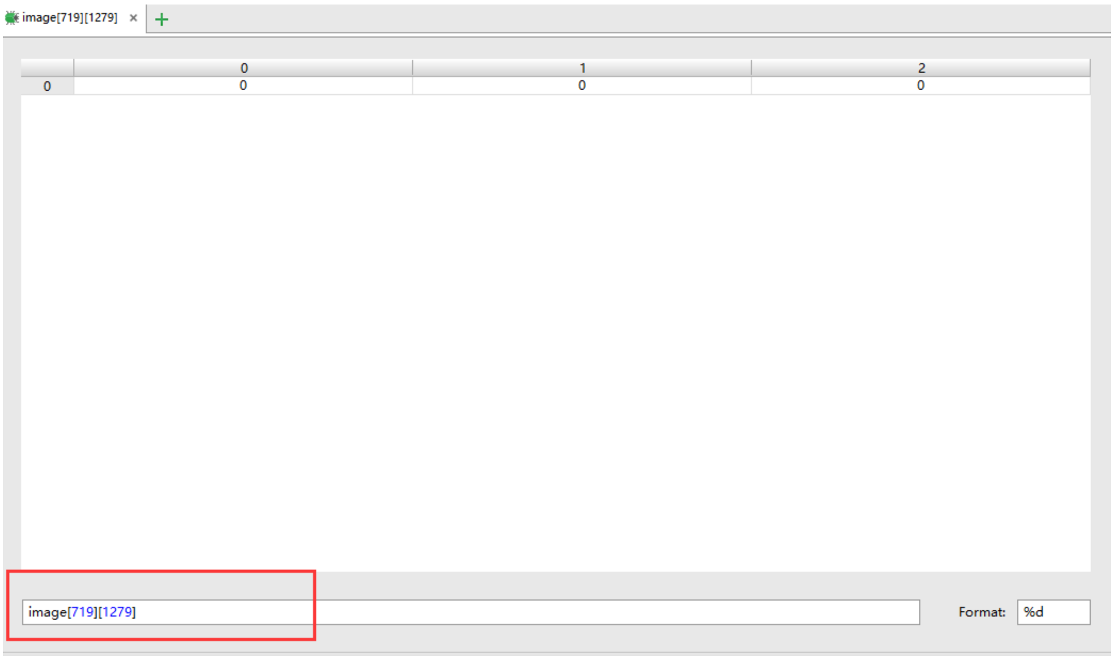
1. 图片的加载

每个像素可有各自的颜色值，可采三原色显示，因而又分成红、绿、蓝三种子像素（RGB 色域）。使用 PIL 读入图片之后使用 numpy 转换为矩阵：

Image 是一个“二维数组”，这个数组中的每一个元素又是一个 RGB 三原色组成像素点(分别是 0-255 之间的数值)

image[719][1279]

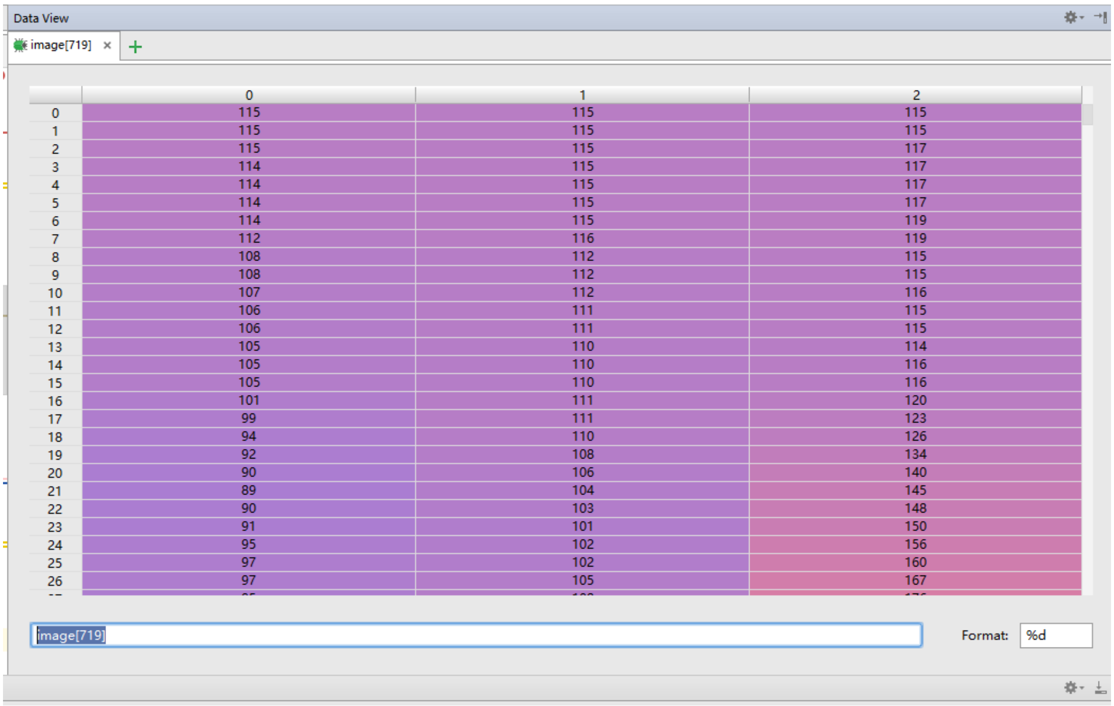


	0	1	2
0	0	0	0

image[719][1279]

Format: %d

image[719]



	0	1	2
0	115	115	115
1	115	115	115
2	115	115	117
3	114	115	117
4	114	115	117
5	114	115	117
6	114	115	119
7	112	116	119
8	108	112	115
9	108	112	115
10	107	112	116
11	106	111	115
12	106	111	115
13	105	110	114
14	105	110	116
15	105	110	116
16	101	111	120
17	99	111	123
18	94	110	126
19	92	108	134
20	90	106	140
21	89	104	145
22	90	103	148
23	91	101	150
24	95	102	156
25	97	102	160
26	97	105	167

image[719]

Format: %d

2. 彩色图像自动阈值分割

参考：

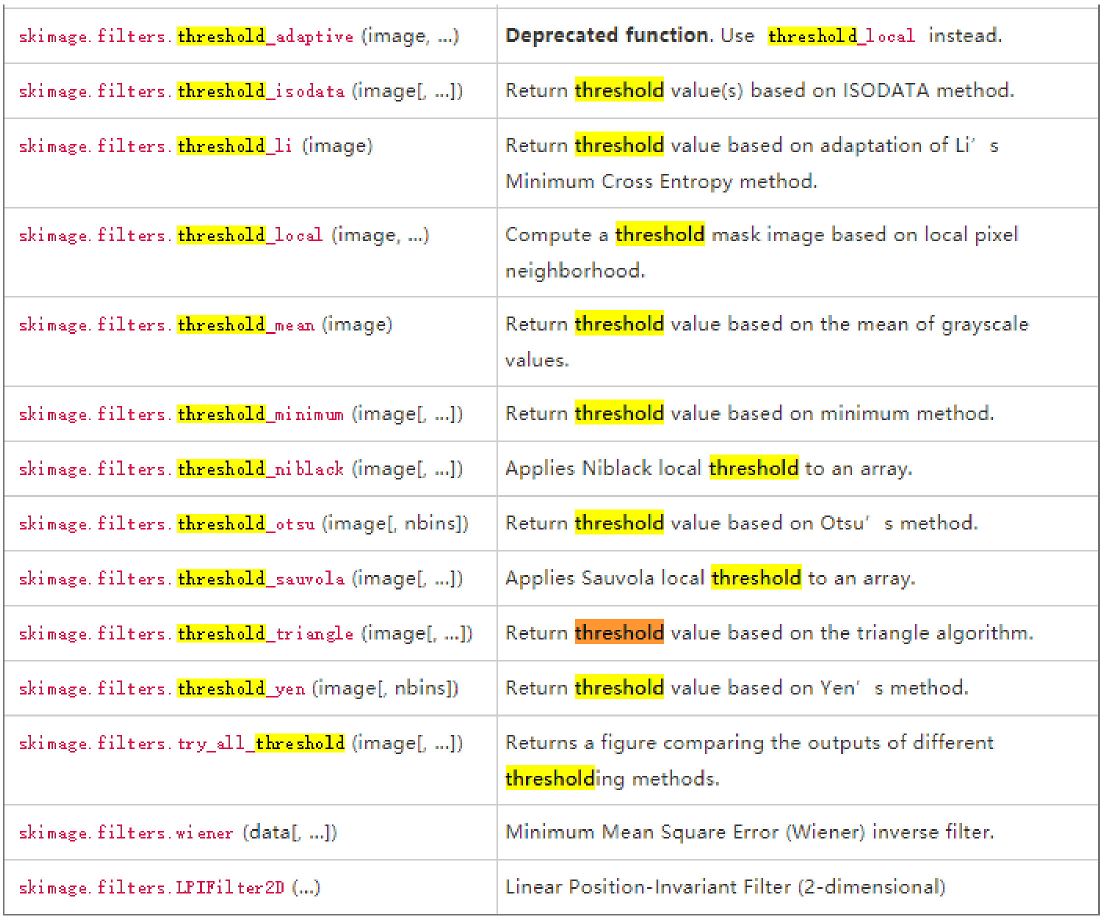
http://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.threshold_adaptive
http://imagej.net/Auto_Threshold#Available_methods

