7 阈值化技术

图像分割是许多计算机视觉应用的关键环节。它通常用于将图像分割成不同的区域。理想情况下，这些区域对应于从背景中提取的真实对象。因此，图像分割是图像识别和内容分析的重要步骤。图像阈值法是一种简单而有效的图像分割方法，它根据像素的强度值对像素进行分割，可以将图像分割为前景和背景。

本章将学习OpenCV及以及sci -image图像处理库提供的阈值技术，作为图像分割的关键，了解阈值技术在计算机视觉项目中的重要性。

本章主要内容如下

引入阈值技术
简单的阈值技术
自适应阈值技术
大津阈值算法
三角形的阈值算法
阈值彩色图像
阈值彩色图像

7.1 技术要求

技术要求如下

Python和OpenCV
python IDE。
NumPy和Matplotlib包
scikit-image图像处理库(本章最后一节可选)。参见使用sci -image部分的阈值算法，以了解如何为基于conda的发行版安装它)。要使用pip安装它，请参阅以下说明。
SciPy库(本章最后一节可选)也是必需的。要使用pip安装它，请参阅以下说明。
关于如何安装这些需求的更多细节将在第一章设置OpenCV中介绍。

7.2 安装scikit-image

请使用以下命令安装scikit-image

```
pip install scikit-image
```

7.3 安装SciPy

使用以下命令

```
pip install scipy
```

7.4 引入阈值技术

阈值法是一种简单有效的图像前景和背景分割方法。图像分割的目的是将图像的表示修改为另一种更容易处理的表示。例如，图像分割通常用于根据对象的某些属性(例如颜色、边缘或直方图)从背景中提取对象。最简单的阈值方法是，如果像素强度小于某个预定义的常数(阈值)，则使用黑色像素替换源图像中的每个像素;如果像素强度大于阈值，则使用白色像素替换源图像中的每个像素。

OpenCV为阈值图像提供了cv2.threshold()函数。本章下一小节中更详细地了解这个函数。

在introduction.py阈值脚本中，用了cv2.threshold()函数，其中包含预定义的阈值0、50、100、150、200和250，以查看不同阈值图像是如何变化的。

例如，对阈值为thresh = 50的进行图像阈值处理，代码如下

```
ret1, thresh1 = cv2.threshold(gray_image, 50, 255, cv2.THRESH_BINARY)
```

