# 图像的阈值处理

电子信息工程系

袁羽

# 目录

CONTENTS

1 阈值处理函数

2 “非黑即白”的图像

3 零处理

4 截断处理

5 自适应处理

6 Otsu方法

7 阈值处理的作用

# 图像的阈值处理

阈值是图像处理中一个很重要的概念，类似一个“像素值的标准线”。所有像素值都与这条“标准线”进行比较，最后得到3种结果：像素值比阈值大、像素值比阈值小或像素值等于阈值。程序根据这些结果将所有像素进行分组，然后对某一组像素进行“加深”或“变淡”操作，使得整个图像的轮廓更加鲜明，更容易被计算机或肉眼识别。

在图像处理的过程中，阈值的使用使得图像的像素值更单一，进而使得图像的效果更简单。首先，把一幅彩色图像转换为灰度图像，这样图像的像素值的取值范围即可简化为0~255。然后，通过阈值使得转换后的灰度图像呈现出只有纯黑色和纯白色的视觉效果。

例如，当阈值为127时，把小于127的所有像素值都转换为0（即纯黑色），把大于127的所有像素值都转换为255（即纯白色）。虽然会丢失一些灰度细节，但是会更明显地保留灰度图像主体的轮廓。

# 一 阈值处理函数

OpenCV提供的threshold()方法用于对图像进行阈值处理，threshold()方法的语法如下：

```
retval, dst = cv2.threshold(src, thresh, maxval, type)
```

参数说明：

src：被处理的图像，可以是多通道图像。

thresh：阈值，阈值在125～150取值的效果最好。

maxval：阈值处理采用的最大值。

type：阈值处理类型。

# 一 阈值处理函数

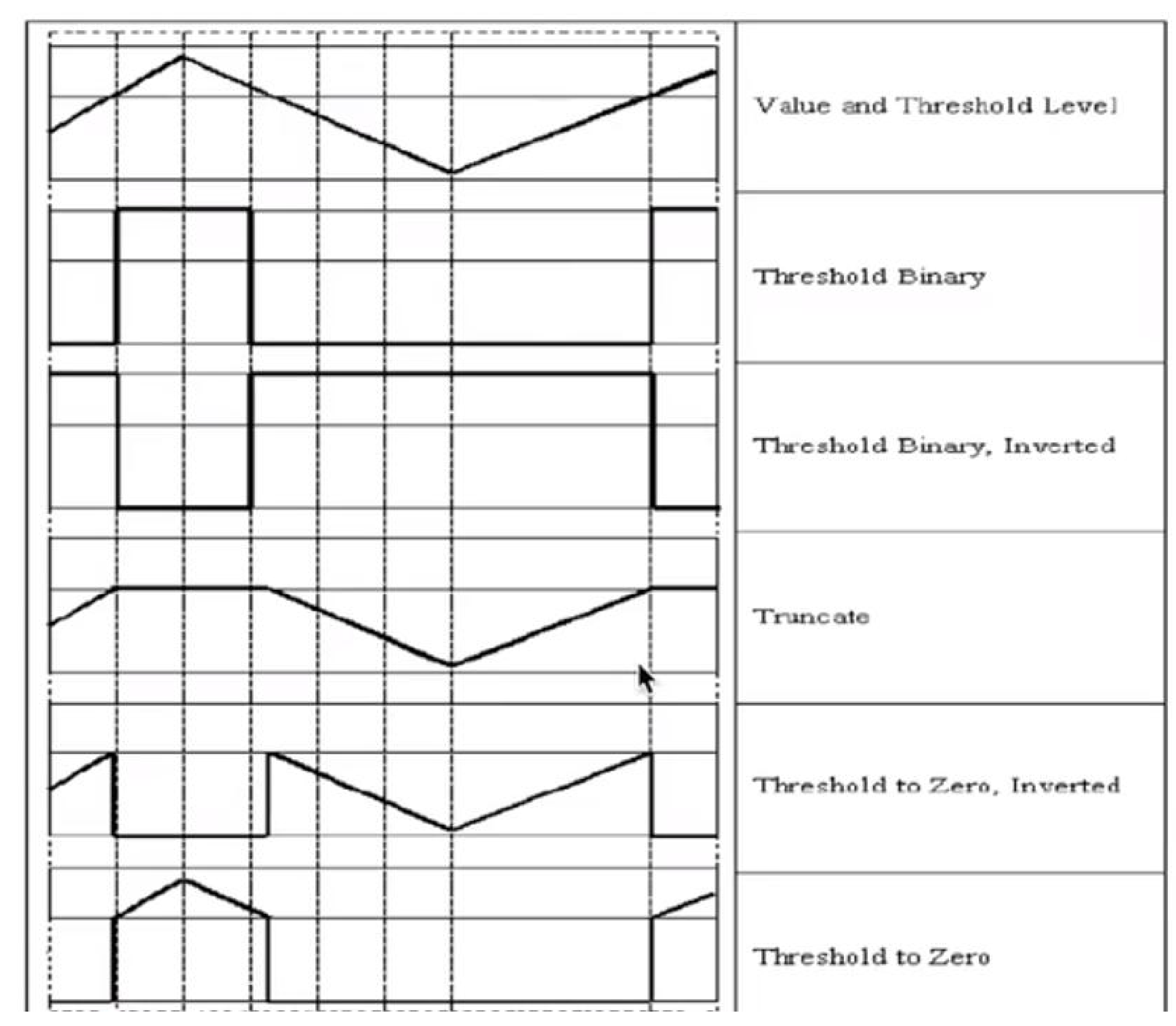
type：阈值处理类型。

| 阈值处理类型 | 含义 |
| --- | --- |
| cv2.THRESH_BINARY | 阈值二值化：大于阈值的部分像素值变为最大值，其他变为0 |
| cv2.THRESH_BINARY_INV | 阈值反二值化：大于阈值的部分变为0，其他部分变为最大值 |
| cv2.THRESH_TOZERO | 阈值取零：大于阈值的部分不变，其余部分变为0 |
| cv2.THRESH_TOZERO_INV | 阈值反取零:大于阈值的部分变为0，其余部分不变 |
| cv2.THRESH_TRUNC | 截断阈值化 :大于阈值的部分变为阈值，其余部分不变 |

