[**OpenCV计算机视觉学习（2）——图像算术运算 &图像阈值（数值计算，掩膜mask操作，边界填充，二值化）**](https://www.cnblogs.com/wj-1314/p/9592346.html)

### 如果需要处理的原图及代码，请移步小编的GitHub地址

　　传送门：[请点击我](https://github.com/LeBron-Jian/ComputerVisionPractice)

　　如果点击有误：<https://github.com/LeBron-Jian/ComputerVisionPractice>

　　在OpenCV中我们经常会遇到一个名字：Mask（掩膜）。很多函数都使用到它，那么这个Mask到底是什么呢，下面我们从图像基本运算开始，一步一步学习掩膜。

### 1，图像算术运算

　　图像的算术运算有很多种，比如两幅图像可以相加，相减，相乘，相除，位运算，平方根，对数，绝对值等；图像也可以放大，缩小，旋转，还可以截取其中的一部分作为ROI（感兴趣区域）进行操作，各个颜色通道还可以分别提取对各个颜色通道进行各种运算操作。总之，对图像可以进行的算术运算非常的多。这里先学习图片间的数学运算，图像混合，按位运算。

#### 1.1 图片加法

　　要叠加两张图片，可以用 cv2.add() 函数，相加两幅图片的形状（高度/宽度/通道数）必须相同， numpy中可以用  res = img1 + img2 相加，但这两者的结果并不相同。

x = np.uint8([250])

y = np.uint8([10])

print(cv2.add(x, y)) # 250+10 = 260 => 255

print(x + y) # 250+10 = 260 % 256 = 4

　　如果是二值化图片（只有0和255），两者结果是一样的（用 numpy的方式更简便一些）。

　　这里我们代入图像中看一下：

#encoding:utf-8

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 举一个极端的例子，真的只是运气好，遇到了。。。。

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

img\_add = img + 10

img\_add2 = cv2.add(img, img\_add)

print(img[0:4, :, 0])

print(img\_add[0:4, :, 0])

print(img\_add2[0:4, :, 0])

'''

这个是 logo1.jpg 的效果

[[246 246 246 ... 246 246 246]

[246 246 246 ... 246 246 246]

[246 246 246 ... 246 246 246]

[246 246 246 ... 246 246 246]]

[[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]]

[[246 246 246 ... 246 246 246]

[246 246 246 ... 246 246 246]

[246 246 246 ... 246 246 246]

[246 246 246 ... 246 246 246]]

这个是 lena.jpg 的效果

[[126 125 124 ... 128 120 90]

[127 126 124 ... 135 131 96]

[124 123 121 ... 144 138 96]

[116 119 116 ... 73 56 35]]

[[136 135 134 ... 138 130 100]

[137 136 134 ... 145 141 106]

[134 133 131 ... 154 148 106]

[126 129 126 ... 83 66 45]]

[[255 255 255 ... 255 250 190]

[255 255 255 ... 255 255 202]

[255 255 252 ... 255 255 202]

[242 248 242 ... 156 122 80]]

# 我们发现 使用numpy库的加法，则运算结果取模

使用opencv的add()函数，则运算结果当大于255，则取255

'''

**注意：OpenCV中的加法与Numpy的加法是有所不同的，OpenCV的加法是一种饱和操作，而Numpy的加法是一种模操作。**

**Numpy库的加法**

　　其运算方法是：目标图像 = 图像1 + 图像2，运算结果进行取模运算

* 当像素值 小于等于  255 时，结果为：“图像1 + 图像2”，例如：120+48=168
* 当像素值 大于255 时，结果为：对255取模的结果，例如：（255 + 64） % 255 = 64

**OpenCV的加法**

　　其运算方法是：目标图像 = cv2.add（图像1， 图像2）

* 当像素值 小于等于  255 时，结果为：“图像1 + 图像2”，例如：120+48=168
* 当像素值 大于255 时，结果为：255，例如：255 + 64 = 255

　　两种方法对应的代码如下：

# encoding:utf-8

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 读取图片

img = cv2.imread('logo1.jpg')

test = img

# 方法一：Numpy加法运算

result1 = img + test

# 方法二：OpenCV加法运算

result2 = cv2.add(img, test)

all\_pic = np.column\_stack((img, result1, result2))

# 显示图像

cv2.imshow('img result1 result2', all\_pic)

# cv2.imshow("original", img)

# cv2.imshow("result1", result1)

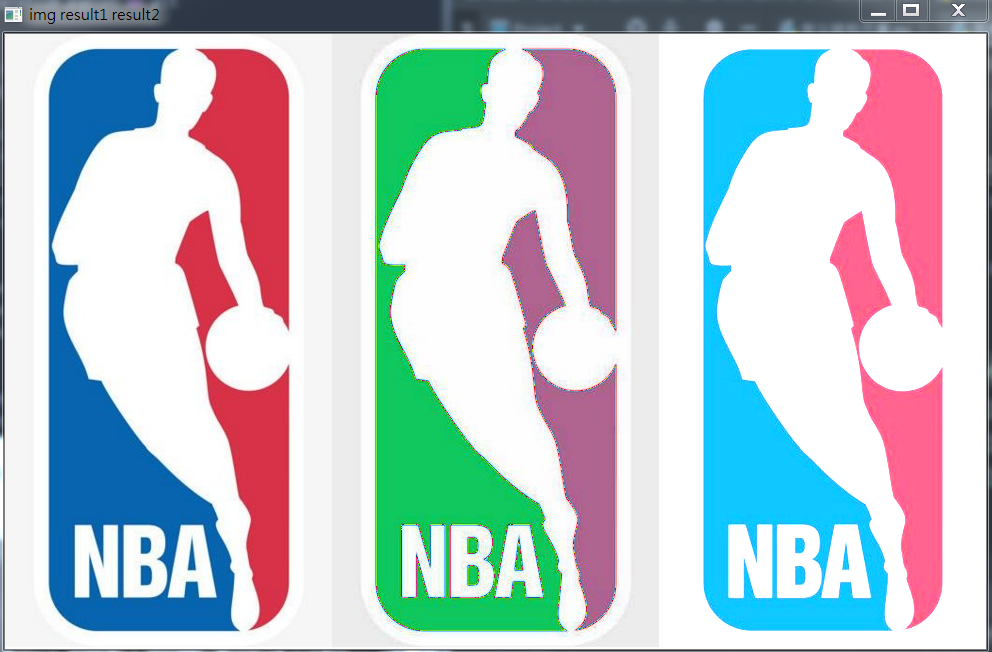
# cv2.imshow("result2", result2)