多通道图像的简单分割，可以给定阈值向量，然后给定范围，可以是三维的球形，或者立方体，这个就要看具体的设计了，比如举个简单的例子，给定 RGB 中心阈值为 $\vec{T}$(R0,G0,B0)，阈值为 100， 那么对于像素点(x,y)处的色彩向量 $\vec{I}$(Rxy,Gxy,Bxy)
那么只要满足

$|\vec{T} - \vec{I}|<100$

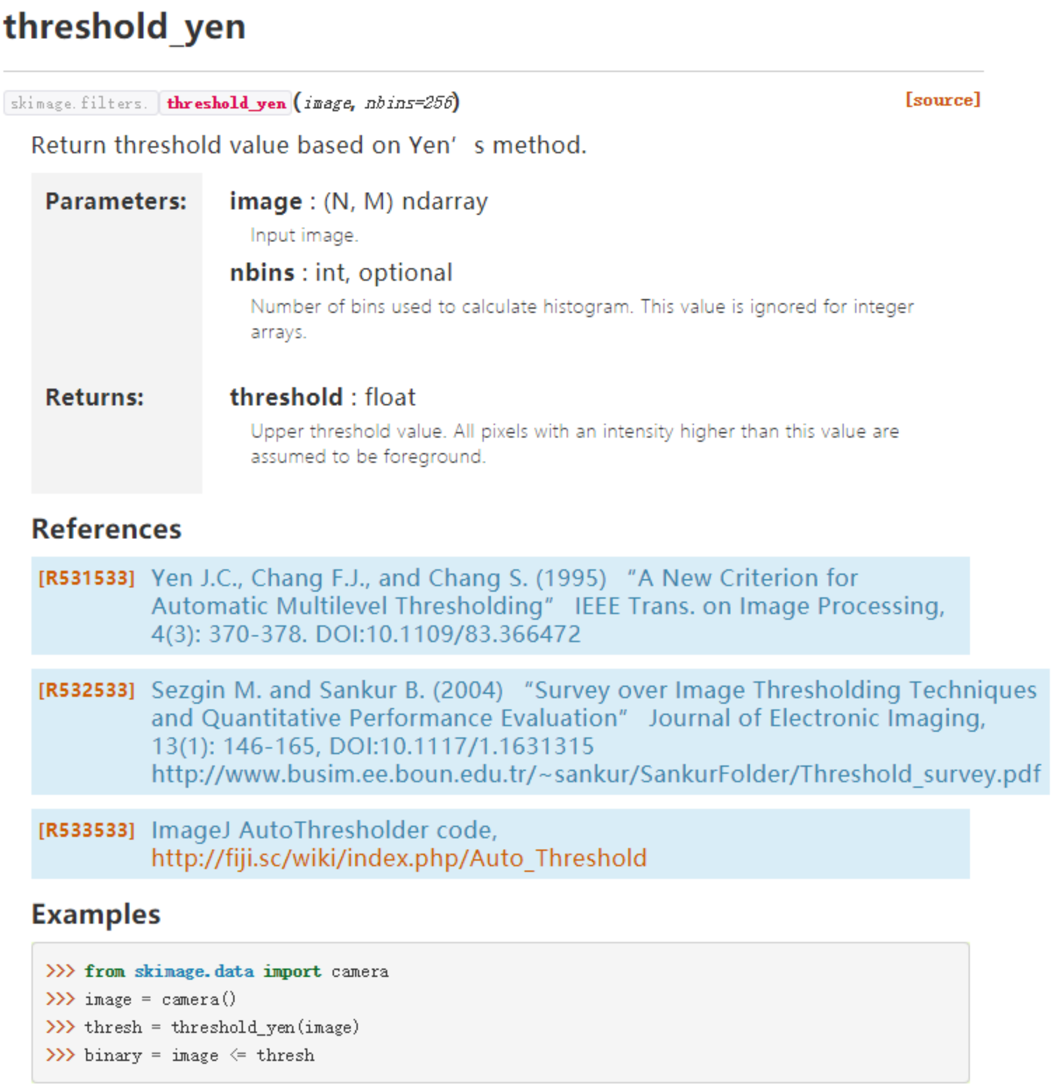
的点满足要求，为目标点，否则为背景点。
算法同样适用于其他色彩空间，但要根据具体情况来设计，所以灵活性很强。

关于自动阈值算法我们研究了几种：



skimage.filters.threshold_adaptive (image, ...)	**Deprecated function**. Use threshold_local instead.
skimage.filters.threshold_isodata (image[, ...])	Return threshold value(s) based on ISODATA method.
skimage.filters.threshold_li (image)	Return threshold value based on adaptation of Li' s Minimum Cross Entropy method.
skimage.filters.threshold_local (image, ...)	Compute a threshold mask image based on local pixel neighborhood.
skimage.filters.threshold_mean (image)	Return threshold value based on the mean of grayscale values.
skimage.filters.threshold_minimum (image[, ...])	Return threshold value based on minimum method.
skimage.filters.threshold_niblack (image[, ...])	Applies Niblack local threshold to an array.
skimage.filters.threshold_otsu (image[, nbins])	Return threshold value based on Otsu' s method.
skimage.filters.threshold_sauvola (image[, ...])	Applies Sauvola local threshold to an array.
skimage.filters.threshold_triangle (image[, ...])	Return threshold value based on the triangle algorithm.
skimage.filters.threshold_yen (image[, nbins])	Return threshold value based on Yen' s method.
skimage.filters.try_all_threshold (image[, ...])	Returns a figure comparing the outputs of different thresholding methods.
skimage.filters.wiener (data[, ...])	Minimum Mean Square Error (Wiener) inverse filter.
skimage.filters.LPIFilter2D (...)	Linear Position-Invariant Filter (2-dimensional)

这个阈值到底怎么样确定比较好？我们使用的是 from skimage.filters import threshold_yen 函数，基于统计学的方法自动给我们确定了一个阈值，可以减少光线等其他的影响。



threshold_yen

skimage.filters.threshold_yen(*image*, *nbins=256*) [source]

Return threshold value based on Yen’ s method.

Parameters:	**image** : (N, M) ndarray Input image. **nbins** : int, optional Number of bins used to calculate histogram. This value is ignored for integer arrays.
Returns:	**threshold** : float Upper threshold value. All pixels with an intensity higher than this value are assumed to be foreground.