# 一 阈值处理函数

thresholdType

type：阈值处理类型。



# 一 阈值处理函数

OpenCV提供的threshold()方法用于对图像进行阈值处理，threshold()方法的语法如下：

```
retval, dst = cv2.threshold(src, thresh, maxval, type)
```

返回值说明：

retval：处理时采用的阈值。

dst：经过阈值处理后的图像。

# 二 “非黑即白”的图像

二值化处理和反二值化处理使得灰度图像的像素值两极分化，灰度图像呈现出只有纯黑色和纯白色的视觉效果。

1.二值化处理（cv2.THRESH_BINARY）：大于阈值的部分像素值变为最大值，其他变为0 。

2.反二值化处理（cv2.THRESH_BINARY_INV）：大于阈值的部分变为0，其他部分变为最大值。

# 二 “非黑即白”的图像

1、二值化处理（cv2.THRESH_BINARY）

二值化处理也叫二值化阈值处理，该处理让图像仅保留两种像素值，或者说所有像素都只能从两种值中取值。进行二值化处理时，每一个像素值都会与阈值进行比较，将大于阈值的像素值变为最大值，将小于或等于阈值的像素值变为0，计算公式如下：

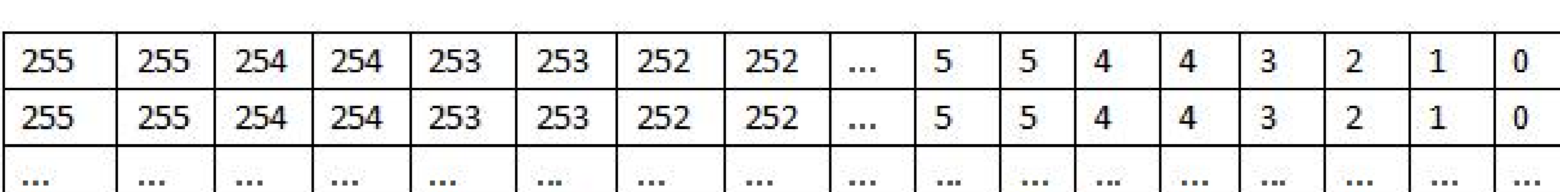
if 像素值 <= 阈值: 像素值 = 0

if 像素值 > 阈值: 像素值 = 最大值

通常二值化处理是使用255作为最大值，因为灰度图像中255表示纯白色，能够很清晰地与纯黑色进行区分，所以灰度图像经过二值化处理后呈现“非黑即白”的效果。

# 二 “非黑即白”的图像

1、二值化处理



| 255 | 255 | 254 | 254 | 253 | 253 | 252 | 252 | ... | 5 | 5 | 4 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 254 | 254 | 253 | 253 | 252 | 252 | ... | 5 | 5 | 4 | 4 | 3 | 2 | 1 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

# 例 二值化处理白黑渐变图：取0~255的中间值127作为阈值，将255作为最大值。

大于阈值的像素值变为最大值，将小于或等于阈值的像素值变为0。

# 例 二值化处理白黑渐变图：取0~255的中间值127作为阈值，将255作为最大值。

```
import cv2

img = cv2.imread("black.png", 0)  # 将图像读成灰度图像

#阈值处理函数：retval, dst = cv2.threshold(src, thresh, maxval, type)

t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)  # 二值化阈值处理

cv2.imshow('img', img)  # 显示原图

cv2.imshow('dst1', dst1)  # 二值化阈值处理效果图

cv2.waitKey()  # 按下任何键盘按键后

cv2.destroyAllWindows()  # 释放所有窗体
```

# 例 二值化处理白黑渐变图：观察不同阈值的处理效果。

```
import cv2
img = cv2.imread("black.png", 0)  # 将图像读成灰度图像
#阈值处理函数：retval, dst = cv2.threshold(src, thresh, maxval, type)
t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)  # 二值化阈值处理
t2, dst2 = cv2.threshold(img, 210, 255, cv2.THRESH_BINARY)  # 调高阈值效果
cv2.imshow('dst1', dst1)  # 展示阈值为127时的效果
cv2.imshow('dst2', dst2)  # 展示阈值为210时的效果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

# 例 二值化处理白黑渐变图：观察不同阈值的处理效果。

通过修改阈值大小可以调整黑白交界的位置。例如，分别采用127和210作为阈值，对比处理结果

# 例 二值化处理白黑渐变图：观察不同最大值的处理效果。

```
import cv2
img = cv2.imread("black.png", 0)  # 将图像读成灰度图像
#阈值处理函数：retval, dst = cv2.threshold(src, thresh, maxval, type)
t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)  # 二值化阈值处理
t3, dst3 = cv2.threshold(img, 127, 150, cv2.THRESH_BINARY)  # 调低最大值效果
cv2.imshow('dst1', dst1)  # 展示最大值为255时的效果
cv2.imshow('dst3', dst3)  # 展示最大值为15时的效果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

# 例 二值化处理白黑渐变图：观察不同最大值的处理效果。

像素值的最小值默认为0，但最大值可以由开发者设定。如果最大值不是255（纯白色），那么“非黑”的像素就不一定是纯白色了。例如，灰度值150表现为“灰色”。

# 二 “非黑即白”的图像

2、反二值化处理（cv2.THRESH_BINARY_INV)

反二值化处理也叫反二值化阈值处理，其结果为二值化处理的相反结果。将大于阈值的像素值变为0，将小于或等于阈值的像素值变为最大值。原图像中白色的部分变成黑色，黑色的部分变成白色。计算公式如下：

if 像素值 <= 阈值: 像素值 = 最大值

if 像素值 > 阈值: 像素值 = 0

# 例 对图像进行反二值化处理。

```
import cv2
img = cv2.imread("black.png", 0) # 将图像读成灰度图像
t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY) # 二值化阈值处理
t4, dst4 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV) # 反二值化阈值处理
cv2.imshow('dst1', dst1) # 展示二值化效果
cv2.imshow('dst4', dst4) # 展示反二值化效果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

# 例 对图像进行反二值化处理。

# 例 彩色图像进行二值化处理和反二值化处理。

```
import cv2
img = cv2.imread("lenna.jpg") # 读取图像
t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY) # 二值化阈值处理
t2, dst2 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV) # 反二值化阈值处理
cv2.imshow('img', img) # 展示原图像
cv2.imshow('dst1', dst1) # 展示二值化效果
cv2.imshow('dst2', dst2) # 展示反二值化效果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