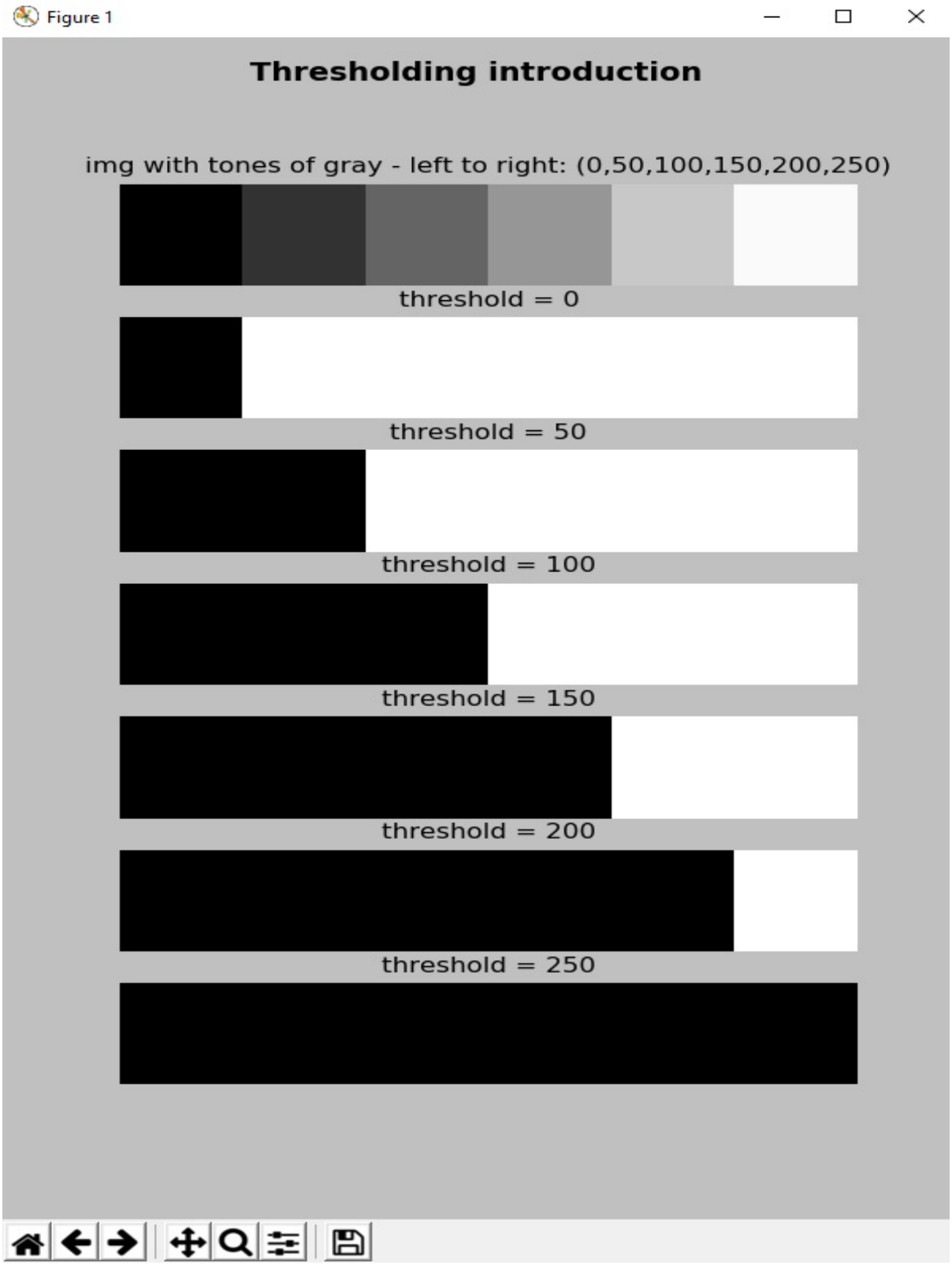
这里，thresh1是经过阈值处理的图像，为黑白图像。强度小于50的像素为黑色，强度大于50的像素为白色。

在下面的代码中可以看到另一个例子，其中的thresh5对应于经过阈值处理的图像

```
ret5, thresh5 = cv2.threshold(gray_image, 200, 255, cv2.THRESH_BINARY)
```

在这种情况下，强度小于200的像素为黑色，而强度大于200的像素为白色。

上述脚本的输出截图如下：



在截图中，可以看到源图像，它是一个样本图像，其中一些大小相等的区域填充了不同色调的灰色。更具体地说，这些灰色调是0、50、100、150、200和250。build_sample_image()函数生成示例图像，如下所示

```
def build_sample_image():
    """Builds a sample image with 50x50 regions of different tones of gray"""

    # Define the different tones.
    # The end of interval is not included
    tones  =  np.arange(start=50,  stop=300,  step=50)
    #   print(tones)

    # Initialize result with the first 50x50 region with 0-intensity level result = n

    # Build the image concatenating horizontally the regions: for tone in tones:
    img = np.ones((50, 50, 3), dtype="uint8") * tone result  =  np.concatenate((resul

    return result
```

用于生成此图像的NumPy函数(np.ones()、np.zeros()、np.arange()、np.concatenate()和np.fliplr())简述如下

ones():返回一个指定形状和类型的数组，其中填充了单位值;在本例中，形状为(50,50,3)，dtype="uint8"。
zeros():返回一个给定形状和类型的数组，其中填充了0;在本例中，形状为(50,50,3)，dtype="uint8"。
arange()：以所提供的步长，在给定间隔内返回均匀间隔的值，但不包括区间的最后一个值，在本例中为300。
concatenate():将数组序列沿现有轴(在本例中为轴=1)进行连接，以水平连接图像。