References

[R531533] Yen J.C., Chang F.J., and Chang S. (1995) “A New Criterion for Automatic Multilevel Thresholding” IEEE Trans. on Image Processing, 4(3): 370-378. DOI:10.1109/83.366472

[R532533] Sezgin M. and Sankur B. (2004) “Survey over Image Thresholding Techniques and Quantitative Performance Evaluation” Journal of Electronic Imaging, 13(1): 146-165, DOI:10.1117/1.1631315 http://www.busim.ee.boun.edu.tr/~sankur/SankurFolder/Threshold_survey.pdf

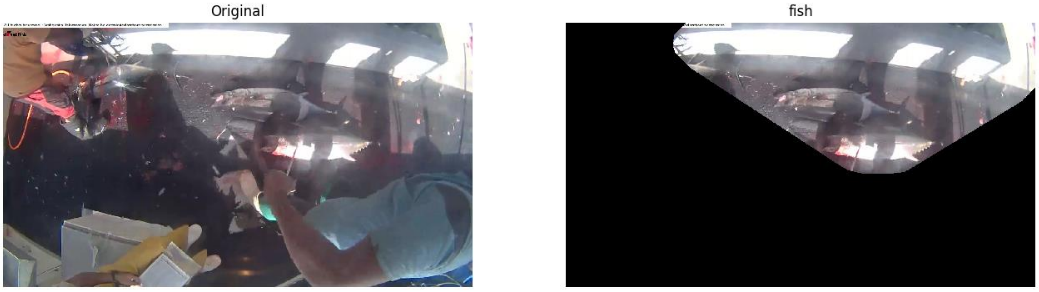
[R533533] ImageJ AutoThresholder code, http://fiji.sc/wiki/index.php/Auto_Threshold

Examples

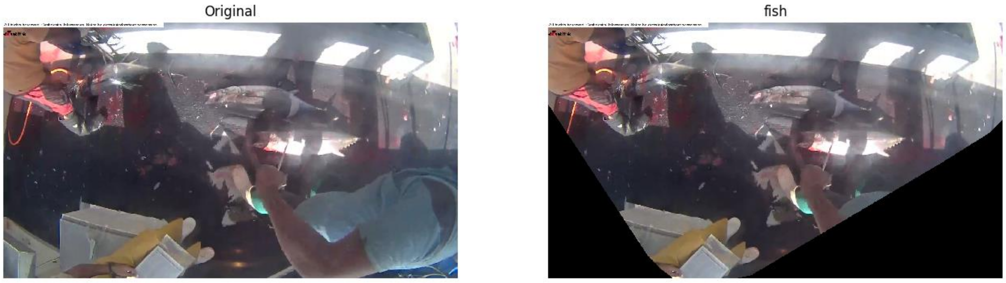
```
>>> from skimage.data import camera
>>> image = camera()
>>> thresh = threshold_yen(image)
>>> binary = image <= thresh
```