# 三 零处理

零处理会将某一个范围内的像素值变为0，并允许范围之外的像素保留原值。零处理包括低于阈值零处理（cv2.THRESH_TOZERO）和超出阈值零处理（cv2.THRESH_TOZERO_INV）。

1.低于阈值零处理也叫低阈值零处理，该处理将低于或等于阈值的像素值变为0，大于阈值的像素值保持原值，计算公式如下：

```
if 像素值 <= 阈值: 像素值 = 0
if 像素值 > 阈值: 像素值 = 原值
```

2.超出阈值零处理也叫超阈值零处理，该处理将大于阈值的像素值变为0，小于或等于阈值的像素值保持原值。计算公式如下：

```
if 像素值 <= 阈值: 像素值 = 原值
if 像素值 > 阈值: 像素值 = 0
```

# 例 对图像进行低于阈值零处理。

低于或等于阈值的像素值变为0，大于阈值的像素值保持原值。

# 例 对图像进行低于阈值零处理。

```
import cv2
img = cv2.imread("black.png", 0)  # 将图像读成灰度图像
t5, dst5 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)  # 低于阈值零处理
cv2.imshow('img', img)  # 显示原图
cv2.imshow('dst5', dst5)  # 低于阈值零处理效果图
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

# 例 对图像进行超出阈值零处理。

大于阈值的像素值变为0，小于或等于阈值的像素值保持原值。

# 例 对图像进行超出阈值零处理。

```
import cv2
img = cv2.imread("black.png", 0) # 将图像读成灰度图像
t6, dst6 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV) # 超出阈值零处理
cv2.imshow('img', img) # 显示原图
cv2.imshow('dst6', dst6) # 超出阈值零处理效果图
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

# 例 对彩色图像进行低于阈值零处理和超出阈值零处理。

```
import cv2
img = cv2.imread("lenna.jpg")  # 将图像读成灰度图像
t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)  # 低于阈值零处理
t2, dst2 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV)  # 超出阈值零处理
cv2.imshow('img', img)  # 显示原图
cv2.imshow('dst1', dst1)  # 低于阈值零处理效果图
cv2.imshow('dst2', dst2)  # 超出阈值零处理效果图
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

# 四 截断处理

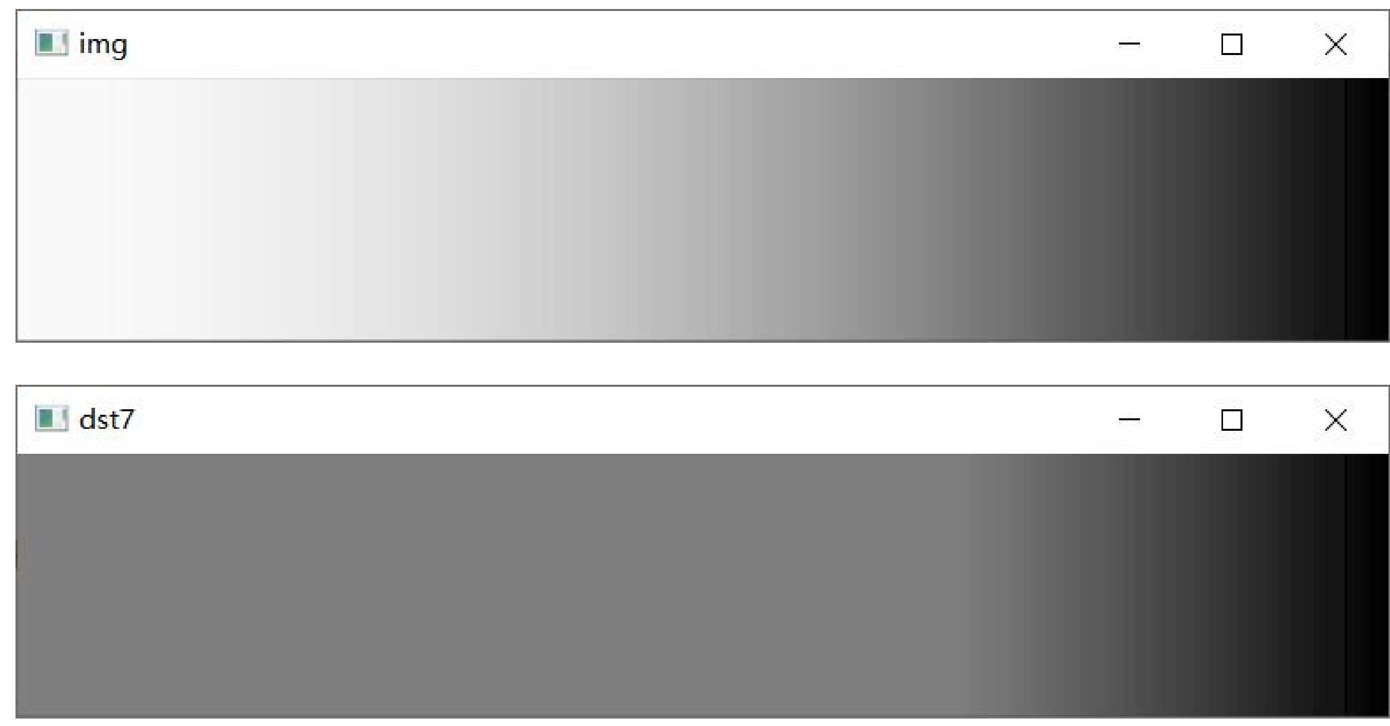
　　截断处理也叫截断阈值处理（cv2.THRESH_TRUNC），该处理将图像中大于阈值的像素值变为和阈值一样的值，小于或等于阈值的像素保持原值，其公式如下:

if 像素 <= 阈值: 像素 = 原值

if 像素 > 阈值: 像素 = 阈值

# 例 对图像进行截断处理。

大于阈值的像素值变为和阈值一样的值，小于或等于阈值的像素保持原值。



# 例 对图像进行截断处理。

```
import cv2
img = cv2.imread("black.png", 0)  # 将图像读成灰度图像
t6, dst6 = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)  # 截断处理
cv2.imshow('img', img)  # 显示原图
cv2.imshow('dst6', dst6)  # 超出阈值零处理效果图
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