# 等待显示

cv2.waitKey(0)

cv2.destroyAllWindows()

　　原图及其效果图如下：



 　　其中，result1为Numpy的方法，result2为OpenCV的方法。

#### 1.2  图像混合

　　图像融合通常是指将2张或者两张以上的图像信息融合到1张图像上，融合的图像含有更多的信息，能够更方便人们观察或计算机处理。

　　图像融合是在图像加法的基础上增加了系数和亮度调节量。

**图像融合：目标图像 = 图像1\*系数1 + 图像2\*系数2 + 亮度调节量**

　　图像混合 cv2.addWeighted() 也是一种图片相加的操作，只不过两幅图片的权重不一样， y 相当于一个修正值：

**dst = α\*img1 + β\*img2 + γ**

**PS：当  alpha 和  beta 都等于1，则相当于图片相加。**

　　代码如下：

import cv2

import numpy as np

img1 = cv2.imread('lena\_small.jpg')

img2 = cv2.imread('opencv\_logo\_white.jpg')

# print(img1.shape, img2.shape) # (187, 186, 3) (184, 193, 3)

img2 = cv2.resize(img2, (186, 187))

# print(img1.shape, img2.shape)

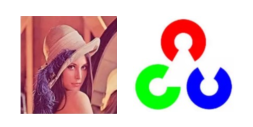
res = cv2.addWeighted(img1, 0.6, img2, 0.4, 0)

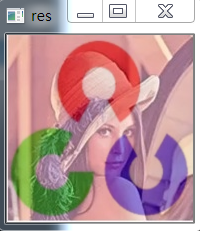
cv2.imshow("res", res)

cv2.waitKey(0)

cv2.destroyAllWindows()

　　注意这里，两张图片的尺寸必须一致。原图和结果图如下：





#### 1.3  图像矩阵减法

　　图像矩阵减法与加法其实类似，我们这不多做说明，只贴函数：

函数原型：cv2.subtract(src1, src2, dst=None, mask=None, dtype=None)

src1：图像矩阵1

src1：图像矩阵2

dst：默认选项

mask：默认选项

dtype：默认选项

#### 1.4  按位运算

　　按位操作有：AND ，OR， NOT，XOR 等。cv2.bitwise\_and(), cv2.bitwise\_not(), cv2.bitwise\_or(), cv2.bitwise\_xor()分别执行按位与/或/非/异或运算。下面我们贴一下opencv中的函数

bitwise\_or—图像或运算

函数原型：cv2.bitwise\_or(src1, src2, dst=None, mask=None)

src1：图像矩阵1

src1：图像矩阵2

dst：默认选项

mask：默认选项

bitwise\_xor—图像异或运算

函数原型：bitwise\_xor(src1, src2, dst=None, mask=None)

src1：图像矩阵1

src1：图像矩阵2

dst：默认选项

mask：默认选项

bitwise\_not—图像非运算

函数原型：bitwise\_not(src1, src2, dst=None, mask=None)

src1：图像矩阵1

src1：图像矩阵2

dst：默认选项

mask：默认选项

　　掩膜就是用来对图片进行全局或局部的遮挡，当我们提取图像的一部分，选择非矩阵ROI时这些操作会很有用，常用于Logo投射。

　　通过 threshold 函数将图片固定阈值二值化（图像二值化定义：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和白的视觉效果）

　　一幅图像包括目标物体，背景还有噪声，要想从多值的数字图像中直接提取出目标物体，常用的方法就是设定一个阈值T，用 T 将图像的数据分为两部分：大于 T 的像素群和小于 T 的像素群。这是研究灰度变换的最特殊的方法，称为图像二值化（Binarization）

　　下面做一个例子，关于Logo投射。（下面首先展示两张照片，一张原图，一张logo图，目的是投射logo到原图上）





　　思路如下：我们的目的是把 logo 放在左边，所以我们只关心这一块区域，下面我们的目的是创建掩码（这是在Logo图上），并且保留除了logo以外的背景（这是在原图），然后进行融合（这是在原图），最后融合放在原图。

 　　代码如下：

# \_\*\_coding:utf-8\_\*\_

import cv2

import numpy as np

img\_photo = cv2.imread('james.jpg')

img\_logo = cv2.imread('logo1.jpg')

print(img\_logo.shape, img\_photo.shape)

# (615, 327, 3) (640, 1024, 3)

rows, cols, channels = img\_logo.shape

photo\_roi = img\_photo[0:rows, 0:cols]

gray\_logo = cv2.cvtColor(img\_logo, cv2.COLOR\_BGR2GRAY)

# 中值滤波

midian\_logo = cv2.medianBlur(gray\_logo, 5)

# mask\_bin 是黑白掩膜

ret, mask\_bin = cv2.threshold(gray\_logo, 127, 255, cv2.THRESH\_BINARY)

# mask\_inv 是反色黑白掩膜

mask\_inv = cv2.bitwise\_not(mask\_bin)

# 黑白掩膜 和 大图切割区域 去取和

img\_photo\_bg\_mask = cv2.bitwise\_and(photo\_roi, photo\_roi, mask=mask\_bin)

