 493

持续创作，加速成长！这是我参与「掘金日新计划 · 6 月更文挑战」的第6天，[点击查看活动详情](https://juejin.cn/post/7099702781094674468)

本文是我之前在微信公众号上的一篇文章记录。原链接为：[# OpenCV学习笔记：图像二值化](https://link.juejin.cn/?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzA5NTM3MDI0OA%3D%3D%26mid%3D2247484076%26idx%3D1%26sn%3D4b3595dc236c09e2c9aa6fd4dacb9d7d%26chksm%3D90412fcba736a6dda4013a545b81214df6606f0c54c0b126033622abe8b125e06d04444ee4c6%23rd)

图像二值化（ Image Binarization）就是将图像上的像素点的灰度值设置为0或255，也就是将整个图像呈现出明显的黑白效果的过程。即将256个亮度等级的灰度图像通过适当的阀值选取而获得仍然可以反映图像整体和局部特征的二值化图像。

由于二值图像数据足够简单，许多视觉算法都依赖二值图像。通过二值图像，能更好地分析物体的形状和轮廓。二值图像也常常用作原始图像的**掩模（又称遮罩、蒙版，Mask）**：它就像一张部分镂空的纸，把我们不感兴趣的区域遮掉。进行二值化有多种方式，其中最常用的就是采用**阈值法（Thresholding）** 进行二值化。

Opencv中提供了 threshold()和adaptiveThreshold()两个函数用于实现图像的二值化。

**直接阈值化 - threshold()**

下面我们先介绍threshold，它使用全局阈值，即对整个图像中的每一个像素都选用相同的阈值。该函数原型如下：

**double** **threshold**(InputArray src,

OutputArray dst,

**double** thresh,

**double** maxval,

**int** type);

复制代码

* src：源图像，只能是CV\_8U或者CV\_32F位。
* dst：输出图像 ，与输入图像有相同的大小，数据类型和通道数。
* thresh：二值化的阈值
* maxval：dst图像中最大值
* type：二值化方法类型，如下：

**enum** **ThresholdTypes** {

THRESH\_BINARY = 0,*//灰度值大于阀值的为最大值，其他的为0*

THRESH\_BINARY\_INV = 1,*//灰度值大于阀值的为0，其他的为最大值*

THRESH\_TRUNC = 2,*//灰度值大于阀值的为阀值，其他的不变*

THRESH\_TOZERO = 3,*//灰度值大于阀值的不变，其他的值为0*

THRESH\_TOZERO\_INV = 4,*//灰度值大于阀值的为0，其他的值不变*

THRESH\_MASK = 7,

THRESH\_OTSU = 8,*//大律法自动寻求全局阀值*

THRESH\_TRIANGLE = 16*//三角形法自动寻求全局阀值*

};

复制代码

看下如下程序：

**#include <iostream>**

**#include <string>**

**#include <cmath>**

**#include "opencv2/opencv.hpp"**

*//使用CommandLineParser对输入的参数进行分析，获取输入的图片路径*

std::string **GetFileName**(**int** argc,**char**\* argv[])

{

*/\**

*argc : the size of argv[]*

*argv : the parameters of comdline*

*\*/*

**const** **char**\* key\_map = {

"{help h usage? || usage information}"

"{@picture || input picture}"

};

*/\**

*\* 函数功能：构造函数*

*\* 参数：*

*\* [in] argc main函数中的第一参数，即运行程序中获得指令的个数*

*\* [in] argv main函数中的第二个参数，即运行程序中指令的内容*

*\* [in] key\_map 当启动程序是没有输入任何指令，则使用key\_map中默认的指指令。*

*\* 备注：*

*\* key\_map中的格式：*

*\* "{ s| 123asd |string parameter}*

*\* { d| 100 |digit parameter }*

*\* { @c| false |without camera }*

*\* { 1| some text |help }*

*\* { 2| 333 |another help }"*

*\*参数or指令名称|指令or参数内容 |指令说明*

*\**

*\**

*\* 运行程序输入指令的方式如下：（例如程序名称为extest.exe）*

*\* extest.exe -s=123asdd -d=1000 -@c=10*

*\* 注意：指令名称前面需要加一个“-”，或“--”。当输入指令后面没有参数，默认为true*

*\* 另外，前面加@的指令可以不输入指令名称，直接设置指令内容即可。*

*\* 没有输入的指令，则使用key\_map中的默认值*

*\*/*

cv::CommandLineParser **parser**(argc, argv, key\_map);

*//如果没有输入指令，则会使用上面设置的默认指令，则存在‘help’指令*

**if** (parser.has("help"))

{

parser.printMessage();

exit(0);

}

*//检查输入的指令是否有有错误，即无法解析*

**if** (!parser.check())

{

parser.printErrors();

exit(-1);

}

std::string fileName = parser.get<std::string>(0);

**return** fileName;

}

**void** **onChangeTrackBar**(**int** pos, **void** \*userdata)

{

cv::Mat dstImage;

cv::Mat grayImage = \*(cv::Mat\*)userdata;

cv::threshold(grayImage, dstImage, 130, 255, pos);

cv::imshow("grayImage", dstImage);

}

**int** **main**(**int** argc,**char**\* argv[])

{

std::string fileName = GetFileName(argc, argv);

cv::Mat srcImage = cv::imread(fileName);

**if** (srcImage.empty())

{

std::cerr << "failed to read image" << std::endl;

exit(-1);