对于第一张图片使用 threshold_yen 识别的结果：

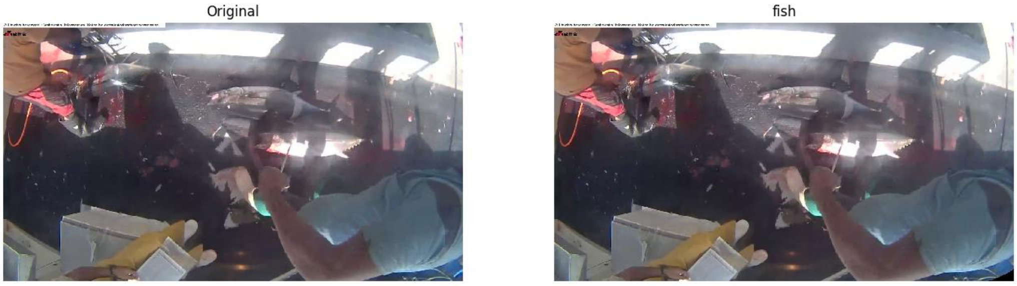
使用 threshold_triangle



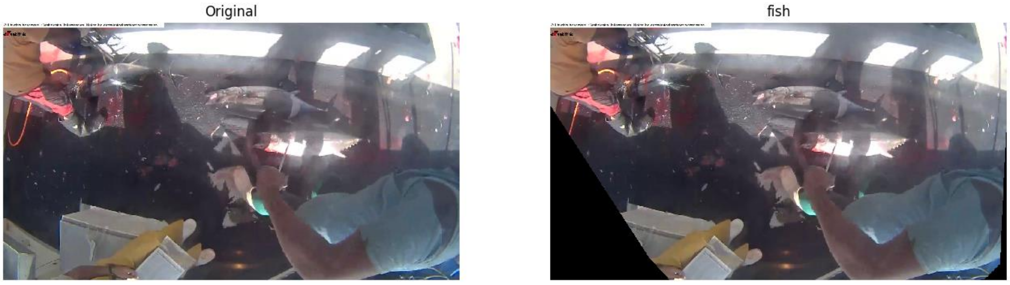
threshold_otsu



threshold_li

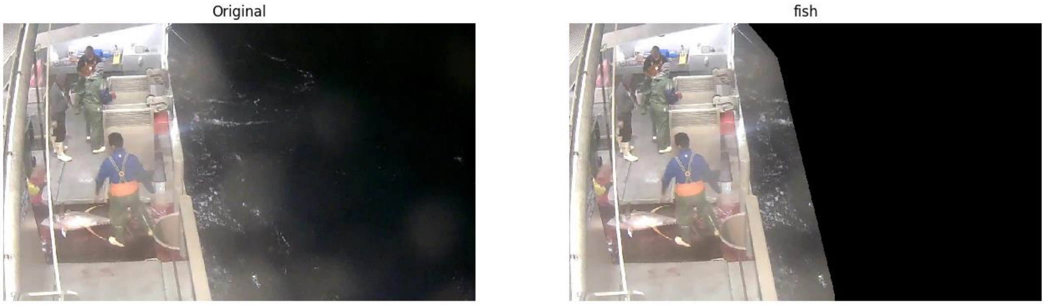


threshold_isodata

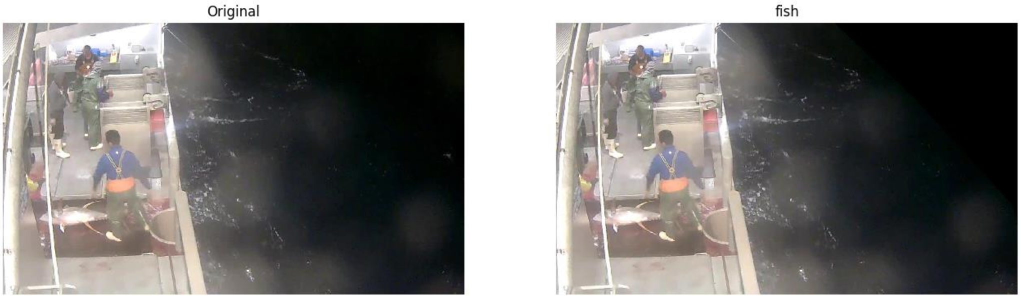


对于第二张图片

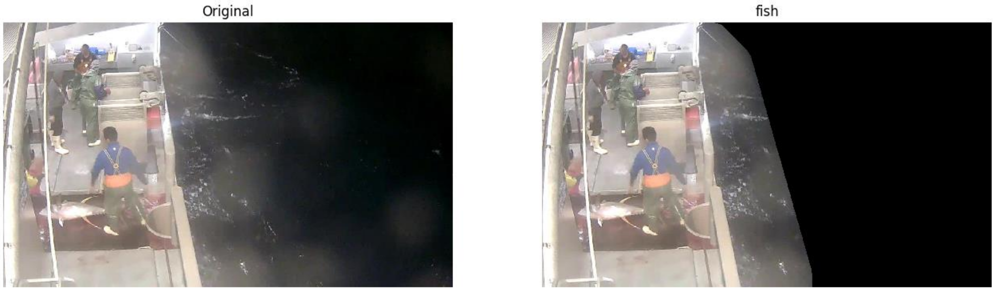
threshold_yen



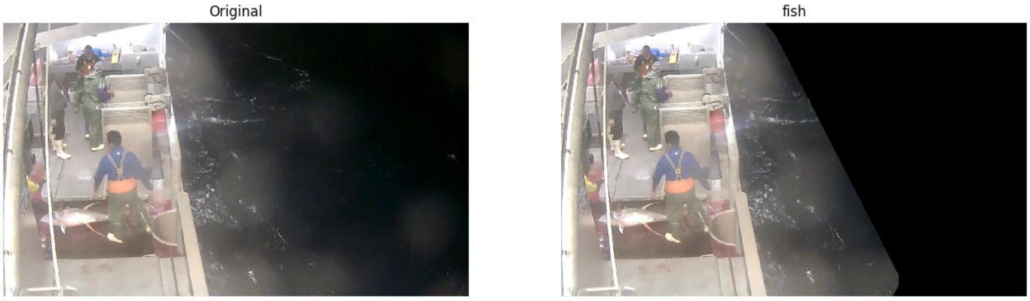
`threshold_triangle`



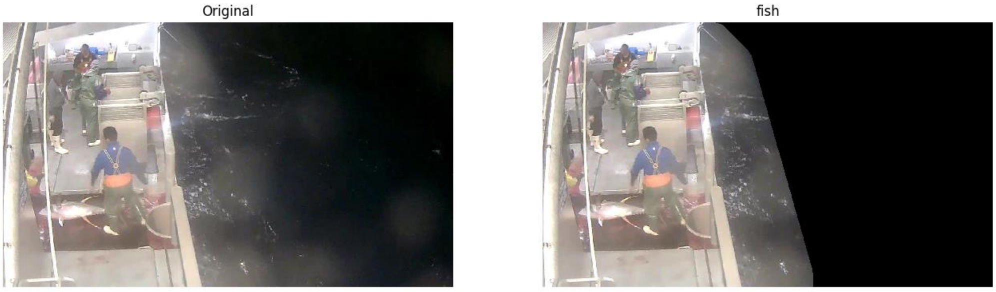
`threshold_otsu`



`threshold_li`

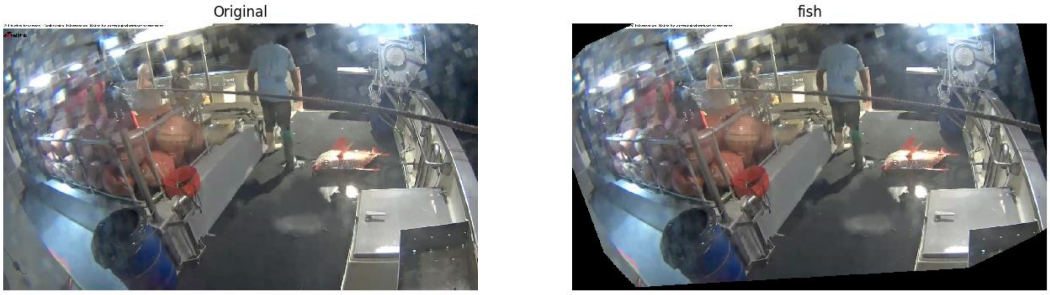


`threshold_isodata`

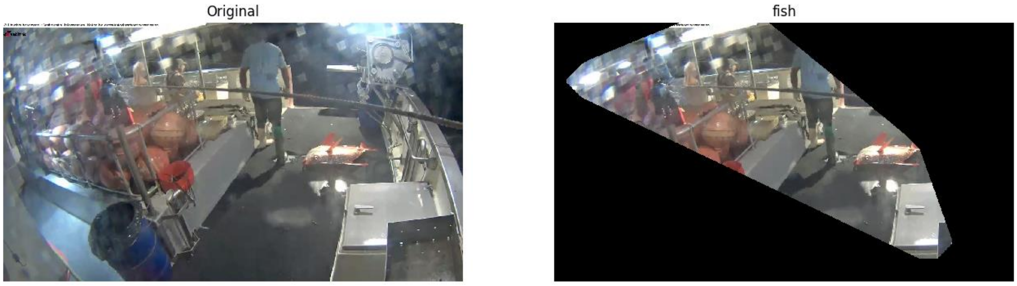


对于第三张图片

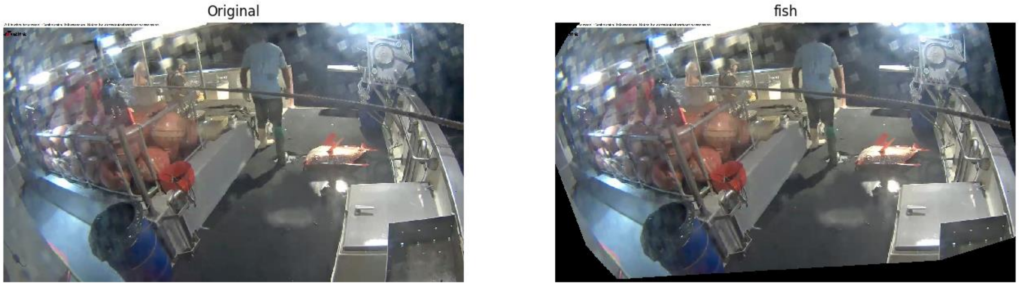
`threshold_yen`



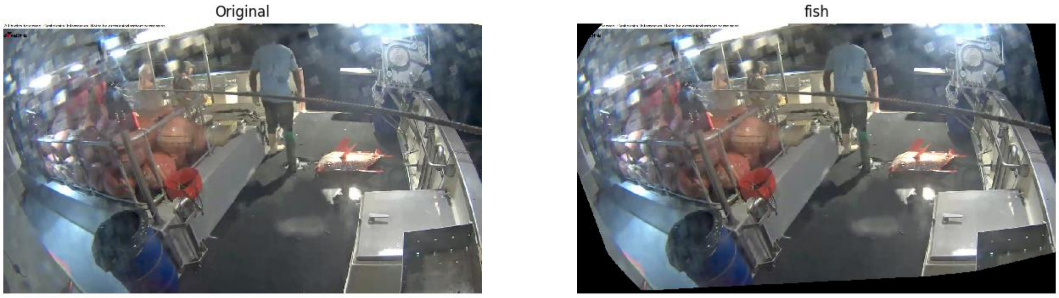
`threshold_triangle`



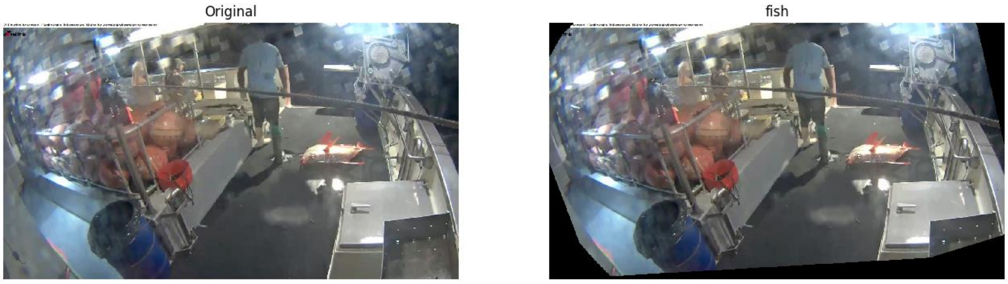
`threshold_otsu`



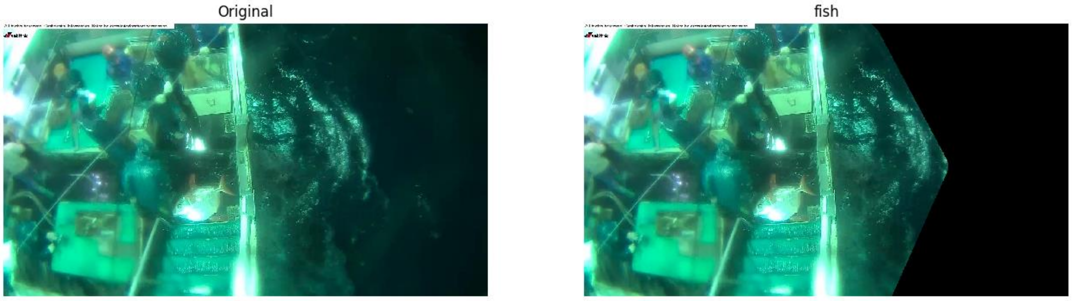
`threshold_li`

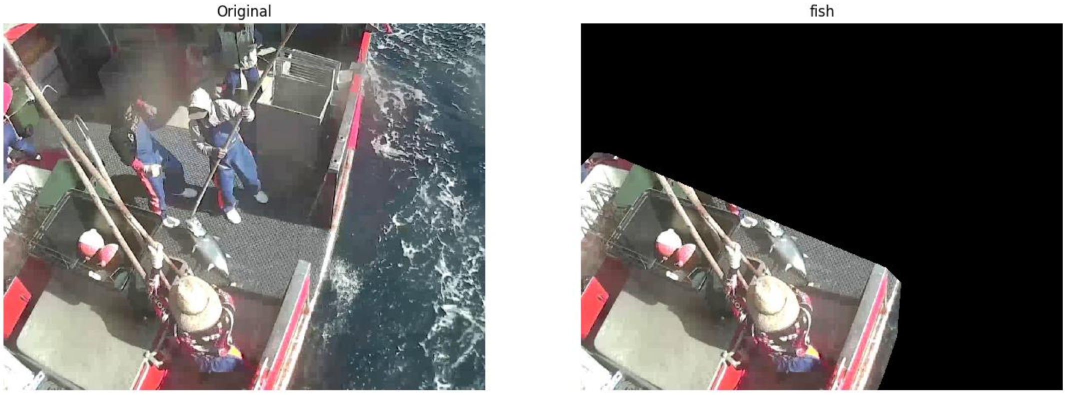


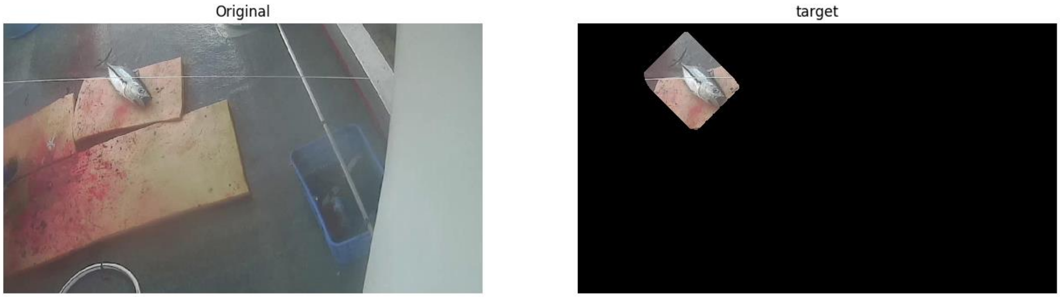
`threshold_isodata`



`threshold_triangle`

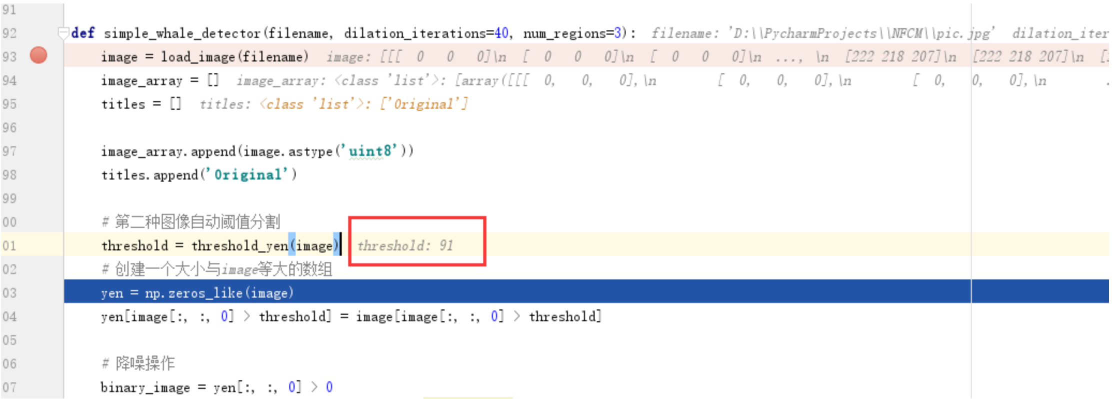






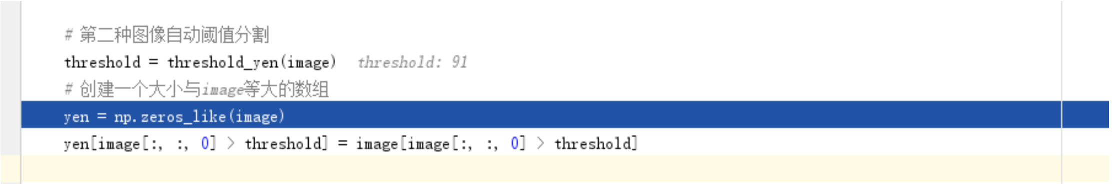
经过多次类似的对比我们初步认定 `threshold_triangle` 目前表现较好的自动阈值分割算法，最终我们使用损失函数来度量各个算法