# 反色黑白掩膜 和 logo 取和

img2\_photo\_fg\_mask = cv2.bitwise\_and(img\_logo, img\_logo, mask=mask\_inv)

dst = cv2.add(img\_photo\_bg\_mask, img2\_photo\_fg\_mask)

img\_photo[0:rows, 0:cols] = dst

cv2.imshow("mask\_bin", mask\_bin)

cv2.imshow("mask\_inv", mask\_inv)

cv2.imshow("img\_photo\_bg\_mask", img\_photo\_bg\_mask)

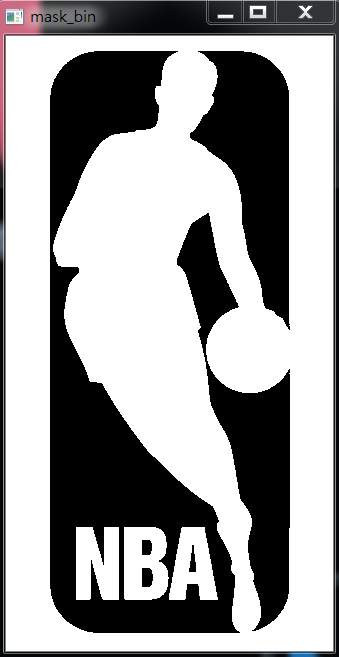
cv2.imshow("img2\_photo\_fg\_mask", img2\_photo\_fg\_mask)

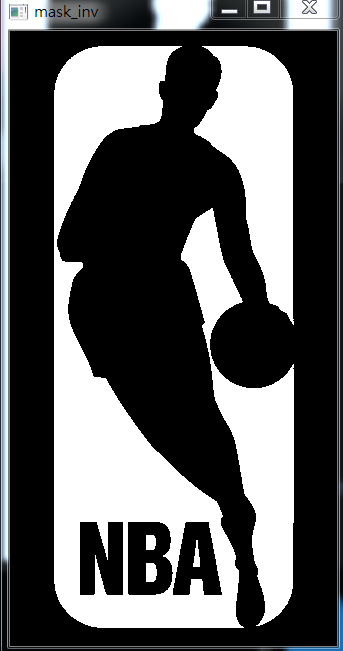
cv2.imshow("img\_photo", img\_photo)

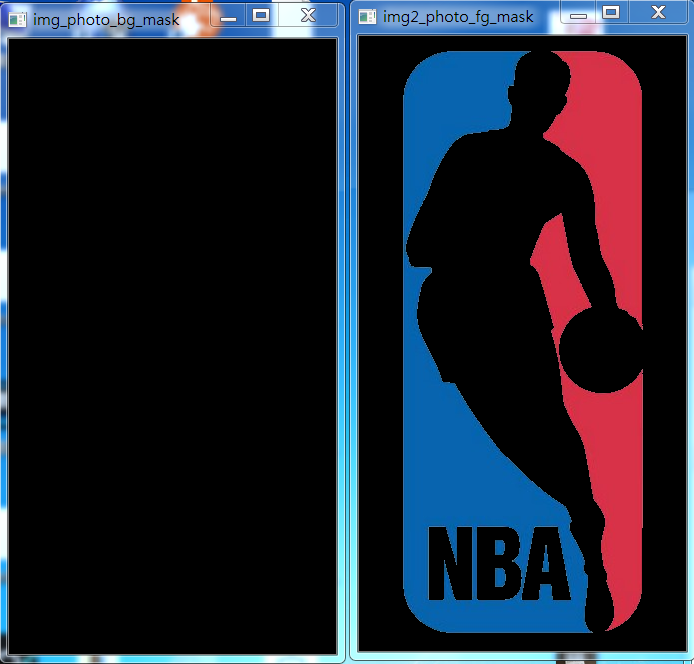
cv2.waitKey(0)

cv2.destroyAllWindows()

　　图示过程如下：





 　　下面第一张是黑色是因为 背景图中 ，左边就是黑色，所以这里不显示而已。

 　　最终形态如下：



### 2，掩膜（mask）

　　在有些图像处理的函数中有的参数里面会有 mask 参数，即此函数支持掩膜操作。

　　首先我们要理解什么是掩膜？，其次掩膜有什么作用呢？

#### 2.1  掩膜（mask）的概念

　　简单来说：掩膜是用一副二值化图片对另外一幅图片进行局部的遮挡。

　　首先我们从物理的角度来看看 mask 到底是什么过程。

　　数字图像处理中的掩膜的概念是借鉴于 PCB 制版的过程，在半导体制作中，许多芯片工艺步骤采用光刻技术，用于这些步骤的图形”底片”称为掩膜（也称为“掩模”），其作用是：在硅片上选定的区域中对一个不透明的图形模板遮盖，继而下面的腐蚀或扩散将只影响选定的区域意外的区域。

　　图形掩膜（Image mask）与其类似，用选定的图形，图形或物体，对处理的图像（全部或局部）进行遮挡，来控制图像处理的区域或处理过程。用于覆盖的特点图像或物体称为**掩膜或模板**。光学图像处理中，掩膜可以足胶片，滤光片等。**掩膜是由0和1组成的一个二进制图像**。当在某一功能中应用掩膜时，1值区域被处理，被屏蔽的0值区域不被包括在计算中。通过制定的数据值，数据范围，有限或无限值，感兴趣区和注释文件来定义图像掩膜，也可以应用上述选项的任意组合作为输入来建立掩膜。

#### 2.2 掩膜的作用

　　数字图像处理中，掩膜为二维矩阵数组，有时也用多值图像，图像掩膜主要用于：

* 1，**提取感兴趣区**，用预先制作的感兴趣区掩膜与待处理图像相乘，得到感兴趣区图像，感兴趣区内图像值保持不变，而区外图像值都为零。
* 2，**屏蔽作用**，用掩膜对图像上某些区域做屏蔽，使其不参加处理或不参加处理参数的计算，或仅对屏蔽区做处理或统计。
* 3，**结构特征提取**，用相似性变量或图像匹配方法检测和提取图像中与掩膜相似的结构特征。
* 4，**特殊性质图像的制作**