}

cv::Mat grayImage;

*//将图像转成灰色图*

cv::cvtColor(srcImage, grayImage, cv::COLOR\_BGR2GRAY);

*//新建窗口*

cv::namedWindow("grayImage");

*//在窗口上显示图像*

cv::imshow("grayImage", grayImage);

*//在窗口上创建一个trackbar*

cv::createTrackbar("threshold", "grayImage", 0, 4, onChangeTrackBar, (**void**\*)&grayImage);

cv::imshow("srcImage", srcImage);

cv::waitKey(0);

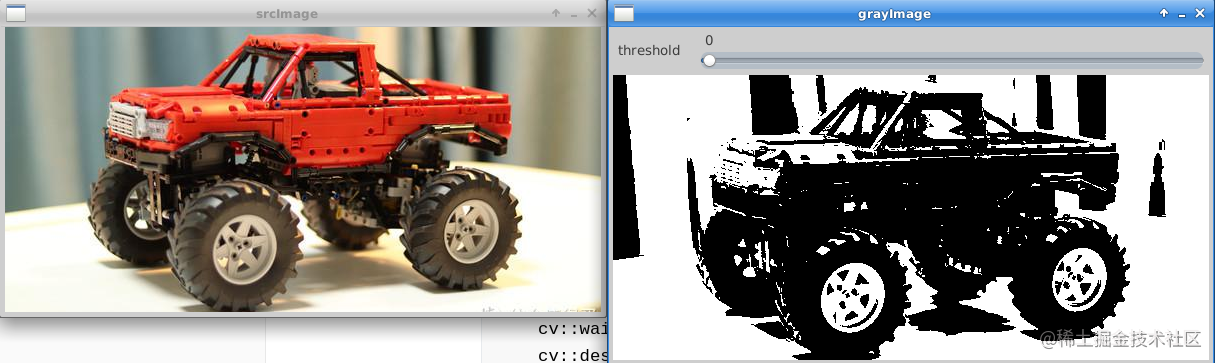
cv::destroyAllWindows();

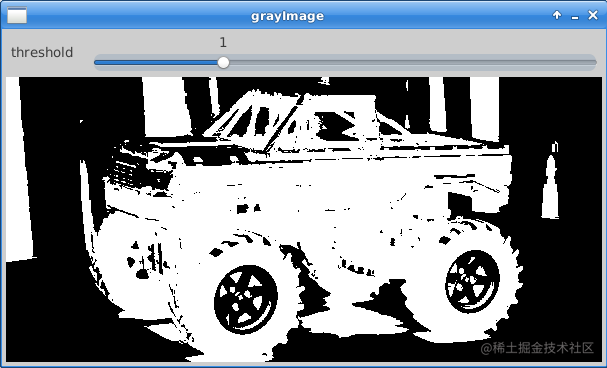
**return** 0;

}

复制代码

这个程序可以一直滑动滚动条的滑块改变不同的值，而且图像会出现相应的变化，如下图滑块到不同位置的效果：





这里其实是靠createTrackbar事件处理得到的，学过一点计算机的人就可以理解为该函数其实一直运行在后台检测是否有滑块移动这种中断产生，然后当你移动滑块时就会触发中断，这时回调函数就相当于中断服务函数来处理中断（不完全准确但是可以这么理解）。

这个程序就是通过滑动不同的二值化类型来进行图像二值化处理。

**自适应阈值化 - adaptiveThreshold()**

接下来我们介绍一下另外一个二值化函数adaptiveThreshold()自适应阀值的二值化方法。

**void** **cv::adaptiveThreshold**(

cv::InputArray src, *// 输入图像*

cv::OutputArray dst, *// 输出图像*

**double** maxValue, *// 二值化最大值*

**int** adaptiveMethod, *// 自适应方法，平均或高斯*

**int** thresholdType *// 阈值化类型*

**int** blockSize, *// 块大小*

**double** C *// 常量*

);

复制代码

* src：输入图像，必须是8位单通道(CV\_8UC1)图像；
* dst：需和原图像尺寸类型一致；
* maxValue：二值化最大值；
* adaptiveMethod：用于指定自适应阈值的算法，ADAPTIVE\_THRESH\_MEAN\_C ，ADAPTIVE\_THRESH\_GAUSSIAN\_C
* thresholdType： 二值化方法类型：必须是THRESH\_BINARY或者THRESH\_BINARY\_INV
* blockSize：用来计算阈值的象素邻域大小:，一般是3, 5, 7, 的奇数
* C：与方法有关的参数。对方法 ADAPTIVE\_THRESH\_MEAN\_C 和 ADAPTIVE\_THRESH\_GAUSSIAN\_C， 它是一个从均值或加权均值提取的常数, 有时也可以是小数或负数。

adaptiveThreshold()支持两种自适应方法，即ADAPTIVE\_THRESH\_MEAN\_C（平均）和ADAPTIVE\_THRESH\_GAUSSIAN\_C（高斯）。在两种情况下，自适应阈值T(x, y)。通过计算每个像素周围blockSize \* blockSize大小像素块的加权均值并减去常量C得到。其中，blockSize的大小必须为奇数；如果使用平均的方法，则所有像素周围的权值相同；如果使用高斯的方法，则（x,y）周围的像素的权值则根据其到中心点的距离通过高斯方程得到。