我们以一张图片为例，经过这个方法自动计算出的阈值：



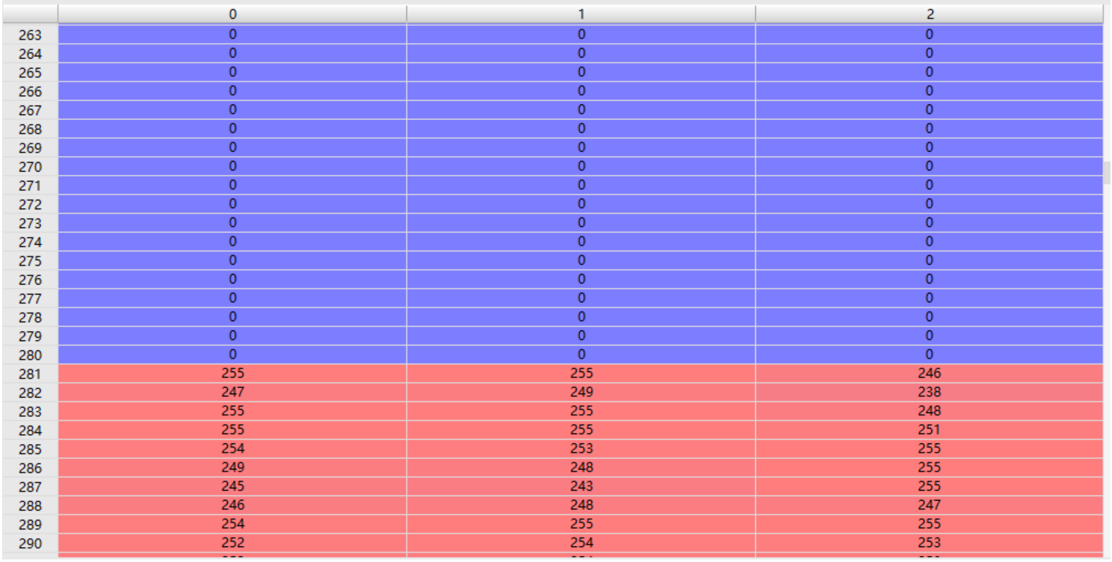
```
def simple_whale_detector(filename, dilation_iterations=40, num_regions=3):  filename: 'D:\\PycharmProjects\\NFCM\\pic.jpg'  dilation_iter
    image = load_image(filename)  image: [[[  0    0    0]\n  [  0    0    0]\n  [  0    0    0]\n  ...,  \n  [222 218 207]\n  [222 218 207]\n  [
    image_array = []  image_array: <class 'list'>: [array([[[  0,    0,    0],\n        [  0,    0,    0],\n        [  0,   0,    0],\n
    titles = []  titles: <class 'list'>: ['Original']

    image_array.append(image.astype('uint8'))
    titles.append('Original')

    # 第二种图像自动阈值分割
    threshold = threshold_yen(image)  threshold: 91
    # 创建一个大小与image等大的数组
    yen = np.zeros_like(image)
    yen[image[:, :, 0] > threshold] = image[image[:, :, 0] > threshold]

    # 降噪操作
    binary_image = yen[:, :, 0] > 0
```

然后我们筛选出像素点大于阈值的



```
# 第二种图像自动阈值分割
threshold = threshold_yen(image)  threshold: 91
# 创建一个大小与image等大的数组
yen = np.zeros_like(image)
yen[image[:, :, 0] > threshold] = image[image[:, :, 0] > threshold]
```