　　掩膜是一种图像滤镜的模板，试用掩膜经常处理的是遥感图像。当提取道路或者河流，或者房屋时，通过一个 N\*N 的矩阵来对图像进行像素过滤，然后将我们需要的地物或者标志突出显示出来，这个矩阵就是一种掩膜。在OpenCV中，掩膜操作时相对简单的。大致的意思是，通过一个掩膜矩阵，重新计算图像中的每一个像素值。掩膜矩阵控制了旧图像当前位置以及周围位置像素对新图像当前位置像素值的影响力度。用数学术语将，即我们自定义一个权重表。

　　在所有图像基本运算的操作函数中，凡是带有掩膜（mask）的处理函数，其掩膜都参与运算（输入图像运算完之后再与掩膜图像或矩阵运算）。

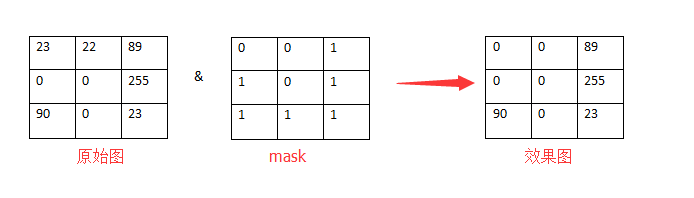
#### 2.3  通过掩膜操作实现图像对比图的改变

 　　矩阵的掩膜操作非常简单，根据掩膜来重新计算每个像素的像素值，掩膜（mask）也被称为内核。

**什么是图和掩膜的与运算呢？**

　　其实就是原图中的每个像素和掩膜中的每个对应像素进行与运算。比如1 & 1 = 1；1 & 0 = 0；

　　比如一个 3\*3 的图像与 3\*3 的掩膜进行运算，得到的结果图像就是：



 　　说白了，mask就是位图，来选择哪个像素允许拷贝，哪个像素不允许拷贝，如果mask像素的值时非0的，我们就拷贝它，否则不拷贝。

#### 2.4  mask小结

　　1，图像中，各种位运算，比如与，或，非运算与普通的位运算类似。

　　2，如果用一句话总结，掩膜就是两幅图像之间进行的各种位运算操作。

代码：

#\_\*\_coding:utf-8\_\*\_

import cv2

import numpy as np

def mask\_processing(path):

image = cv2.imread(path) # 读图

# cv2.imshow("Oringinal", image) #显示原图

print(image.shape[:2]) # (613, 440)

# 输入图像是RGB图像，故构造一个三维数组，四个二维数组是mask四个点的坐标，

site = np.array([[[300, 280], [150, 280], [150, 50], [300, 50]]], dtype=np.int32)

im = np.zeros(image.shape[:2], dtype="uint8") # 生成image大小的全白图

cv2.polylines(im, site, 1, 255) # 在im上画site大小的线，1表示线段闭合，255表示线段颜色

cv2.fillPoly(im, site, 255) # 在im的site区域，填充颜色为255

mask = im

cv2.namedWindow('Mask', cv2.WINDOW\_NORMAL) # 可调整窗口大小，不加这句不可调整

cv2.imshow("Mask", mask)

masked = cv2.bitwise\_and(image, image, mask=mask) # 在模板mask上，将image和image做“与”操作

cv2.namedWindow('Mask to Image', cv2.WINDOW\_NORMAL) # 同上

cv2.imshow("Mask to Image", masked)

cv2.waitKey(0) # 图像一直显示，键盘按任意键即可关闭窗口

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

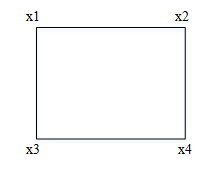
path = 'irving.jpg'

mask\_processing(path)

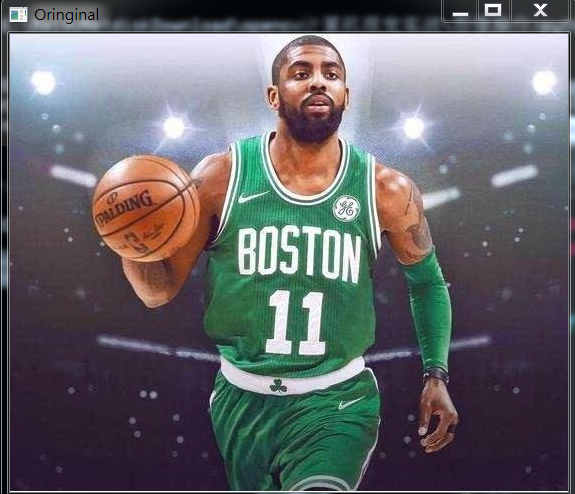
　　代码说明：

1，考虑到当图像尺寸太大，所以我们用 cv2.namedWindow() 函数可以指定窗口是否可以调整大小。在默认情况下，标志为  cv2.WINDOW\_AUTOSIZE。但是，如果指定标志为 cv2.WINDOW\_Normal，则可以调整窗口的大小，这些操作可以让我们的工作更方便一些。

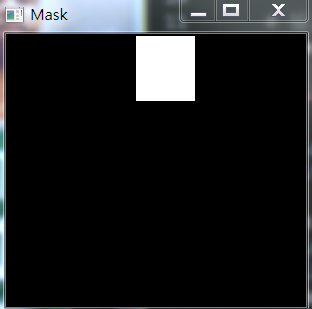
2，对坐标轴的理解，上面代码中的四个坐标从第一个到最后一个分别对应下图中的 x1  x2  x4   x3。（**我实际实验是这样的，如果有不同想法，可以交流**）。