看下如下程序进行自适应阀值化效果：

**void** **test\_adaptive\_threshold**()

{

*//must be CV\_8UC1*

cv::Mat src = cv::imread("car.png", cv::IMREAD\_GRAYSCALE);

cv::Mat dst\_mean, dst\_gauss;

**int** maxVal = 255;

**int** blockSize = 55;

**double** C = 0;

cv::adaptiveThreshold(src, dst\_mean, maxVal, cv::ADAPTIVE\_THRESH\_MEAN\_C, cv::THRESH\_BINARY, blockSize, C);

cv::adaptiveThreshold(src, dst\_gauss, maxVal, cv::ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv::THRESH\_BINARY, blockSize, C);

cv::imshow("dst\_mean", dst\_mean);

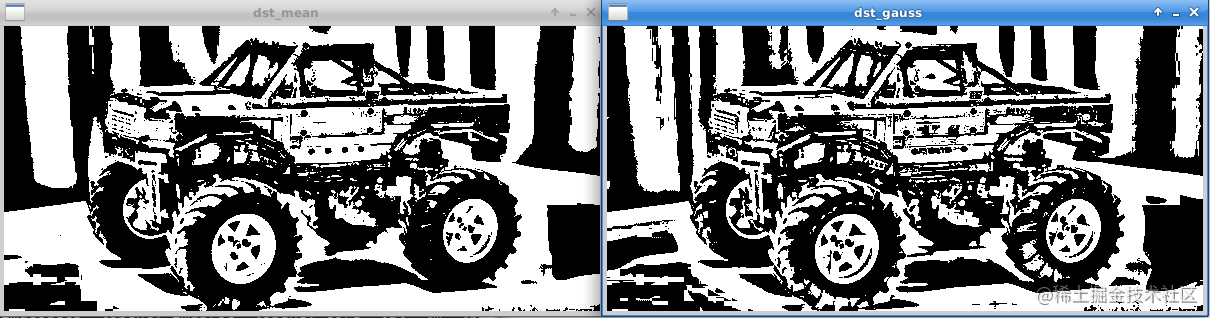
cv::imshow("dst\_gauss", dst\_gauss);

cv::waitKey(0);

**return**;

}

程序执行效果如下：



**OpenCV学习笔记：阈值化与自适应阈值化**

  在对一幅图像进行处理时，很多时候我们都会用到[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，如最常用的边缘检测算法：Canny边缘检测算法就要求执行前必须先转换为灰度图才行，但是在有的工程中遇到的问题，需要将灰度图转换为二值化图后再进行Canny才会有更好的效果。二值化操作也有注意点，如何最大可能的保存原图像的数据完整性，就需要严格设置阈值点了。OpenCV中提供两种阈值化方法，普通阈值化和自适应阈值化。

**普通阈值化**

[OpenCV](https://so.csdn.net/so/search?q=OpenCV&spm=1001.2101.3001.7020)中提供的常规阈值化的函数为：  
CVAPI(double) cvThreshold( const CvArr\* src, CvArr\* dst, double threshold, double max\_value, int threshold\_type );  
其中：src为输入的原图像；dst为输出的目标图像（此函数的输入与输出图像可以为同一个图像，src与dst最好为单通道的灰度图才能有好的效果）；threshold为要设定的阈值；max\_value为最大取值；threshold\_type为阈值化操作的可选择类型，具体如下：

CV\_THRESH\_BINARY =0, /\* value = value > threshold ? max\_value : 0

CV\_THRESH\_BINARY\_INV =1, /\* value = value > threshold ? 0 : max\_value \*/

CV\_THRESH\_TRUNC =2, /\* value = value > threshold ? threshold : value \*/

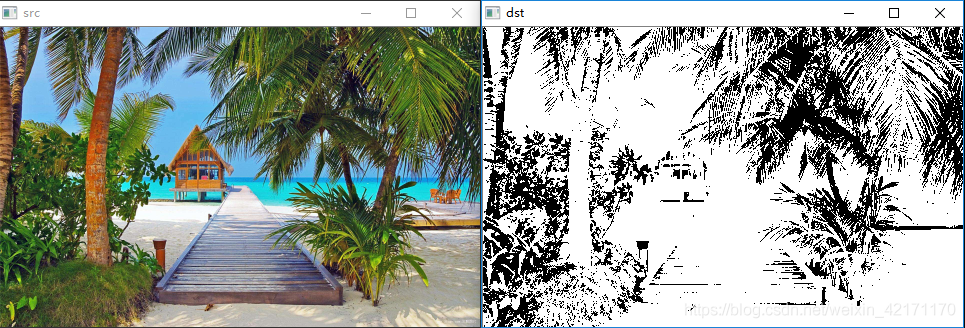
CV\_THRESH\_TOZERO =3, /\* value = value > threshold ? value : 0 \*/