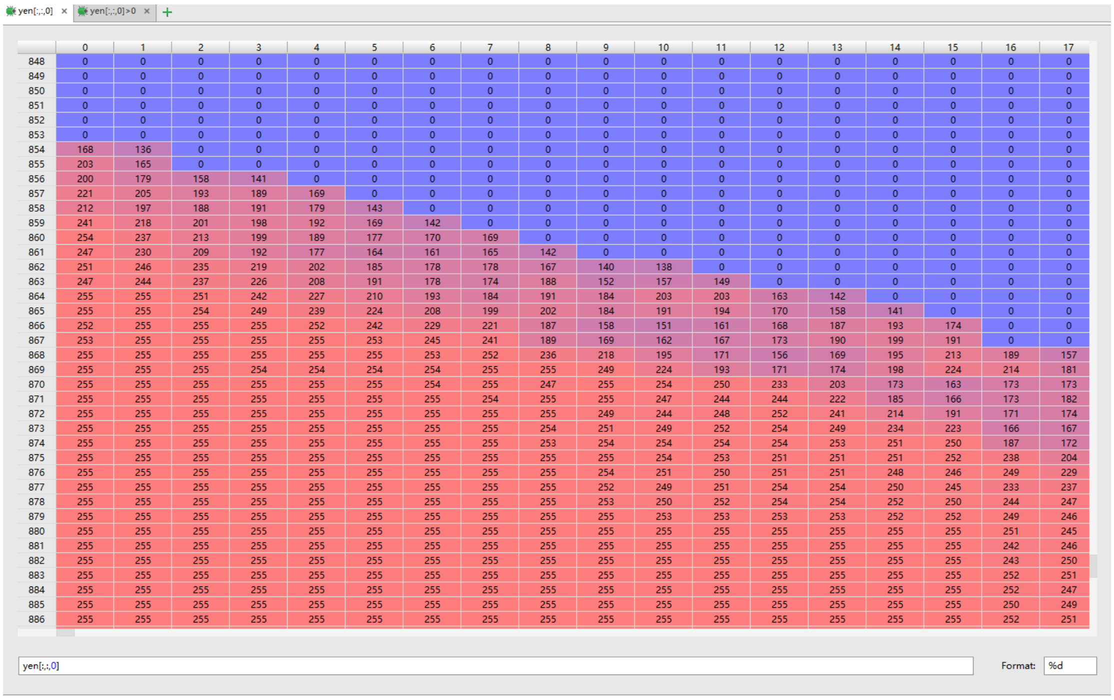
并将小于阈值的像素点值为 0.



	0	1	2
263	0	0	0
264	0	0	0
265	0	0	0
266	0	0	0
267	0	0	0
268	0	0	0
269	0	0	0
270	0	0	0
271	0	0	0
272	0	0	0
273	0	0	0
274	0	0	0
275	0	0	0
276	0	0	0
277	0	0	0
278	0	0	0
279	0	0	0
280	0	0	0
281	255	255	246
282	247	249	238
283	255	255	248
284	255	255	251
285	254	253	255
286	249	248	255
287	245	243	255
288	246	248	247
289	254	255	255
290	252	254	253

3. 图像的降噪

首先求出图片中大于 0 的像素点

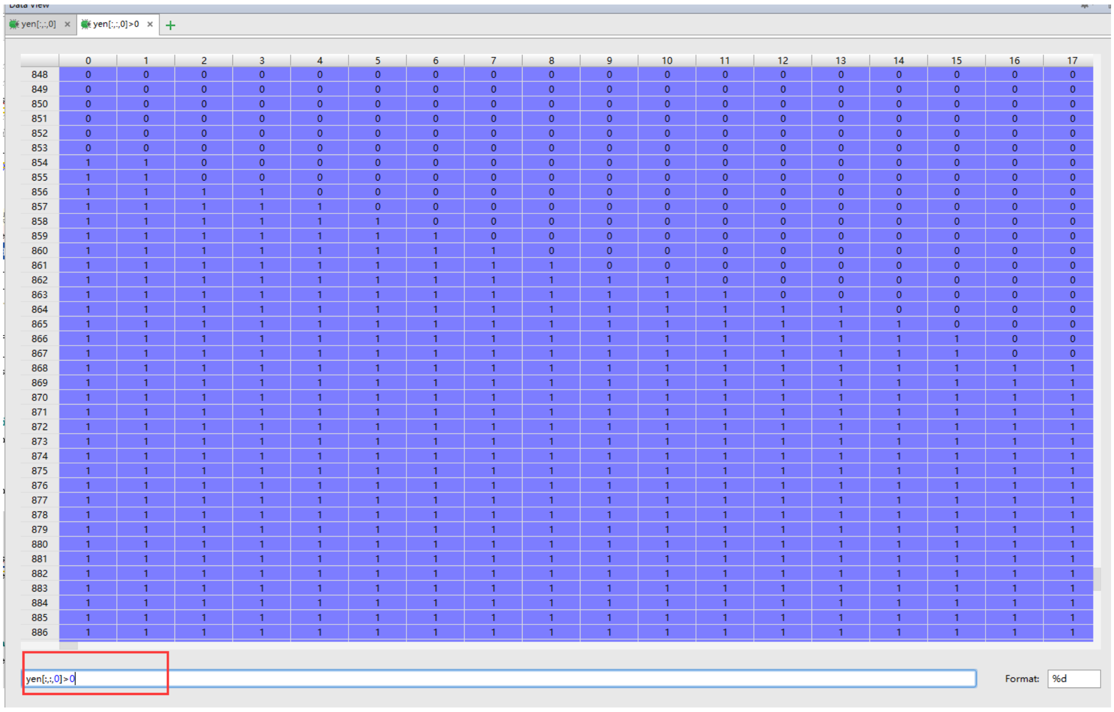


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
848	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
849	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
850	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
851	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
852	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
853	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
854	168	136	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
855	203	165	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
856	200	179	158	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0
857	221	205	193	189	169	0	0	0	0	0	0	0	0	0	0	0	0	0
858	212	197	188	191	179	143	0	0	0	0	0	0	0	0	0	0	0	0
859	241	218	201	198	192	169	142	0	0	0	0	0	0	0	0	0	0	0
860	254	237	213	199	189	177	170	169	0	0	0	0	0	0	0	0	0	0
861	247	230	209	192	177	164	161	165	142	0	0	0	0	0	0	0	0	0
862	251	246	235	219	202	185	178	178	167	140	138	0	0	0	0	0	0	0
863	247	244	237	226	208	191	178	174	188	152	157	149	0	0	0	0	0	0
864	255	255	251	242	227	210	193	184	191	184	203	203	163	142	0	0	0	0
865	255	255	254	249	239	224	208	199	202	184	191	194	170	158	141	0	0	0
866	252	255	255	255	252	242	229	221	187	158	151	161	168	187	193	174	0	0
867	253	255	255	255	255	253	245	241	189	169	162	167	173	190	199	191	0	0
868	255	255	255	255	255	255	253	252	236	218	195	171	156	169	195	213	189	157
869	255	255	255	254	254	254	254	255	255	249	224	193	171	174	198	224	214	181
870	255	255	255	255	255	255	254	255	247	255	254	250	233	203	173	163	173	173
871	255	255	255	255	255	255	255	254	255	255	247	244	244	222	185	166	173	182
872	255	255	255	255	255	255	255	255	255	249	244	248	252	241	214	191	171	174
873	255	255	255	255	255	255	255	255	254	251	249	252	254	249	234	223	166	167
874	255	255	255	255	255	255	255	255	253	254	254	254	254	253	251	250	187	172
875	255	255	255	255	255	255	255	255	255	255	254	253	251	251	251	252	238	204
876	255	255	255	255	255	255	255	255	255	254	251	250	251	251	248	246	249	229
877	255	255	255	255	255	255	255	255	255	252	249	251	254	254	250	245	233	237
878	255	255	255	255	255	255	255	255	255	253	250	252	254	254	252	250	244	247
879	255	255	255	255	255	255	255	255	255	255	253	253	253	253	252	252	249	246
880	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	251	245
881	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	242	246
882	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	243	250
883	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	252	251
884	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	252	247
885	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	250	249
886	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	252	251

yen[:,:,0]