　　原图如下：



 　　mask与处理后图的结果如下：





### 3，边界填充

　　在做深度学习的时候，难免遇到需要填充边界。边缘填充是什么呢？

　　因为对于图像的卷积操作，最边缘的像素一般无法处理，所以卷积核中心倒不了最边缘像素。这就需要先将图像的边界填充，再根据不同的填充算法进行卷积操作，得到的新图像就是填充后的图像。

 　　如果你想在图像周围创建一个边，就像相框一样，你可以使用 cv2.copyMakeBorder() 函数，这经常在卷积运算或 0 填充时被用到，这个函数如下：

def copyMakeBorder(src, top, bottom, left, right, borderType, dst=None, value=None):

　　参数解释：

* src：输入图像
* top，buttom，left，right 对应边界的像素数目（分别为图像上面， 下面， 左面，右面填充边界的长度）
* borderType 要添加哪种类型的边界，类型如下：

　　　　——cv2.BORDER\_CONSTANT 添加有颜色的**常数值边界**，还需要下一个参数（value）

　　　　——cv2.BORDER\_REFLECT 边界元素的镜像，反射法，即以最边缘的像素为对称轴。比如: fedcba|abcdefgh|hgfedcb

　　　　——cv2.BORDER\_REFLECT\_101 or cv2.BORDER\_DEFAULT跟BORDER\_REFLECT类似，但是由区别。例如: gfedcb|abcdefgh|gfedcba

　　　　——cv2.BORDER\_REPLICATE 复制法，重复最后一个元素。例如: aaaaaa|abcdefgh|hhhhhhh

　　　　——cv2.BORDER\_WRAP 不知道怎么说了, 就像这样: cdefgh|abcdefgh|abcdefg

* value 边界颜色，通常用于常量法填充中，即边界的类型是 cv2.BORDER\_CONSTANT，

　　为了更好的理解这几种类型，请看下面代码演示：

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 读取图片

img = cv2.imread('kd1.jpg') # (221, 405, 3)

# img = cv2.imread('lena.jpg') # (263, 263, 3)

# print(img.shape)

# 各个边界需要填充的值, 为了展示效果，这里填充的大一些

top\_size, bottom\_size, left\_size, right\_size = (50, 50, 50, 50)

# 复制法 重复边界，填充 即复制最边缘像素

replicate = cv2.copyMakeBorder(img, top\_size, bottom\_size,

left\_size, right\_size,

borderType=cv2.BORDER\_REPLICATE)

# 反射法 反射边界，填充 即对感兴趣的图像中的像素在两边进行复制，

# 例如 fedcba|abcdefgh|hgfedcb

reflect = cv2.copyMakeBorder(img, top\_size, bottom\_size,

left\_size, right\_size,

borderType=cv2.BORDER\_REFLECT)

# 反射101边界法 反射101边界，填充 这个是以最边缘为轴，对称 ，

# 例如 gfedcb|abcdefg|gfedcba

reflect101 = cv2.copyMakeBorder(img, top\_size, bottom\_size,

left\_size, right\_size,

borderType=cv2.BORDER\_REFLECT\_101)

# 外包装法 填充

# 例如 cdefgh|abcdefgh|abcdegf

wrap = cv2.copyMakeBorder(img, top\_size, bottom\_size,

left\_size, right\_size,

borderType=cv2.BORDER\_WRAP)

# 常量法，常数值填充 ，常量值可以自己设定 value=0

constant = cv2.copyMakeBorder(img, top\_size, bottom\_size,

left\_size, right\_size,

borderType=cv2.BORDER\_CONSTANT,

value=(0, 255, 0))

plt.subplot(231)

plt.imshow(img, 'gray')

plt.title('origin')

plt.subplot(232)

plt.imshow(replicate, 'gray')

plt.title('replicate')

plt.subplot(233)

plt.imshow(reflect, 'gray')

plt.title('reflect')

plt.subplot(234)

plt.imshow(reflect101, 'gray')

plt.title('reflect101')

plt.subplot(235)

plt.imshow(wrap, 'gray')

plt.title('wrap')

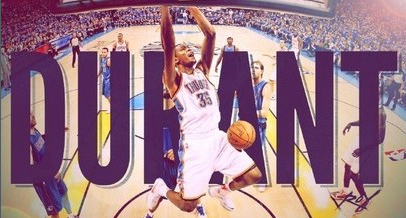
plt.subplot(236)

plt.imshow(constant, 'gray')

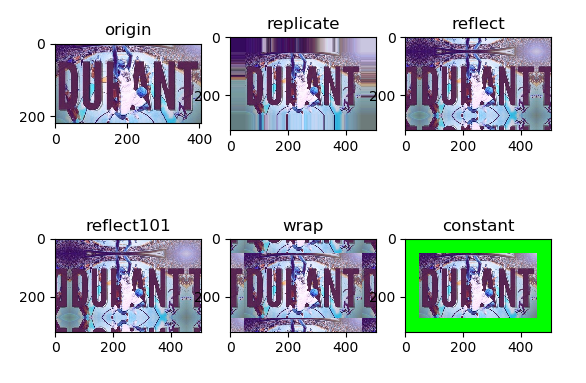
plt.title('constant')

plt.show()