生成样本图像后，下一步是用不同的阈值对其进行阈值化。在本例中，阈值为0、50、100、150、200和250。

阈值对应样本图像中的不同灰度值。对具有不同阈值的样本图像进行阈值化的代码如下

```
ret1, thresh1 = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY)
ret2, thresh2 = cv2.threshold(gray_image, 50, 255, cv2.THRESH_BINARY)
ret3, thresh3 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_BINARY)
ret4, thresh4 = cv2.threshold(gray_image, 150, 255, cv2.THRESH_BINARY)
ret5, thresh5 = cv2.threshold(gray_image, 200, 255, cv2.THRESH_BINARY)
ret6, thresh6 = cv2.threshold(gray_image, 250, 255, cv2.THRESH_BINARY)
```

根据阈值和样本图像的不同灰度值，可以看到阈值化后的黑白图像是如何变化的

- 对图像进行阈值处理后，一般输出是黑白图像。在前几章中，屏幕截图的背景也是白色的。在本章中，使用fig.patch.set_facecolor('silver')将屏幕截图的背景改为银色。

7.5 简单阈值

简单阈值设置，OpenCV提供了cv2.threshold()函数，该函数在前一节中简单介绍过。此方法的签名如下

- cv2.threshold(src, thresh, maxval, type, dst=None) -> retval, dst

cv2.threshold()函数对源输入数组(多通道、8位或32位浮点)应用固定级别的阈值。级别调整由threshold参数进行，设置阈值类型，下一小节中进一步解释。

类型如下

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV
- cv2.THRESH_OTSU
- cv2.THRESH_TRIANGLE

此外，maxval参数设置cv2.THRESH_BINARY cv2.THRESH_BINARY_INV阈值类型的最大值。在cv2.THRESH_OTSU cv2.HRESH_TRIANGLE阈值类型,输入图像单通道的。

在本节将研究可能的配置，以理解所有这些参数。

7.6 阈值类型

阈值公式描述了阈值操作的类型。考虑到src是源(初始)图像，dst对应于阈值化后的目标(结果)图像。在这个意义上，src(x, y)对应源图像像素(x, y)的强度，dst(x, y)对应目标图像像素(x, y)的强度。

下面是cv2.threshold _binary的公式

$$dst(x,y) = \begin{cases} maxval & 如果src(x,y) > thresh \\ 0 & 其他 \end{cases}$$

如果像素src(x, y)的强度高于thresh，则将新像素强度设置为maxval参数,否则，像素设置为0

下面是cv2.threshold _binary_inv的公式

$$dst(x,y) = \begin{cases} 0 & 如果src(x,y) > thresh \\ maxval & 其他 \end{cases}$$

如果像素src(x, y)的强度高于thresh，则将新的像素强度设置为0。否则，设置为maxval。

下面是cv2.THRESH_TRUNC的公式

$$dst(x,y) = \begin{cases} threshold & 如果src(x,y) > thresh \\ src(x,y) & 其他 \end{cases}$$

如果像素src(x, y)的强度高于thresh，则将新的像素强度设置为threshold，否则，将其设置为src(x, y)。

下面是cv2.threshold _tozero的公式

$$dst(x,y) = \begin{cases} src(x,y) & 如果src(x,y) > thresh \\ 0 & 其他 \end{cases}$$

如果像素src(x, y)的强度高于thresh，新像素值设为0，否则src(x, y)，否则设为0。

下面是cv2.threshold _tozero_inv的公式

$$dst(x,y) = \begin{cases} 0 & 如果src(x,y) > thresh \\ src(x,y) & 其他 \end{cases}$$