# 例 对彩色图像进行截断处理。

```
import cv2
img = cv2.imread("lenna.jpg")  # 读取图像
t1, dst1 = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)  # 截断处理
cv2.imshow('img', img)  # 展示原图
cv2.imshow('dst', dst1)  # 展示截断效果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

图像经过截断处理后，整体颜色都会变暗。彩色图像经过截断处理后，在降低亮度的同时还会让浅颜色区域的颜色变得更浅

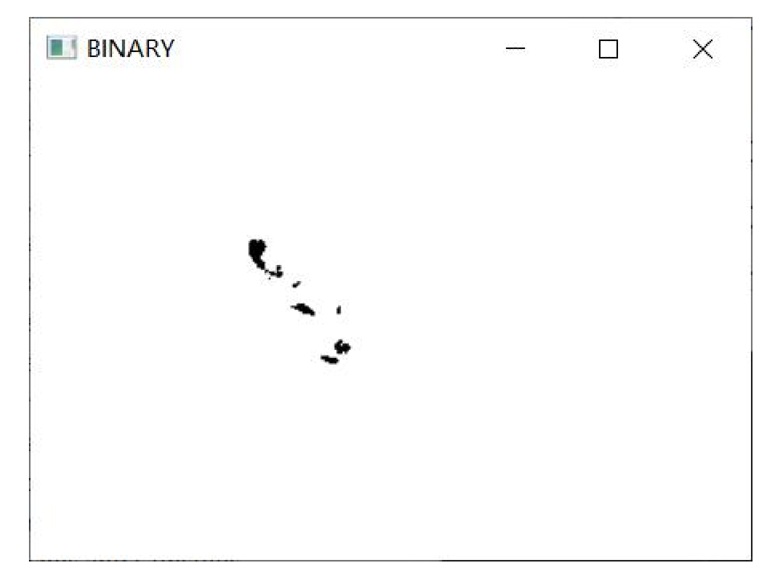
# 五 自适应处理

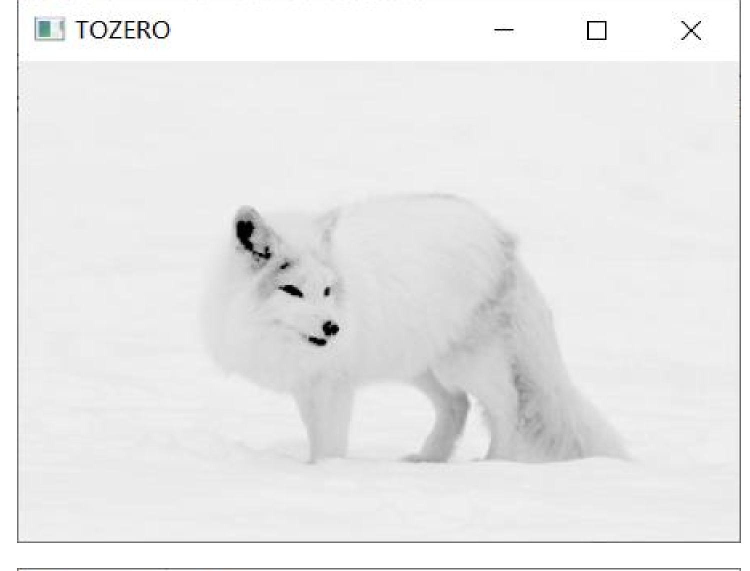
前面已经依次对cv2.THRESH_BINARY、cv2.THRESH_BINARY_INV、cv2.THRESH_TOZERO、cv2.THRESH_TOZERO_INV和cv2.THRESH_TRUNC这5种阈值处理类型进行了详解。如果是一幅色彩均衡的图像，所以直接使用一种阈值处理类型就能够对图像进行阈值处理。很多时候图像的色彩是不均衡的，如果只使用一种阈值处理类型，就无法得到清晰有效的结果。

# 例 使用常用的5种阈值处理类型对色彩不均衡的图像进行处理。

```
import cv2
image = cv2.imread("bj.png") # 读取bj.png
image_Gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # 将bj.png转换为灰度图像
t1, dst1 = cv2.threshold(image_Gray, 127, 255, cv2.THRESH_BINARY) # 二值化阈值处理
t2, dst2 = cv2.threshold(image_Gray, 127, 255, cv2.THRESH_BINARY_INV) # 反二值化阈值处理
t3, dst3 = cv2.threshold(image_Gray, 127, 255, cv2.THRESH_TOZERO) # 低于阈值零处理
t4, dst4 = cv2.threshold(image_Gray, 127, 255, cv2.THRESH_TOZERO_INV) # 超出阈值零处理
t5, dst5 = cv2.threshold(image_Gray, 127, 255, cv2.THRESH_TRUNC) # 截断处理
# 分别显示经过5种阈值类型处理后的图像
cv2.imshow("BINARY", dst1)
cv2.imshow("BINARY_INV", dst2)
cv2.imshow("TOZERO", dst3)
cv2.imshow("TOZERO_INV", dst4)
cv2.imshow("TRUNC", dst5)
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 销毁所有窗口
```

# 例 使用常用的5种阈值处理类型对色彩不均衡的图像进行处理。



# 五 自适应处理

OpenCV提供了一种改进的阈值处理技术：图像中的不同区域使用不同的阈值。把这种改进的阈值处理技术称作自适应阈值处理也称自适应处理，自适应阈值是根据图像中某一正方形区域内的所有像素值按照指定的算法计算得到的。与前面讲解的5种阈值处理类型相比，自适应处理能更好地处理明暗分布不均的图像，获得更简单的图像效果。

OpenCV提供了adaptiveThresHold()方法对图像进行自适应处理，adaptiveThresHold()方法的语法如下：

```
dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)
```

# 五 自适应处理

dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

参数说明：

src：被处理的图像。需要注意的是，该图像需是灰度图像。

maxValue：阈值处理采用的最大值。

adaptiveMethod：自适应阈值的计算方法。

| 类型 | 含义 |
| --- | --- |
| ADAPTIVE_THRESH_MEAN_C | 为局部邻域块的像素平均值，该算法是先求出块中的均值，再减去参数 C。 |
| ADAPTIVE_THRESH_GAUSSIAN_C | 为局部邻域块的像素高斯加权和。该算法是在区域中（x,y）周围的像素根据高斯函数按照他们离中心点的距离进行加权计算，再减去参数 C |

# 五 自适应处理