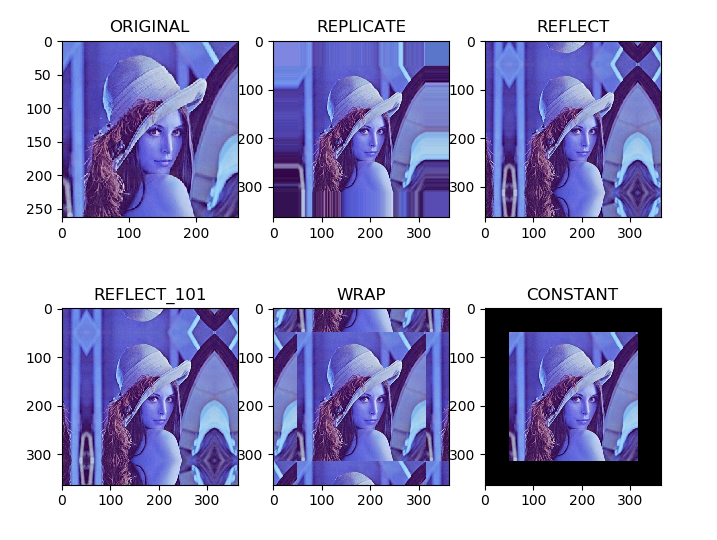
 　　原图1如下：



　　处理的效果图如下：



 　　效果2如下：



#### 注意：plt.imshow() 显示图片色差问题

　　我们都知道  cv2.imshow() 显示的原始图片是BGR格式，即原图如下所示：



 　　那通过opencv将BGR格式转换为RGB格式，图显示如下：



 　　这就解释了为什么plt.imshow()显示图片色差问题，原因就是读取图片的通道不同。

#### 3.1  细节函数

　　为了能快速对比出各个方法得出的图像的区别，可以使用np.vstack()或者np.hstack()对比，将图像放在同一个窗口。

rec=np.hstack((replicate,reflect))

cv\_show("replicate\_reflect",rec)

　　注意：使用np.vstack()或者np.hstack()函数时，图像的大小必须一致，不然会报错。

　　　　    使用np.vstack()或者np.hstack()函数时，可能会出现图像显示不完全情况

### 4，图像阈值（二值化）

#### 4.1  图像二值化原理

　　二值化核心思想，设阈值，大于阈值的为0（黑色）或 255（白色），使图像称为黑白图。

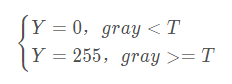
　　阈值可固定，也可以自适应阈值。

　　自适应阈值一般为一点像素与这点为中序的区域像素平均值或者高斯分布加权和的比较，其中可以设置一个差值也可以不设置。

　　图像的阈值化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T将图像的像素分为两类：大于T的像素群和小于T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。所谓图像的二值化 ，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只有黑和白的视觉效果。一幅图像包括目标物体、背景还有噪声，要想从多值的数字图像中直接提取出目标物体。

　　常用的二值化算法下所示：



 　　当灰度Gray小于阈值T的时候，其像素设置为0，表示黑色；当灰度Gray大于或等于阈值T时，其Y值为255，表示白色。

**全局阈值**就是一幅图像包括目标物体、背景还有噪声，要想从多值的数字图像中直接提取出目标物体；常用的方法就是设定一个阈值T，用T将图像的数据分成两部分：大于T的像素群和小于T的像素群。这是研究灰度变换的最特殊的方法，称为图像的二值化（Binarization）。

**局部阈值**就是当同一幅图像上的不同部分的具有不同亮度时。这种情况下我们需要采用自适应阈值。此时的阈值是根据图像上的每一个小区域计算与其对应的阈值。因此在同一幅图像上的不同区域采用的是不同的阈值，从而使我们能在亮度不同的情况下得到更好的结果。

　　二值化处理广泛应用于各行各业，比如生物学中的细胞图分割，交通领域的车牌设计等。在文化应用领域中，通过二值化处理将所需民族文物图像转换为黑白两色图，从而为后面的图像识别提供更好的支撑作用。

#### 4.2  简单阈值处理（全局阈值）

　　Python-OpenCV中提供了阈值（threshold）函数：

threshold(src, thresh, maxval, type, dst=None)

　　变量的作用：

* 第一个参数 src  指原图像，原图像应该是灰度图，**只能输入单通道图像**
* 第二个参数 thresh  指用来对像素值进行分类的阈值
* 第三个参数 maxval 指当像素值高于（有时是小于，根据 type 来决定）阈值时应该被赋予的新的像素值，在二元阈值THRESH\_BINARY和逆二元阈值THRESH\_BINARY\_INV中使用的最大值
* 第四个参数 dst 指不同的不同的阈值方法，这些方法包括以下五种类型：

　　　　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，否则设为零

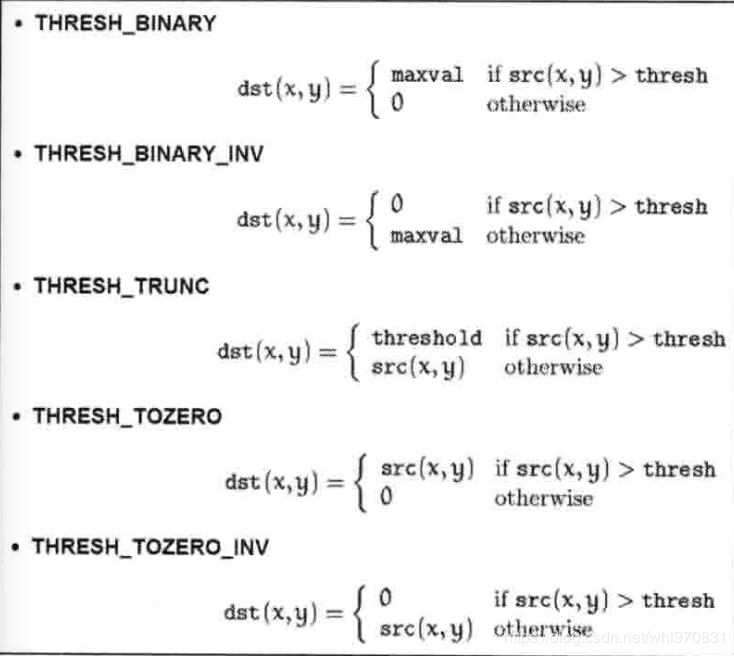
　　　　cv2.THRESH\_TOZERO\_INV       THRESH\_TOZERO 的反转

　　（盗图来自：https://blog.csdn.net/whl970831/article/details/98231314  https://blog.csdn.net/Eastmount/article/details/83548652）