Format: %d

将他转换为二值图片，像素点要么是 0 或者是 1

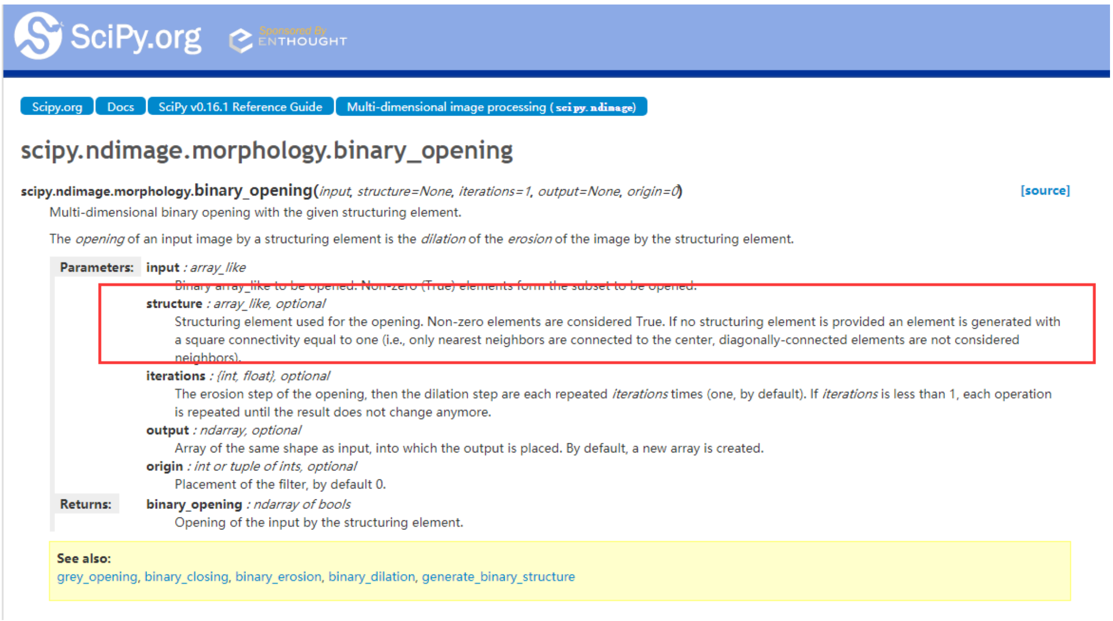


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
848	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
849	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
850	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
851	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
852	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
853	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
854	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
855	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
856	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
857	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
858	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
859	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
860	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
861	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
862	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
863	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
864	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
865	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
866	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
867	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
868	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
869	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
870	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
871	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
872	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
873	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
874	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
875	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
876	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
877	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
878	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
879	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
880	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
881	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
882	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
883	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
884	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
885	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
886	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

yen[:,:,0]>0

Format: %d

```
binary_image = binary_opening(binary_image, structure=structure)
```


SciPy.org
Sponsored By
ENTHOUGHT
Scipy.org
Docs
SciPy v0.16.1 Reference Guide
Multi-dimensional image processing (scipy.ndimage)

scipy.ndimage.morphology.binary_opening

scipy.ndimage.morphology.binary_opening(*input, structure=None, iterations=1, output=None, origin=0*) [source]

Multi-dimensional binary opening with the given structuring element.

The *opening* of an input image by a structuring element is the *dilation* of the *erosion* of the image by the structuring element.

Parameters: **input** : *array_like*
Binary array_like to be opened. Non-zero (True) elements form the subset to be opened.

structure : *array_like, optional*
Structuring element used for the opening. Non-zero elements are considered True. If no structuring element is provided an element is generated with a square connectivity equal to one (i.e., only nearest neighbors are connected to the center, diagonally-connected elements are not considered neighbors).

iterations : *{int, float}, optional*
The erosion step of the opening, then the dilation step are each repeated *iterations* times (one, by default). If *iterations* is less than 1, each operation is repeated until the result does not change anymore.

output : *ndarray, optional*
Array of the same shape as input, into which the output is placed. By default, a new array is created.

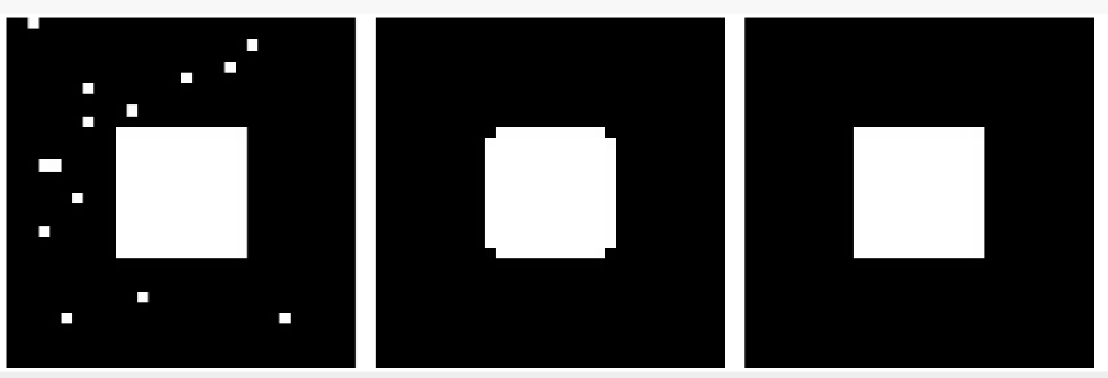
origin : *int or tuple of ints, optional*
Placement of the filter, by default 0.

Returns: **binary_opening** : *ndarray of bools*
Opening of the input by the structuring element.

See also:
grey_opening, binary_closing, binary_erosion, binary_dilation, generate_binary_structure

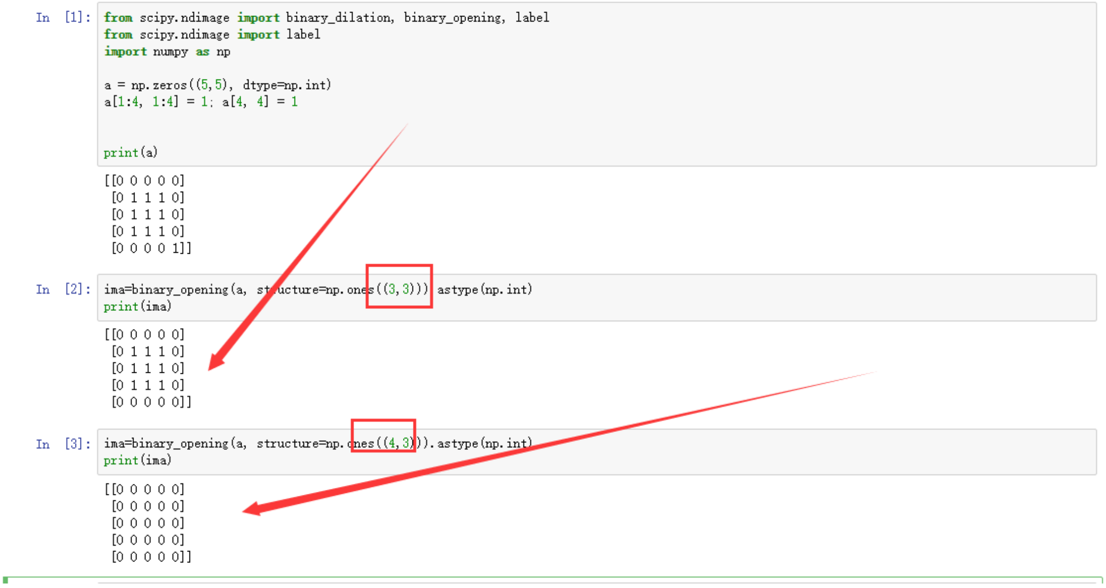
先对二值图片进行：先腐蚀再膨胀操作，此操作可以移除噪声

所谓的腐蚀就是将 0 值扩充到邻近像素。扩大黑色部分，减小白色部分。可用来提取骨干信息，去掉毛刺，去掉孤立的像素。找到像素值为 1 的点，将它的邻近像素点都设置成这个值。1 值表示白，0 值表示黑，因此膨胀操作可以扩大白色值范围，压缩黑色值范围。一般用来扩充边缘或填充小的孔洞。下图就是一个先腐蚀再膨胀的过程：



binary_opening(binary_image, structure=structure)的第二个参数很重要，用于设定局部区域的形状和大小，也就是我们的目标区域的大小，此区域的大小和形状如果设置不当会导致目标无法识别，这也是一大难点。下面通过例子来说明此参数的作用：