```
dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)
```

参数说明:

src：被处理的图像。需要注意的是，该图像需是灰度图像。

maxValue：阈值处理采用的最大值。

adaptiveMethod：自适应阈值的计算方法。

thresholdType：阈值处理类型；需要注意的是，阈值处理类型需是cv2.THRESH_BINARY或cv2.THRESH_BINARY_INV中的一个。

blockSize：一个正方形区域的大小。例如，5指的是5×5的区域。

C：常量。阈值等于均值或者加权值减去这个常量。

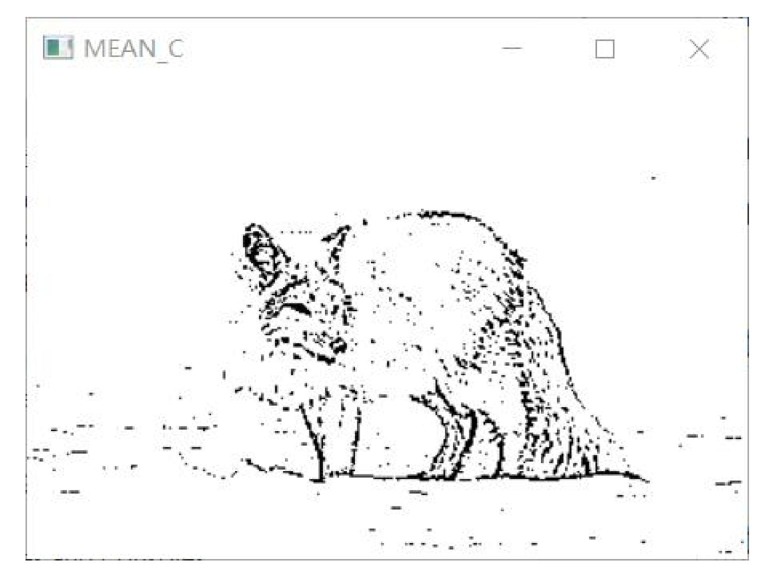
返回值说明:

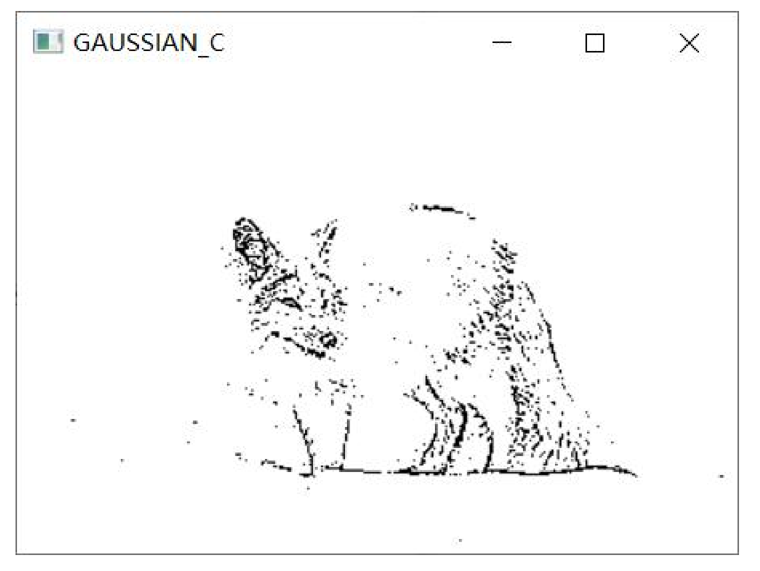
dst：经过阈值处理后的图像。

**例 使用自适应处理：将图像转换为灰度图像，再分别使用cv2.ADAPTIVE_THRESH_MEAN_C和cv2.ADAPTIVE_THRESH_GAUSSIAN_C这两种自适应阈值的计算方法对转换后的灰度图像进行阈值处理。**