CV\_THRESH\_TOZERO\_INV =4, /\* value = value > threshold ? 0 : value \*/

CV\_THRESH\_MASK =7,

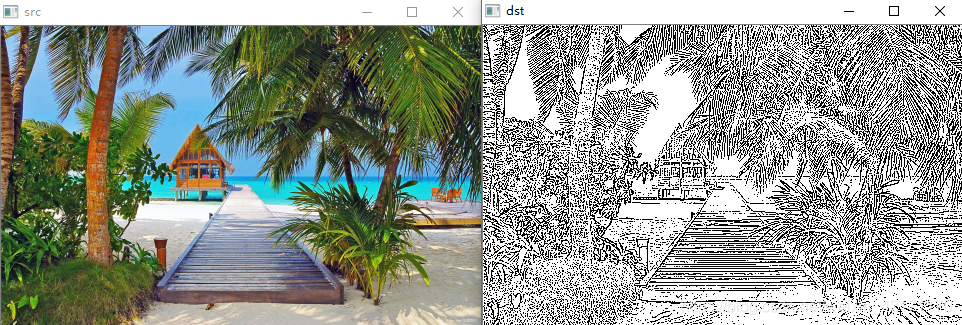
CV\_THRESH\_OTSU =8 /\* use Otsu algorithm to choose the optimal threshold value;

combine the flag with one of the above CV\_THRESH\_\* values \*/

openCV中的测试结果如下图：  
  
这里我设置的threshold为70，使用这种方法二值化后的图像显得不够好，丢失了很多图像信息。

**自适应阈值化**

上面使用普通阈值化操作得到的图像由于使用 70 为临界点进行二值化，有种“一刀切”的感觉，满足条件就为最大值 255 不满足条件就为 0，在有很强照明或者是反射梯度的图像中进行普通阈值化时就会导致有些地方的图像细节信息丢失。OpenCV中提供了一种更好的解决方法：自适应阈值化操作，函数为：  
CVAPI(void) cvAdaptiveThreshold( const CvArr\* src, CvArr\* dst, double max\_value, int adaptive\_method CV\_DEFAULT(CV\_ADAPTIVE\_THRESH\_MEAN\_C), int threshold\_type CV\_DEFAULT(CV\_THRESH\_BINARY), int block\_size CV\_DEFAULT(3), double param1 CV\_DEFAULT(5));  
其中：src与dst为输入原图像与输出目标图像（此函数的输入与输出图像不能为同一个）；  
max\_value为最大取值；  
adaptive\_method为自适应阈值取值方法，分为：  
1、CV\_ADAPTIVE\_THRESH\_MEAN\_C （平均值法）  
2、 CV\_ADAPTIVE\_THRESH\_GAUSSIAN\_C （高斯滤波取值法，即根据权重取平均值）；  
threshold\_type为取阈值类型：必须是CV\_THRESH\_BINARY或者CV\_THRESH\_BINARY\_INV；  
block\_size为用来计算阈值的象素邻域大小: 3, 5, 7, …（只能取奇数，与高斯核有关）；  
param1为与方法有关的参数。对方法 CV\_ADAPTIVE\_THRESH\_MEAN\_C 和 CV\_ADAPTIVE\_THRESH\_GAUSSIAN\_C来说，它是一个从均值或加权均值提取的常数,尽管它可以是负数。  
对方法 CV\_ADAPTIVE\_THRESH\_MEAN\_C，先求出块中的均值，再减掉param1。  
对方法 CV\_ADAPTIVE\_THRESH\_GAUSSIAN\_C ，先求出块中的加权和(gaussian)，再减掉param1。

OpenCV中的测试结果图如下：  
  
可以看出，自适应阈值化对图像中不同明亮度处的像素处理较普通阈值化来说要更有效一点；两种方法各有优劣，实际使用时还有结合工程项目情况来选取。

**OpenCV实现代码**

// Threshold\_Filter.cpp : 定义控制台应用程序的入口点。

//

#include <opencv\cv.h>

using namespace std;

using namespace cv;

#define Threshold 70

int \_tmain(int argc, \_TCHAR\* argv[])

{

IplImage \*image=NULL, \*dst=NULL, \*AdaptDst=NULL, \*ResizeImage=NULL;

image = cvLoadImage(".\\Tree.jpg",1);

ResizeImage = cvCreateImage(cvSize(image->width/4, image->height/4),image->depth,image->nChannels);

dst = cvCreateImage(cvGetSize(ResizeImage),ResizeImage->depth,1);

AdaptDst = cvCreateImage(cvGetSize(ResizeImage),ResizeImage->depth,1);

namedWindow("src",CV\_WINDOW\_AUTOSIZE);

namedWindow("dst",CV\_WINDOW\_AUTOSIZE);

cvResize(image,ResizeImage,CV\_INTER\_AREA); //原图尺寸太大,用此函数调整为原图的1/4大小

cvCvtColor(ResizeImage,dst,CV\_BGR2GRAY); //二值化前先灰度化才有好的效果

//cvThreshold(dst,dst,Threshold,255,CV\_THRESH\_BINARY); //二值化 0 ? 255

cvAdaptiveThreshold(dst,AdaptDst,255,CV\_ADAPTIVE\_THRESH\_GAUSSIAN\_C,CV\_THRESH\_BINARY,3,5);

cvShowImage("src",ResizeImage);

cvShowImage("dst",AdaptDst);

waitKey(0);

cvReleaseImage(&image);

cvReleaseImage(&ResizeImage);

cvReleaseImage(&dst);

cvReleaseImage(&AdaptDst);

destroyAllWindows();

return 0;

}

# OpenCV学习笔记–图像阈值处理、threshold函数、adaptiveThreshold函数、Otsu方法处理

（一）、threshold()函数

（二）、adaptiveThreshold()–自适应阈值处理

（三）Otsu处理

引言：阈值处理在图像中的用处可以用一句话概括：我们设定一个阈值x，当图像中的像素值大于x时，我们想要怎么处理这些像素值；当小于x时我们又想要怎么处理这些像素值。阈值处理中的threshold()函数给出了很好的解决办法。通过阈值处理我们可以得到一个二值图像。下面我们来学一下。这里我们用到的图像都是8位灰度图像。