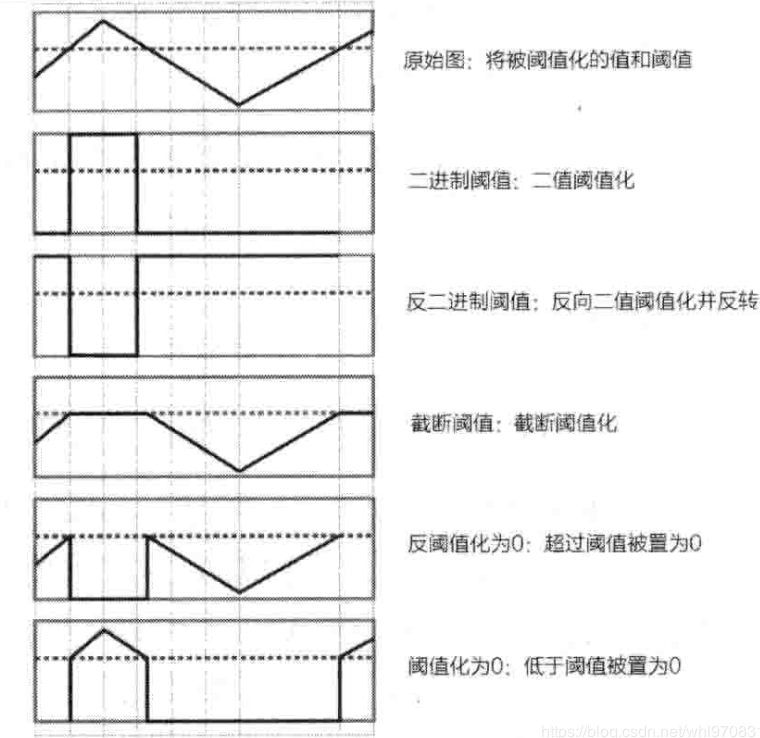
　　详细解析如下：



　　用函数表示如下：



　　对应OpenCV提供的五张图如下，第一张为原图，后面依次为：二进制阈值化，反二进制阈值化，截断阈值化，反阈值化为0，阈值化为0.



 　　代码如下：

# \_\*\_coding:utf-8\_\*\_

import cv2

import numpy as np

from matplotlib import pyplot as plt

def parse\_thresh(path):

img = cv2.imread(path) # 读取原始照片

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

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 = ['Origin Image', 'gray', 'BINARY', 'BINARY\_INV', 'TRUNC', 'TOZERO', 'TOZERO\_INV'] # 标题

images = [img, gray, thresh1, thresh2, thresh3, thresh4, thresh5] # 对应的图

for i in range(7): # 画7次图

plt.subplot(2, 4, i + 1), plt.imshow(images[i], 'gray')

plt.title(titles[i])

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

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

path = 'durant.jpg'

parse\_thresh(path)

　　结果如下：



#### 4.3 自适应阈值处理（局部阈值）

　　Python-OpenCV提供了自适应阈值函数：

cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C, dst=None)

　　参数意义：

* 第一个参数 src 指原图像，原图像应该是灰度图。
* 第二个参数 x 指当像素值高于（有时是小于）阈值时应该被赋予的新的像素值
* 第三个参数 adaptive\_method 参数为：

　　　　CV\_ADAPTIVE\_THRESH\_MEAN\_C  
　　　　CV\_ADAPTIVE\_THRESH\_GAUSSIAN\_C

* 第四个参数 threshold\_type 指取阈值类型：必须是下者之一

　　　　CV\_THRESH\_BINARY  
　　　　CV\_THRESH\_BINARY\_INV

* 第五个参数 block\_size 指用来计算阈值的象素邻域大小: 3, 5, 7, …
* 第六个参数 param1 指与方法有关的参数。

　　　　对方法CV\_ADAPTIVE\_THRESH\_MEAN\_C 和 CV\_ADAPTIVE\_THRESH\_GAUSSIAN\_C， 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。



 　　代码如下：

import cv2

from matplotlib import pyplot as plt

#详细说明参考上方例子

img = cv2.imread('sss.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 Thresholding', '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.xticks([]),plt.yticks([])

plt.show()

　　综合代码：

import cv2 as cv

#全局阈值

def threshold\_demo(image):

gray = cv.cvtColor(image, cv.COLOR\_RGB2GRAY) #把输入图像灰度化

ret, binary = cv.threshold(gray, 0, 255, cv.THRESH\_BINARY | cv.THRESH\_TRIANGLE) #直接阈值化是对输入的单通道矩阵逐像素进行阈值分割。

#print("threshold value %s"%ret)

cv.namedWindow("threshold", cv.WINDOW\_NORMAL)

cv.imshow("threshold", binary)

#局部阈值

def local\_threshold(image):

gray = cv.cvtColor(image, cv.COLOR\_RGB2GRAY) #把输入图像灰度化

binary = cv.adaptiveThreshold(gray, 255, cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv.THRESH\_BINARY, 25, 10) #自适应阈值化能够根据图像不同区域亮度分布，改变阈值

cv.namedWindow("adaptiveThreshold", cv.WINDOW\_NORMAL)

cv.imshow("adaptiveThreshold", binary)

src = cv.imread('sss.jpg')

cv.namedWindow('input\_image', cv.WINDOW\_NORMAL) #设置为WINDOW\_NORMAL可以任意缩放

cv.imshow('input\_image', src) #源图

threshold\_demo(src) #全局

local\_threshold(src) #局部

cv.waitKey(0)

cv.destroyAllWindows()