首先创建一个矩阵



In [1]:
```
from scipy.ndimage import binary_dilation, binary_opening, label
from scipy.ndimage import label
import numpy as np

a = np.zeros((5,5), dtype=np.int)
a[1:4, 1:4] = 1; a[4, 4] = 1


print(a)
```

```
[[0 0 0 0 0]
 [0 1 1 1 0]
 [0 1 1 1 0]
 [0 1 1 1 0]
 [0 0 0 0 1]]
```

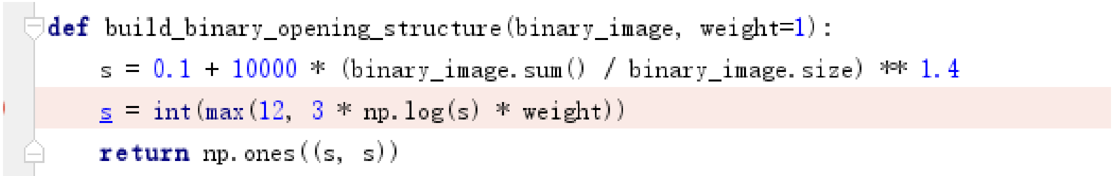
In [2]:
```
ima=binary_opening(a, structure=np.ones((3,3))).astype(np.int)
print(ima)
```

```
[[0 0 0 0 0]
 [0 1 1 1 0]
 [0 1 1 1 0]
 [0 1 1 1 0]
 [0 0 0 0 0]]
```

In [3]:
```
ima=binary_opening(a, structure=np.ones((4,3))).astype(np.int)
print(ima)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

只有目标区域的大小和形状设置的十分妥当的情况下，我们才能准确的识别目标。在这里我们采用的办法是：

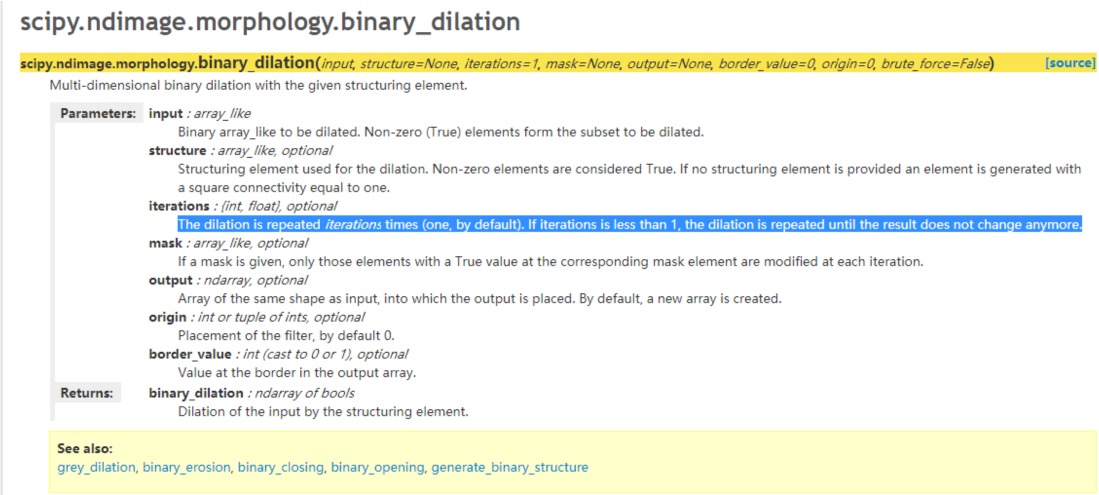


```
def build_binary_opening_structure(binary_image, weight=1):
    s = 0.1 + 10000 * (binary_image.sum() / binary_image.size) ** 1.4
    s = int(max(12, 3 * np.log(s) * weight))
    return np.ones((s, s))
```

通过不断的调参，我们初步认定最佳参数如上所示。

接着对目标再做一次膨胀操作，扩充边缘或填充小的孔洞。

```
binary_image = binary_dilation(binary_image, iterations=dilation_iterations)
```



scipy.ndimage.morphology.binary_dilation

scipy.ndimage.morphology.**binary_dilation**(*input, structure=None, iterations=1, mask=None, output=None, border_value=0, origin=0, brute_force=False*) [source]

Multi-dimensional binary dilation with the given structuring element.

Parameters:

input : *array_like*
Binary array_like to be dilated. Non-zero (True) elements form the subset to be dilated.

structure : *array_like, optional*
Structuring element used for the dilation. Non-zero elements are considered True. If no structuring element is provided an element is generated with a square connectivity equal to one.

iterations : *{int, float}, optional*
The dilation is repeated *iterations* times (one, by default). If iterations is less than 1, the dilation is repeated until the result does not change anymore.

mask : *array_like, optional*
If a mask is given, only those elements with a True value at the corresponding mask element are modified at each iteration.

output : *ndarray, optional*
Array of the same shape as input, into which the output is placed. By default, a new array is created.

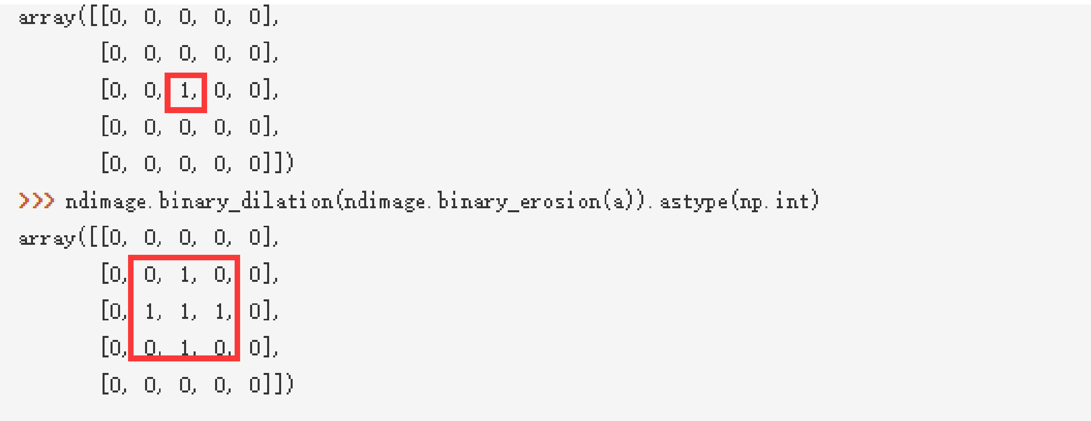
origin : *int or tuple of ints, optional*
Placement of the filter, by default 0.

border_value : *int (cast to 0 or 1), optional*
Value at the border in the output array.

Returns:

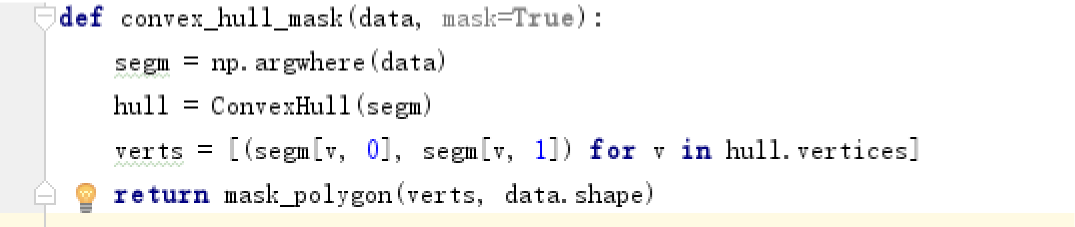
binary_dilation : *ndarray of bools*
Dilation of the input by the structuring element.

See also:
grey_dilation, binary_erosion, binary_closing, binary_opening, generate_binary_structure



```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
>>> ndimage.binary_dilation(ndimage.binary_erosion(a)).astype(np.int)
array([[0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0]])
```

接下来将目标区域的轮廓给标注出来。



```
def convex_hull_mask(data, mask=True):
    segm = np.argwhere(data)
    hull = ConvexHull(segm)
    verts = [(segm[v, 0], segm[v, 1]) for v in hull.vertices]
    return mask_polygon(verts, data.shape)
```