## （一）、threshold()函数

在[opencv](https://aitechtogether.com/tag/opencv)中可以使用cv2.threshold（）函数进行阈值处理。

reval， newimg = cv2.threshold (img, x, maxval, type)

参数说明：

reval ：是一个返回值，这个值是我们在threshhold参数里设置的x的值。

newimg ：是我们经过阈值处理之后的图像。

img：是我们要进行阈值处理的图像，即我们操作的对象。  
x：是我们设置的一个阈值，也就是我们对图像img中的像素点的值进行操作时，是以这个阈值x为标准进行阈值处理的。

maxval：当我们对图像img中的像素点的值和阈值x进行比较时，如果img中的像素点的值大于或者是小于x的值，我们就把图像的像素点的值设置为maxval，maxval最大值是255。其他的设置为0。这个参数只有在type为THRESH\_BINARY或者是THRESH\_BINARY\_INV类型时起作用。

type：这个参数我们好好将分析一下，因为这个参数在阈值处理中对结果有主导作用。

type的类型主要有以下几个：

THRESH\_BINARY：也叫做二值化阈值处理，当type为该类型时，表示我们当前对图像img中的像素值与阈值x进行比较时，当图像img中的像素值大于阈值x时，将图像的当前像素点的值设置为maxval，如果不大于，则设置为0。即经过阈值处理之后的图像是一个二值图像，由最大值maxval和0组成。

THRESH\_BINARY\_INV：也叫做反二值化阈值处理，当type为该类型时，表示我们当前对图像img中的像素值与阈值x进行比较时，当图像img中的像素值大于阈值x时，将图像的当前像素点的值设置为0，如果不大于，则设置为maxval。

THRESH\_TRUNC：也叫做截断阈值处理。当type为该类型时，表示我们当前对图像img中的像素值与阈值x进行比较时，当图像img中的像素值大于阈值x时，那就把大于x的像素点的值均设定为x，不大于x的值则我们不做处理。

THRESH\_TOZERO\_INV：也叫做超阈值0处理。从字面意思我们就可以理解，当type为该类型时，表示我们当前对图像img中的像素值与阈值x进行比较时，当图像img中的像素值大于阈值x时，那就把大于x的像素点的值均设定为0，不大于x的值则我们不做处理。

THRESH\_TOZERO：也叫做阈值0处理。和上面的这个类型刚好相反。当type为该类型时，表示我们当前对图像img中的像素值与阈值x进行比较时，当图像img中的像素值不大于阈值x时，那就把大于x的像素点的值均设定为0，大于x的值则我们不做处理。

文字看着很难受，可视化能够让我们清楚的了解它们之间的区别。接下来我们通过实践来练习一下这个函数，当type取不同的类型时的区别通过程序一眼就能明白。我们通过简单的数组进行举例，然后再用图像看区别。

练习使用带有数组的函数。示例代码如下：

import numpy as np

import cv2

# 声明一个数组，用生成随机数的函数np.random.randint生成一个数组。

data = np.random.randint(0, 256, size=[4, 4], dtype=np.uint8)

print('data=\n', data)

# 使用cv2.threshold()函数

# 设定阈值x为127，这个阈值是自己定义的哈， maxval=255，type = cv2.THRESH\_BINARY

reval1, data\_THRESH\_BINARY = cv2.threshold(data, 127, 255, type=cv2.THRESH\_BINARY)

print('data\_THRESH\_BINARY=\n', data\_THRESH\_BINARY)

# 设定阈值x为120， maxval=255，type = cv2.THRESH\_BINARY\_INV

reval2, data\_THRESH\_BINARY\_INV = cv2.threshold(data, 120, 255, type=cv2.THRESH\_BINARY\_INV)

print('data\_THRESH\_BINARY\_INV=\n', data\_THRESH\_BINARY\_INV)

# 设定阈值x为200，maxval=255，type = cv2.THRESH\_TRUNC

reval3, data\_THRESH\_TRUNC = cv2.threshold(data, 200, 255, type=cv2.THRESH\_TRUNC)

print('data\_THRESH\_TRUNC=\n', data\_THRESH\_TRUNC)

# 设定阈值x为200，maxval=255，type = cv2.THRESH\_TOZERO\_INV

reval4, data\_THRESH\_TOZERO\_INV = cv2.threshold(data, 200, 255, type=cv2.THRESH\_TOZERO\_INV)

print('data\_THRESH\_TOZERO\_INV=\n', data\_THRESH\_TOZERO\_INV)

# 设定阈值x为100，maxval=255，type = cv2.THRESH\_TOZERO

reval5, data\_THRESH\_TOZERO = cv2.threshold(data, 100, 255, type=cv2.THRESH\_TOZERO)

print('data\_THRESH\_TOZERO=\n', data\_THRESH\_TOZERO)