```
import cv2
image = cv2.imread("bj.png") # 读取bj.png
image_Gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # 将bj.png转换为灰度图像
# 自适应阈值的计算方法为cv2.ADAPTIVE_THRESH_MEAN_C
athdMEAM = cv2.adaptiveThreshold\
    (image_Gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 5, 3)
# 自适应阈值的计算方法为cv2.ADAPTIVE_THRESH_GAUSSIAN_C
athdGAUS = cv2.adaptiveThreshold\
    (image_Gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY, 5, 3)
# 显示自适应阈值处理的结果
cv2.imshow("MEAN_C", athdMEAM)
cv2.imshow("GAUSSIAN_C", athdGAUS)
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 销毁所有窗口
```

**例 使用自适应处理：将图像转换为灰度图像，再分别使用cv2.ADAPTIVE_THRESH_MEAN_C和cv2.ADAPTIVE_THRESH_GAUSSIAN_C这两种自适应阈值的计算方法对转换后的灰度图像进行阈值处理。**




与前面讲解的5种阈值处理类型的处理结果相比，自适应处理保留了图像中更多的细节信息，更明显地保留了灰度图像主体的轮廓。

注意：使用自适应阈值处理图像时，如果图像是彩色图像，那么需要先将彩色图像转换为灰度图像。

# 六 Otsu方法

前面5种阈值处理类型的过程中，每个实例设置的阈值都是127，这个127是任意设置的，并不是通过算法计算得到的。对于有些图像，当阈值被设置为127时，得到的效果并不好，这时就需要一个个去尝试，直到找到最合适的阈值。而逐个寻找最合适的阈值不仅工作量大，而且效率低。

为此，OpenCV提供了Otsu方法。Otsu方法能够遍历所有可能的阈值，从中找到最合适的阈值。

Otsu方法的语法与threshold()方法的语法基本一致，只不过在为type传递参数时，要多传递一个参数，即cv2.THRESH_OTSU。cv2.THRESH_OTSU的作用就是实现Otsu方法的阈值处理。

# 六 Otsu方法

Otsu方法（最大类间方差法）的语法：retval, dst = cv2.threshold(src, thresh, maxval, type)

参数说明：

src：被处理的图像。需要注意的是，该图像需是灰度图像。

thresh：阈值，且要把阈值设置为0。

maxval：阈值处理采用的最大值，即255。

type：阈值处理类型。除选择一种阈值处理类型外，还要多传递一个参数，即cv2.THRESH_OTSU。例如，cv2.THRESH_BINARY+cv2.THRESH_OTSU。

返回值说明：

retval：由Otsu方法计算得到并使用的最合适的阈值。

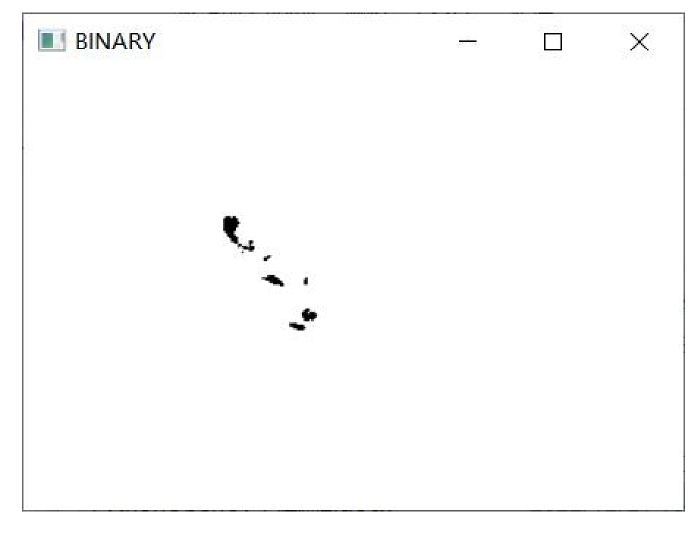
dst：经过阈值处理后的图像。

**例 实现Otsu方法的阈值处理：分别对这幅图像进行二值化处理和实现Otsu方法的阈值处理，对比处理后图像的差异。**

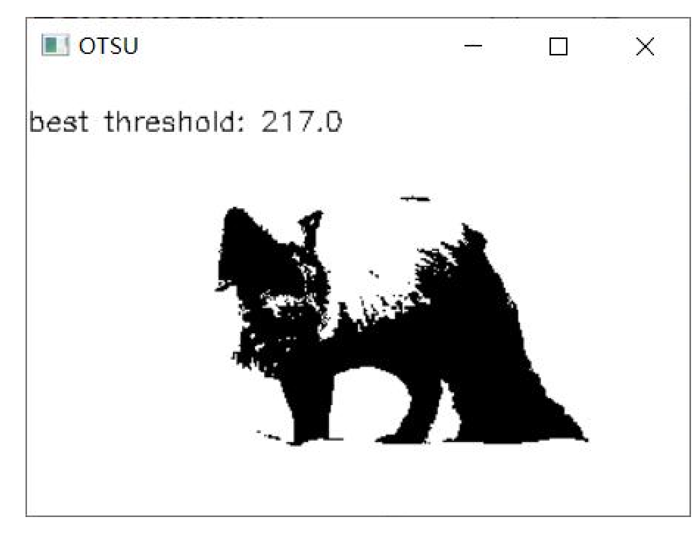
```
import cv2