### 5， Otsu 二值化

　　在使用全局阈值时，我们就是随便给了一个数来做阈值，那我们怎么知道我们选取的这个数的好坏呢？答案就是不停的尝试。如果是一副双峰图像（简单来说双峰图像是指图像直方图中存在两个峰）呢？我们岂不是应该在两个峰之间的峰谷选一个值作为阈值？这就是 Otsu 二值化要做的。简单来说就是对一副双峰图像自动根据其直方图计算出一个阈值。（对于非双峰图像，这种方法得到的结果可能会不理想）。  
　　这里用到到的函数还是 cv2.threshold()，但是需要多传入一个参数（flag）：cv2.THRESH\_OTSU。  
　　这时要把阈值设为 0。然后算法会找到最优阈值，这个最优阈值就是返回值 retVal。如果不使用 Otsu 二值化，返回的retVal 值与设定的阈值相等。  
　　下面的例子中，输入图像是一副带有噪声的图像。第一种方法，设127 为全局阈值。第二种方法，直接使用 Otsu 二值化。第三种方法，先使用一个 5x5 的高斯核除去噪音，然后再使用 Otsu 二值化。

　　代码：

import cv2

from matplotlib import pyplot as plt

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

# 设127 为全局阈值

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

# 直接使用 Otsu 二值化

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

# 先使用一个 5x5 的高斯核除去噪音，然后再使用 Otsu 二值化

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

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

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 range(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()

#### 5.1  Otsu 最大类间方差法原理

　　OTSU 算法是由日本学者 OTSU 于 1979 年提出的一种对图像进行二值化的高效算法。OTSU算法又叫大津算法，其本质是最大类间方差法。

 　　它的原理是利用阈值将原图像分为前景，背景两个图像。

**前景**：用 n1，csum，m1 来表示在当前阈值下的前景的点数，质量距，平均灰度。

**背景**：用n2，sum-csum，m2 来表示在当前阈值下的背景的点数，质量距，平均灰度。

　　当取最佳阈值时，背景应该与前景差别最大，关键在于如何选择衡量差别的标准，而在otsu算法中这个衡量差别的标准就是最大类间方差。

#### 5.2  Otsu 最大类间方差法的性能

　　类间方差法对噪音和目标大小十分敏感，它仅对类间方差为单峰的图像产生较好的分割效果。

　　当目标与背景的大小比例悬殊时，类间方差准则可能呈现双峰或多峰，此时效果不好，但是类间方差法是用时最少的。

#### 5.3  Otsu 最大类间方差法的公式推导

　　记 t 为前景与背景的分割阈值，前景点数占图像比例为 w0，平均灰度为 u0；背景点数占图像比例为 w1，平均灰度为 u1.

　　则图像的总平均灰度为： u = w0 \* u0 + w1 \* u1

　　前景和背景图像的方差：g = w0 \* (u0 - u) \* (u0 - u) + w1 \* (u1 - u) \* (u1 - u) = w0 \* w1 \* (u0 - u1) \* (u0 - u1)

 　　当方差 g 最大时，可以认为此时前景和背景差异最大，此时的灰度 t 是最佳阈值 sb = w0 \* w1\*(u1 - u0)\*(u0 - u1)

 　　代码实现：

# \_\*\_coding:utf-8\_\*\_

import cv2

import numpy as np

def max\_class\_threshold\_variance(origin\_photo):

img = cv2.imread(origin\_photo, -1)

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

retval, dst = cv2.threshold(gray, 0, 255, cv2.THRESH\_OTSU)

cv2.imshow("src", img)

cv2.imshow("gray", gray)

cv2.imshow("dst", dst)

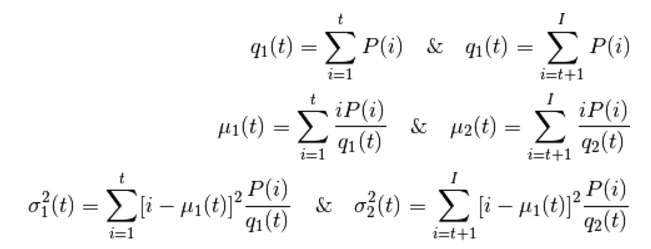
cv2.waitKey(0)

#### ****5.4  Otsu’ ’s 二值化是如何工作的？****

　　 在这一部分我们会演示怎样使用 Python 来实现 Otsu 二值化算法，从而告诉大家它是如何工作的。如果你不感兴趣的话可以跳过这一节。因为是双峰图，Otsu 算法就是要找到一个阈值（t）, 使得同一类加权方差最小，需要满足下列关系式：

## https://img2018.cnblogs.com/i-beta/1226410/201912/1226410-20191204183457164-1206276828.png

 　　其中：



 　　其实就是在两个峰之间找到一个阈值 t，将这两个峰分开，并且使每一个峰内的方差最小。实现这个算法的 Python 代码如下：

img = cv2.imread('noisy2.png',0)

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

# find normalized\_histogram, and its cumulative distribution function

hist = cv2.calcHist([blur],[0],None,[256],[0,256])

hist\_norm = hist.ravel()/hist.max()

Q = hist\_norm.cumsum()

bins = np.arange(256)

fn\_min = np.inf

thresh = -1

for i in xrange(1,256):

p1,p2 = np.hsplit(hist\_norm,[i]) # probabilities

q1,q2 = Q[i],Q[255]-Q[i] # cum sum of classes

b1,b2 = np.hsplit(bins,[i]) # weights

# finding means and variances

m1,m2 = np.sum(p1\*b1)/q1, np.sum(p2\*b2)/q2

v1,v2 = np.sum(((b1-m1)\*\*2)\*p1)/q1,np.sum(((b2-m2)\*\*2)\*p2)/q2

# calculates the minimization function

fn = v1\*q1 + v2\*q2

if fn < fn\_min:

fn\_min = fn

thresh = i

# find otsu's threshold value with OpenCV function

ret, otsu = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

prin(thresh,ret)

参考文献：https://blog.csdn.net/weixin\_42338058/article/details/88568704

 按位运算参考：https://blog.51cto.com/devops2016/2088574

https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_core/py\_image\_arithmetics/py\_image\_arithmetics.html