Python

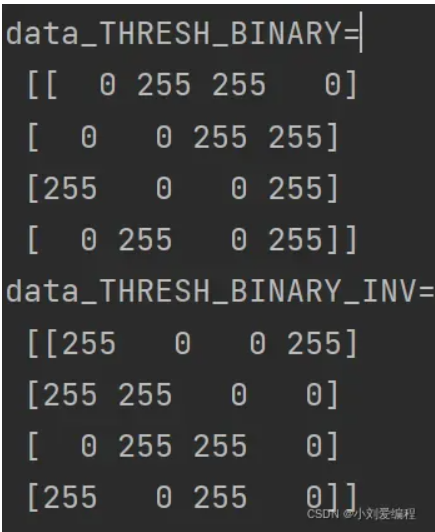
Copy

结果：

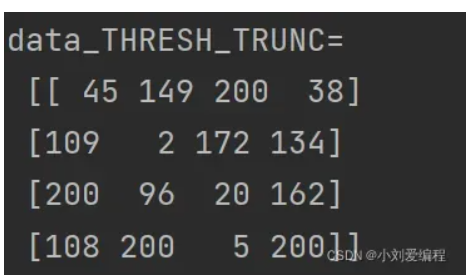
随机生成的数组：



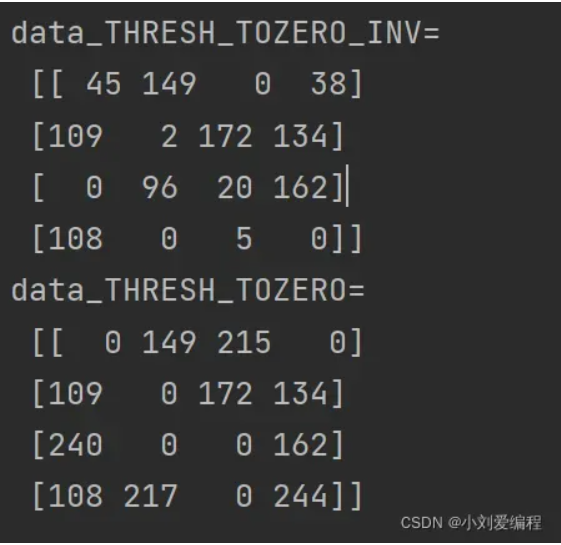
二值化阈值和去二值化阈值的结果如下：



截断阈值处理结果如下，大于200的都改为了200：



超阈值置零和低阈值置零结果如下：



可以结合我们上面解释的类型，处理方式也不同，再了解上面的结果，就一目了然了。

用一张图来说明函数的使用。这里我们对上面的代码稍作调整，将数组替换为图像数组，然后进行阈值处理，直接观察图像就可以看出效果。代码显示如下：

import numpy as np

import cv2

# 读取图像位置

filename = r'C:\Users\LBS\Desktop\lena01.png'

# 读取图像,0的意思是读取灰度图像。

data = cv2.imread(filename, 0)

# 打印图像的矩阵

print('图像矩阵data=\n', data)

# 显示原图像

cv2.imshow('01', data)

# 使用cv2.threshold()函数

# 设定阈值x为127，这个阈值是自己定义的哈， maxval=255，type = cv2.THRESH\_BINARY

reval1, data\_THRESH\_BINARY = cv2.threshold(data, 127, 255, type=cv2.THRESH\_BINARY)

print('data\_THRESH\_BINARY=\n', data\_THRESH\_BINARY)

cv2.imshow('02', data\_THRESH\_BINARY)

# 设定阈值x为120， maxval=255，type = cv2.THRESH\_BINARY\_INV

reval2, data\_THRESH\_BINARY\_INV = cv2.threshold(data, 120, 255, type=cv2.THRESH\_BINARY\_INV)

print('data\_THRESH\_BINARY\_INV=\n', data\_THRESH\_BINARY\_INV)

cv2.imshow('03', data\_THRESH\_BINARY\_INV)

# 设定阈值x为200，maxval=255，type = cv2.THRESH\_TRUNC

reval3, data\_THRESH\_TRUNC = cv2.threshold(data, 200, 255, type=cv2.THRESH\_TRUNC)

print('data\_THRESH\_TRUNC=\n', data\_THRESH\_TRUNC)

cv2.imshow('04', data\_THRESH\_TRUNC)

# 设定阈值x为200，maxval=255，type = cv2.THRESH\_TOZERO\_INV

reval4, data\_THRESH\_TOZERO\_INV = cv2.threshold(data, 200, 255, type=cv2.THRESH\_TOZERO\_INV)

print('data\_THRESH\_TOZERO\_INV=\n', data\_THRESH\_TOZERO\_INV)

cv2.imshow('05', data\_THRESH\_TOZERO\_INV)

# 设定阈值x为100，maxval=255，type = cv2.THRESH\_TOZERO

reval5, data\_THRESH\_TOZERO = cv2.threshold(data, 100, 255, type=cv2.THRESH\_TOZERO)

print('data\_THRESH\_TOZERO=\n', data\_THRESH\_TOZERO)

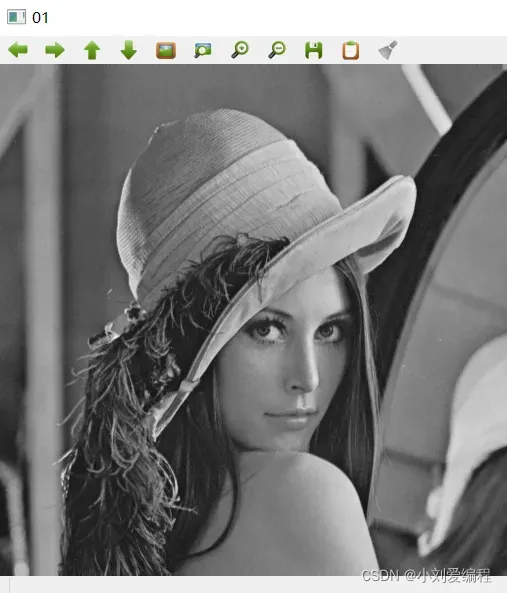
cv2.imshow('06', data\_THRESH\_TOZERO)