image = cv2.imread("bj.png") # 读取bj.png
image_Gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # 转换为灰度图像
t1, dst1 = cv2.threshold(image_Gray, 127, 255, cv2.THRESH_BINARY) # 二值化阈值处理
# 实现Otsu方法的阈值处理
t2, dst2 = cv2.threshold(image_Gray, 0, 255, cv2.THRESH_BINARY  + cv2.THRESH_OTSU)
cv2.putText(dst2, "best threshold: " + str(t2), (0, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1) # 在图像上绘制最合适的阈值
cv2.imshow("BINARY", dst1) # 显示二值化阈值处理的图像
cv2.imshow("OTSU", dst2) # 显示实现Otsu方法的阈值处理
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 销毁所有窗口
```

**例 实现Otsu方法的阈值处理：分别对这幅图像进行二值化处理和实现Otsu方法的阈值处理，对比处理后图像的差异。**

原图




二值化处理




Otsu方法的阈值处理

对比后能够发现，由于图像的亮度较高，使用阈值为127进行二值化阈值处理的结果没有很好地保留图像主体的轮廓，并出现了大量的白色区域。但是，通过实现Otsu方法的阈值处理，不仅找到了最合适的阈值，还将图像主体的轮廓很好地保留了下来，获得了比较好的处理结果。

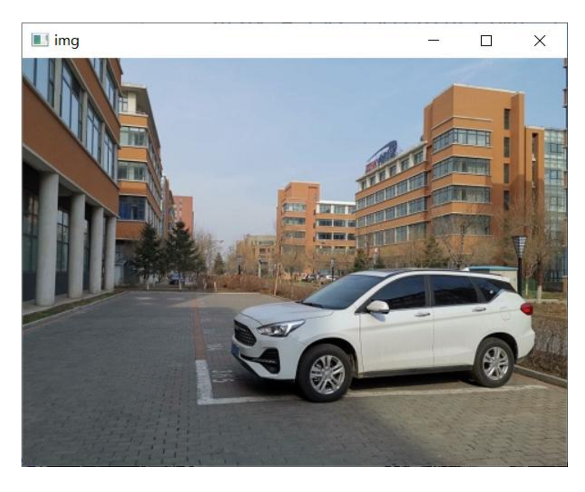
# 七 阈值处理的作用

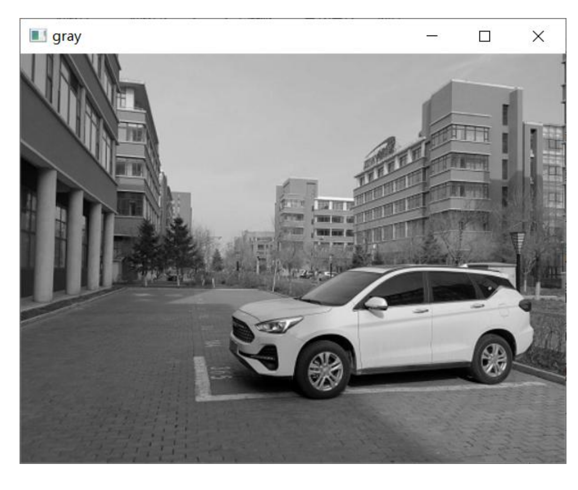
阈值处理在计算机视觉技术中占有十分重要的位置，它是很多高级算法的底层处理逻辑之一。因为二值图像会忽略细节，放大特征，而很多高级算法要根据物体的轮廓来分析物体特征，所以二值图像非常适合做复杂的识别运算。在进行识别运算之前，应先将图像转为灰度图像，再进行二值化处理，这样就得到了算法所需要的物体（大致）轮廓图像。

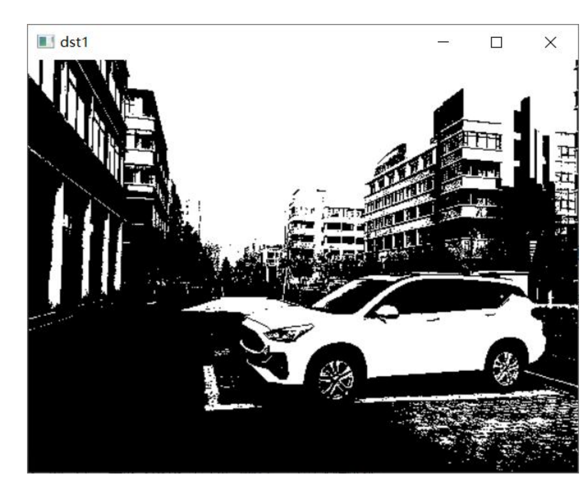
# 例 利用阈值处理勾勒楼房和汽车的轮廓：先将图像转为灰度图像，再将图像分别进行二值化处理和反二值化处理

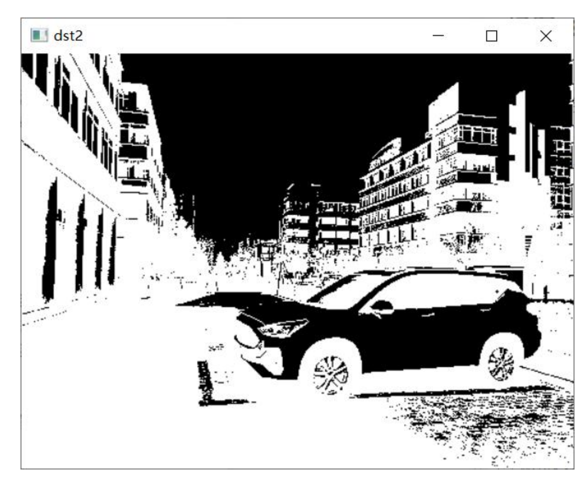
```
import cv2
img = cv2.imread("car.jpg")  # 原始图像
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 转为灰度图像
t1, dst1 = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)  # 二值化阈值处理
t2, dst2 = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)  # 反二值化阈值处理
cv2.imshow("img", img)  # 显示图像
cv2.imshow("gray", gray)
cv2.imshow("dst1", dst1)
cv2.imshow("dst2", dst2)
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