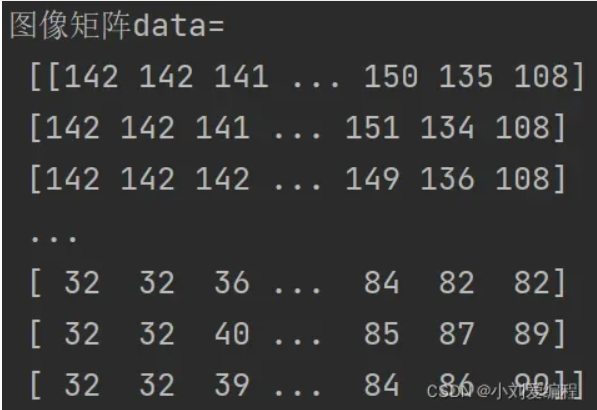
cv2.waitKey()

cv2.destroyAllWindows()

结果：

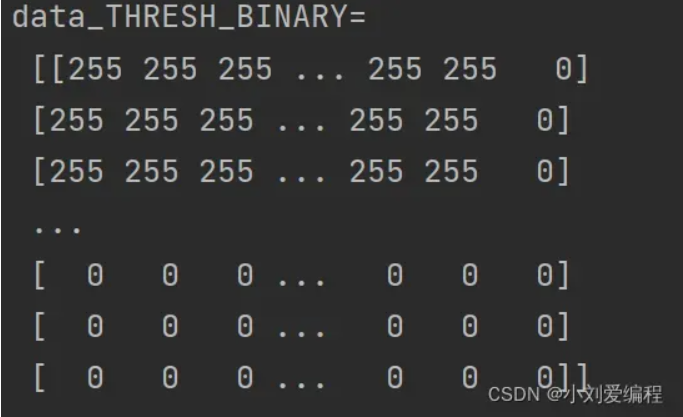
原始图像对应的图像和数组如下图所示。





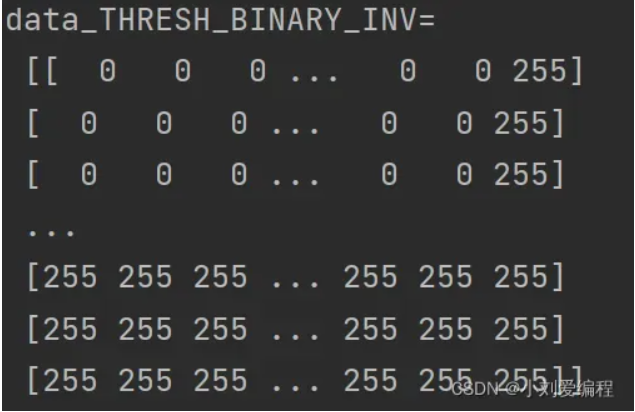
二值化阈值后对应的图像和数组如下图所示。





去二值化阈值化后对应的图像和数组如下图所示。





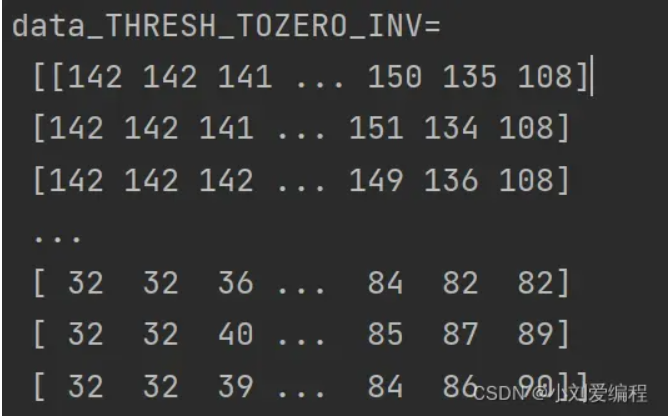
截断阈值后对应的图像和数组如下图所示。





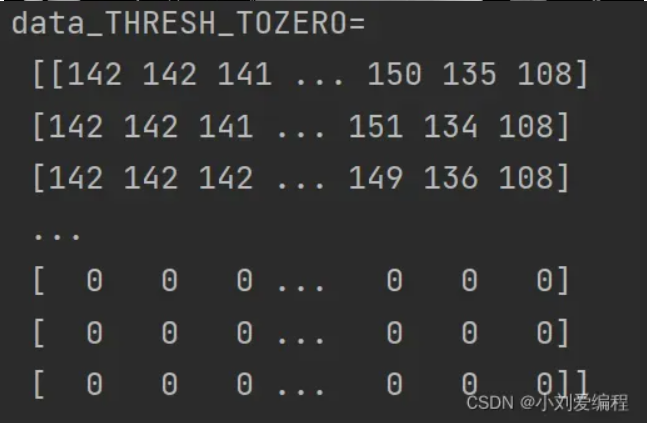
过阈值零处理后对应的图像和数组如下图所示。





低阈值归零后对应的图像和数组如下图所示。





以上就是函数threshold()的使用，大家可以通过更改阈值x的大小进行实践，观察结果图像的效果。

## （二）、adaptiveThreshold()–自适应阈值处理

我们在进行阈值处理时，如果说我们的图像色彩不均衡的话，我们通过一个阈值x，无法进行有效的分割，这时我们就可以使用自适应阈值处理，它就像一个掩膜，关于掩膜概念不懂得话，可以看一下我的[图像处理](https://aitechtogether.com/tag/%e5%9b%be%e5%83%8f%e5%a4%84%e7%90%86)笔记03，这种阈值的处理是根据当前像素点的周围邻域内的像素值的加权平均，获得一个阈值，获得阈值之后的操作就和前面的一样了。