## 例 利用阈值处理勾勒楼房和汽车的轮廓：先将图像转为灰度图像，再将图像分别进行二值化处理和反二值化处理



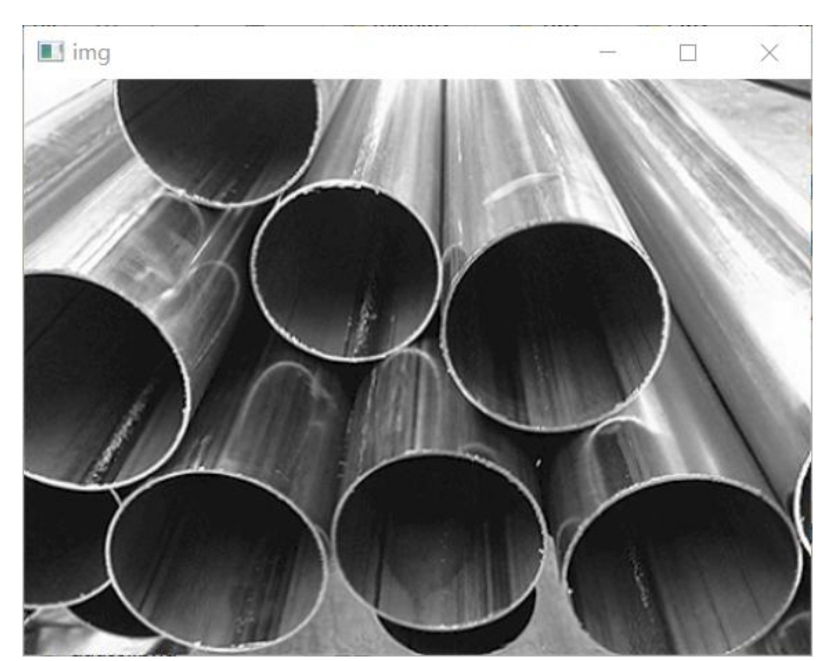



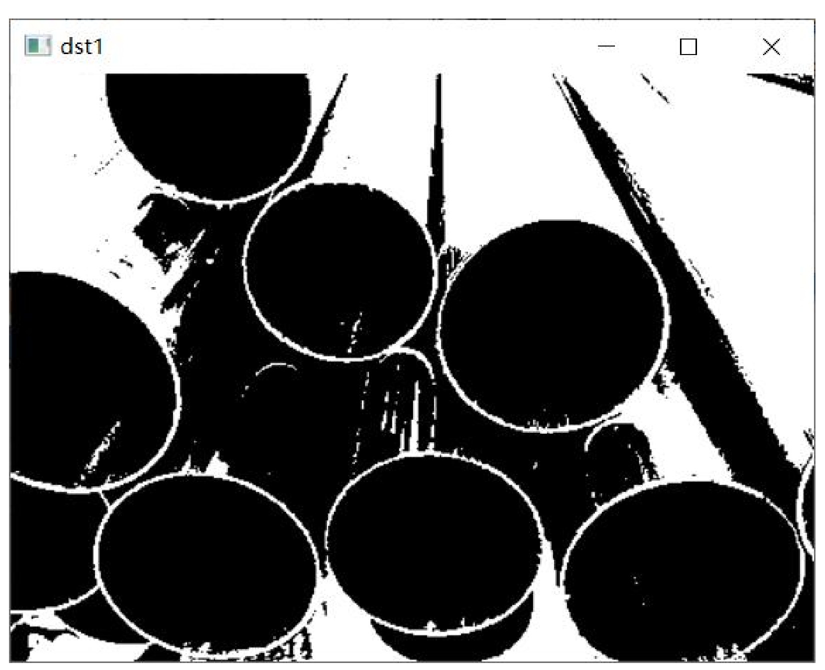



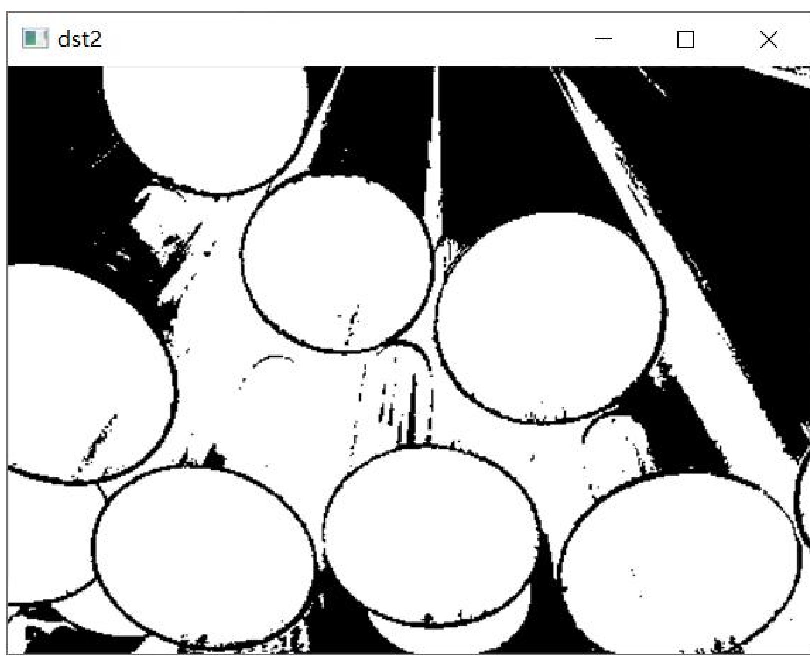




从后面两幅图像可以看到，二值化处理后，图片只有纯黑和纯白两种颜色，图像中的楼房边缘变得更加鲜明，更容易被识别。地面因为颜色较深，所以大面积被涂黑，这样白色的汽车就与地面形成了鲜明的反差。二值化处理后的汽车轮廓在肉眼看来可能还不够明显，但反二值化处理后的汽车轮廓与地面的反差就非常大。**高级图像识别算法可以根据这种鲜明的像素变化来搜寻特征，最后达到识别物体分类的目的。**

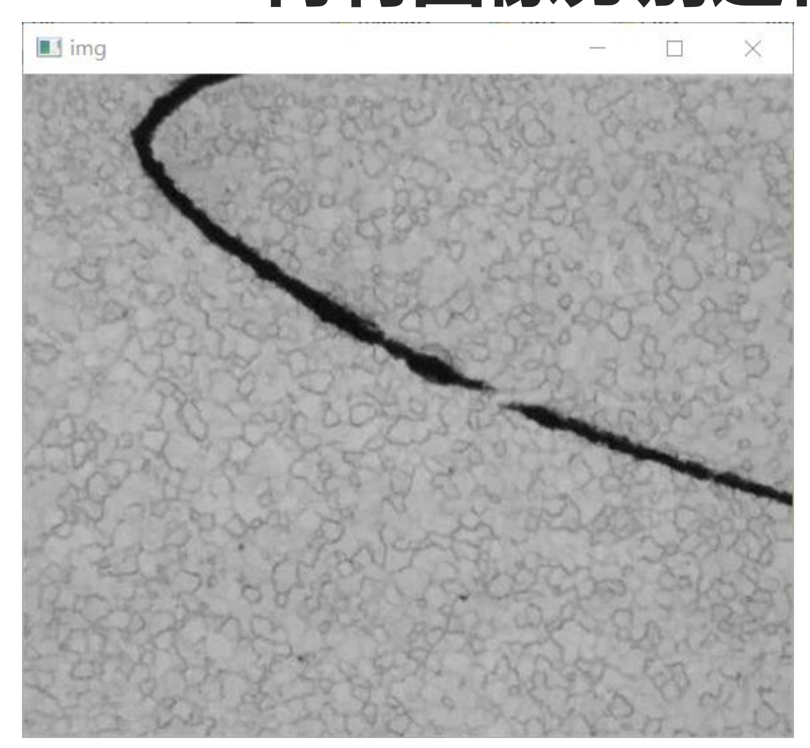
## 例 利用阈值处理勾勒钢管轮廓：先将图像转为灰度图像，再将图像分别进行二值化处理和反二值化处理

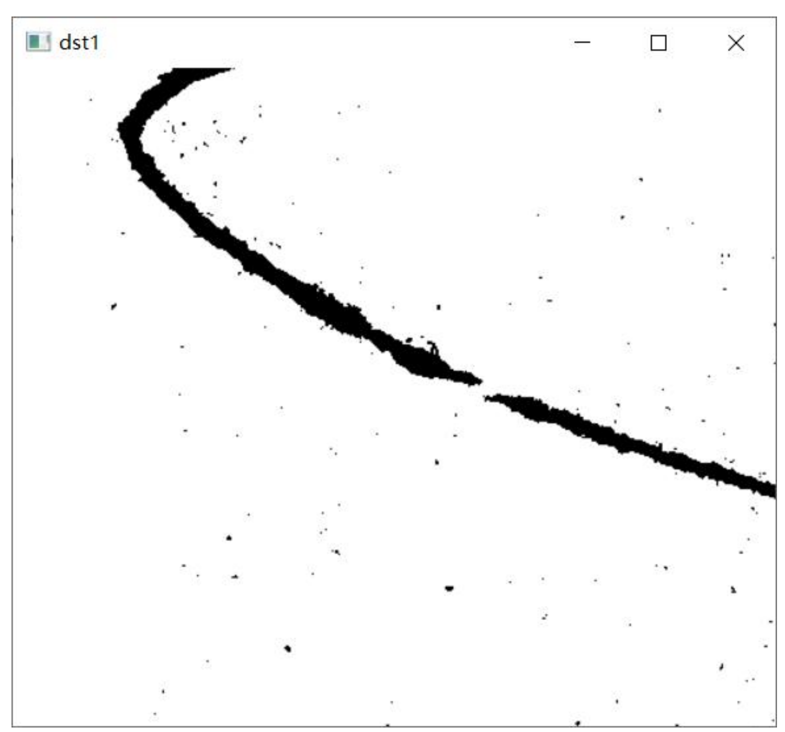


# 例 利用阈值处理勾勒缺陷产品轮廓：先将图像转为灰度图像，再将图像分别进行二值化处理和反二值化处理



# 小结

1.二值化处理

2.零处理

3.截断处理

4.自适应处理

5.Otsu方法

# THANKS