该函数的语法如下：

reval， newimg = cv2.threshold (img,  maxval, adaptiveMethod， type，blocksize，C)

参数说明：与threshhold函数相同的参数这里不再说明，可以看文章开头，我们这里说一下添加的新的参数的意思。

adaptiveMethod：这个参数有表示的是我们获取阈值的方法，有两种，分别是cv2.ADAPTIVE\_THRESH\_MEAN\_C和cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C，即均值掩膜和高斯掩膜，均值掩膜的权值是一样的，高斯掩膜则是通过高斯函数分配权值，只是权值不同，对图像的像素点操作都是加权取平均。然后减去常数C，就得到了当前图像像素点的阈值x，对图像的像素逐个计算出这个阈值。就完成了对一幅图像的阈值处理操作。

type：只能是THRESH\_BINARY\_INV或THRESH\_BINARY

blocksize：指定掩膜的大小，通常为奇数3，5，7…

C：是一个常数

我们通过该函数来处理图像，与thresh函数做一个对比。

代码显示如下：

import numpy as np

import cv2

# 读取图像位置

filename = r'C:\Users\LBS\Desktop\lena01.png'

# 读取图像,0的意思是读取灰度图像。

data = cv2.imread(filename, 0)

# 打印图像的矩阵

print('图像矩阵data=\n', data)

# 显示原图像

cv2.imshow('01', data)

# 使用cv2.threshold()函数

# 设定阈值x为127，这个阈值是自己定义的哈， maxval=255，type = cv2.THRESH\_BINARY

reval1, data\_THRESH\_BINARY = cv2.threshold(data, 127, 255, type=cv2.THRESH\_BINARY)

print('data\_THRESH\_BINARY=\n', data\_THRESH\_BINARY)

cv2.imshow('02', data\_THRESH\_BINARY)

# 使用cv2.adaptiveThreshold()函数,adaptiveMethod=cv2.ADAPTIVE\_THRESH\_MEAN\_C

data\_Mean = cv2.adaptiveThreshold(data, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, blockSize=5, C=3)

print('data\_Mean=\n', data\_Mean)

cv2.imshow('03', data\_Mean)

# 使用cv2.adaptiveThreshold()函数,adaptiveMethod=cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C

data\_GAUSS = cv2.adaptiveThreshold(data, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, blockSize=5, C=3)

print('data\_Mean=\n', data\_GAUSS)

cv2.imshow('04', data\_GAUSS)

cv2.waitKey()

cv2.destroyAllWindows()

结果：

原图是：



threshold函数处理、adaptiveThreshold函数处理的MEAN方法、GAUSSIAN方法结果图如下：







## （三）Otsu处理

这个处理是为了克服我们手动设置的阈值x对当前图像来说不合理的情况。比如当前图像的像素值都大于125，然后我们设置一个阈值为125，用二值化处理，则图像的像素值都会变为255，其图像是一个白板，这样进行阈值处理是不可取的，那么我们可以通过Otsu进行处理，可以根据当前图像中的所有像素值，设定一个最优的阈值，它会综合考虑所有可能的阈值。

在opencv中，我们只需要在函数threshold()中对参数type多添加一个参数cv2.THRESH\_OTSU即可。

声明如下：

reval， newimg = cv2.threshold (img, x, maxval, type+THRESH\_OTSU)

需要注意的是，在使用Otsu方法是，阈值需设为0，函数threshhold会自动寻找最优阈值。

例如：reval， newimg = cv2.threshold (img, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

在实践中看到它。代码显示如下：

import numpy as np

import cv2

# 读取图像位置

filename = r'C:\Users\LBS\Desktop\02.png'

# 读取图像,0的意思是读取灰度图像。

data = cv2.imread(filename, 0)

# 打印图像的矩阵

print('图像矩阵data=\n', data)

# 显示原图像

cv2.imshow('01', data)

# 使用cv2.threshold()函数加上Otsu方法

# 设定阈值x为0，这个阈值需为0， maxval=255，type = cv2.THRESH\_BINARY+cv2.THRESH\_OTSU

reval1, otsu = cv2.threshold(data, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

print('otsu=\n', otsu)

cv2.imshow('02', otsu)

cv2.waitKey()

cv2.destroyAllWindows()

结果如下：

原始图像，处理后的图像，这里我们不指定阈值。





大家可以找一个整体亮度比较高的图像素材，然后通过使用Otsu方法和不使用Otsu方法得到的图像进行比较，你就会知道这个方法的优点所在。

其他参考1：<https://blog.51cto.com/u_15047484/2618452>

其他参考2：<https://www.jianshu.com/p/c39273ad76c1>

其他参考3：<https://cloud.tencent.com/developer/article/1528519?from=15425>