如果像素src(x, y)的强度大于thresh，新像素值设置为0，否则设置为src(x, y)。

Also, the special cv2.THRESH_OTSU and cv2.THRESH_TRIANGLE values can be combined with one of the values previously introduced (cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO, and

cv2.THRESH_TOZERO_INV). In these cases (cv2.THRESH_OTSU and cv2.THRESH_TRIANGLE), the thresholding operation (implemented only for 8-bit images) computes the optimal threshold value instead of the specified thresh value. It should be noted that the thresholding operation returns the computed optimal threshold value.

cv2.THRESH_OTSU, cv2.THRESH_TRIANGLE与前面介绍的值之一组合(cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO, and cv2.THRESH_TOZERO_INV))。在这些情况下(cv2.THRESH_OTSU and cv2.THRESH_TRIANGLE)计算最佳阈值，而不是指定的thresh值。需要注意的是，阈值操作返回计算出的最优阈值。

The thresholding_simple_types.py script helps you understand the aforementioned types. We use the same sample image introduced in the previous section, and we perform a thresholding operation with a fixed threshold value (thresh = 100) with all the previous

thresholding_simple_types.p脚本帮助理解上述类型。我们使用上一节中介绍的相同的示例图像，并对前面的所有类型执行具有固定阈值(thresh = 100)的阈值操作。

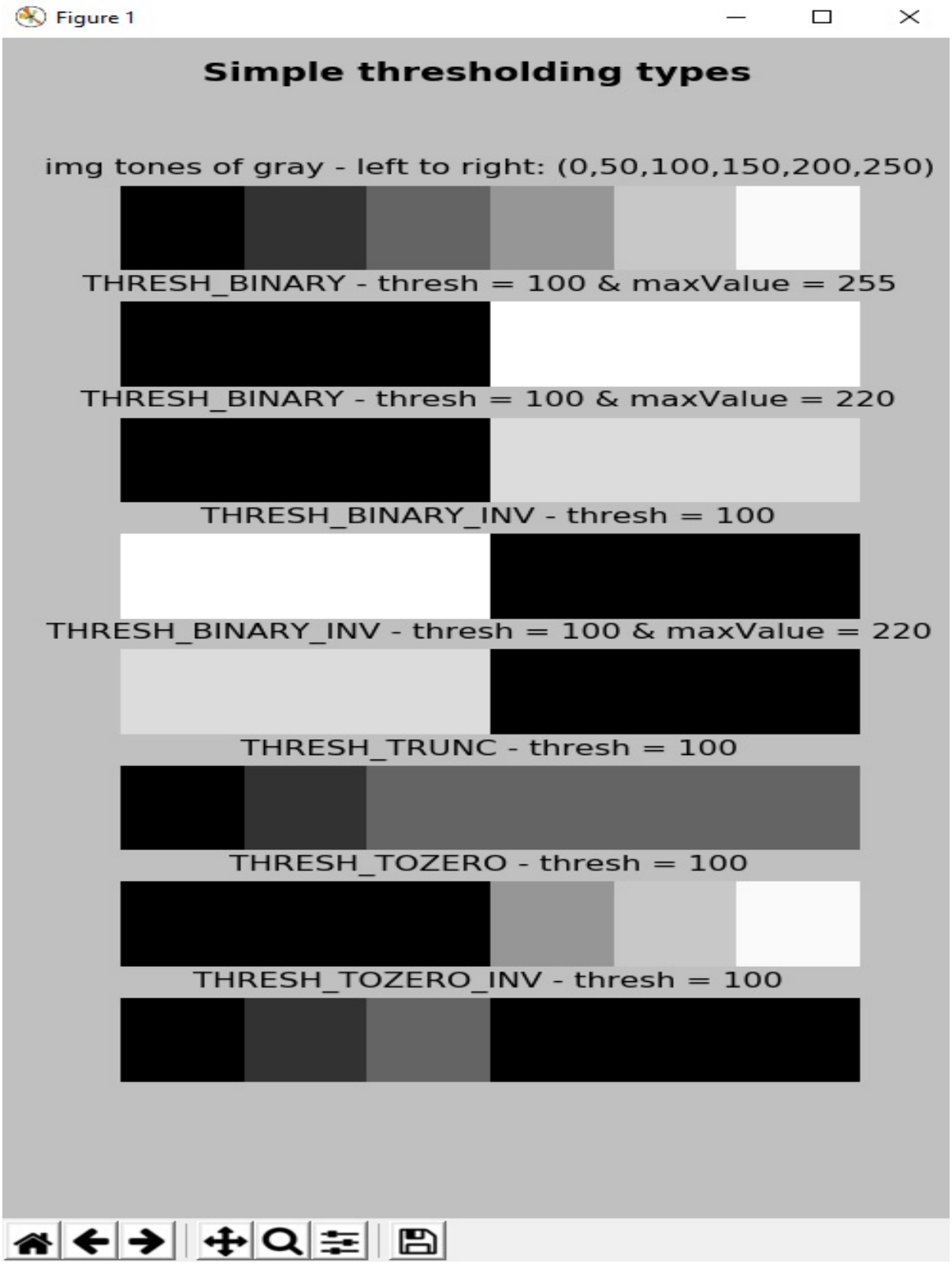
The key code to perform this is as follows:

执行此操作的关键代码如下

- ret1, thresh1 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_BINARY)
- ret2, thresh2 = cv2.threshold(gray_image, 100, 220, cv2.THRESH_BINARY)
- ret3, thresh3 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_BINARY_INV)
- ret4, thresh4 = cv2.threshold(gray_image, 100, 220, cv2.THRESH_BINARY_INV)
- ret5, thresh5 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_TRUNC)
- ret6, thresh6 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_TOZERO)
- ret7, thresh7 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_TOZERO_INV)

As previously mentioned, the maxval parameter sets the maximum value to use only with the cv2.THRESH_BINARY and cv2.THRESH_BINARY_INV thresholding types. In this example, we have set the value of maxval to 255 and 220 for the cv2.THRESH_BINARY and cv2.THRESH_BINARY_INV types in order to see how the thresholded image changes in both cases. The output of this script can be seen in the next screenshot:

如前所述，maxval参数仅用于cv2.THRESH_BINARY 和 cv2.THRESH_BINARY_INV阈值类型设置最大值。在本例中对cv2.THRESH_BINARY 和cv2.THRESH_BINARY_INV类型将cv2的maxval设置为255和220，查看在这两种情况下阈值化后的图像如何变化。脚本输出见截图：



In the previous screenshot, you can see both the original grayscale image and the result of each of the seven thresholding operations that were performed.

Additionally, you can see the effect of the maxval parameter, used only with the cv2.THRESH_BINARY and cv2.THRESH_BINARY_INV thresholding types. More specifically, see, for example, the difference between the first and second thresholding operation results - white versus gray in the result images - and also the difference between the third and fourth thresholding operation results - white versus gray in the result images.
在此截图中，可以看到原始灰度图像和七个阈值操作的结果。此外，可以看到cv2.THRESH_BINARY cv2和THRESH_BINARY_INV阈值类型中，maxval参数的效果。更具体地看，截图中第一个和第二个阈值操作，白色和灰色之间的区别，以及第三个和第四个阈值操作中，白色和灰色之间的区别。

7.7 Simple thresholding applied to a real image

简单的阈值应用于一个真实的图像

In the previous examples, we have applied the simple thresholding operation to a custom-made image in order to see how the different parameters work. In this section, we are going to apply cv2.threshold() to a real image. The thresholding_example.py script performs this. We applied the cv2.threshold() function with different thresholding values as follows - 60,70,80,90,100,110,120,130:

在前面的示例中将简单阈值操作应用于定制的图像，以查看不同的参数是如何工作的。在本节将把cv2.threshold()应用到实际图像中。thresholding_example.py脚本执行此操作。应用了具有不同阈值的cv2.threshold()函数，阈值为60、70、80、90,100,110,120,130，如下所示：

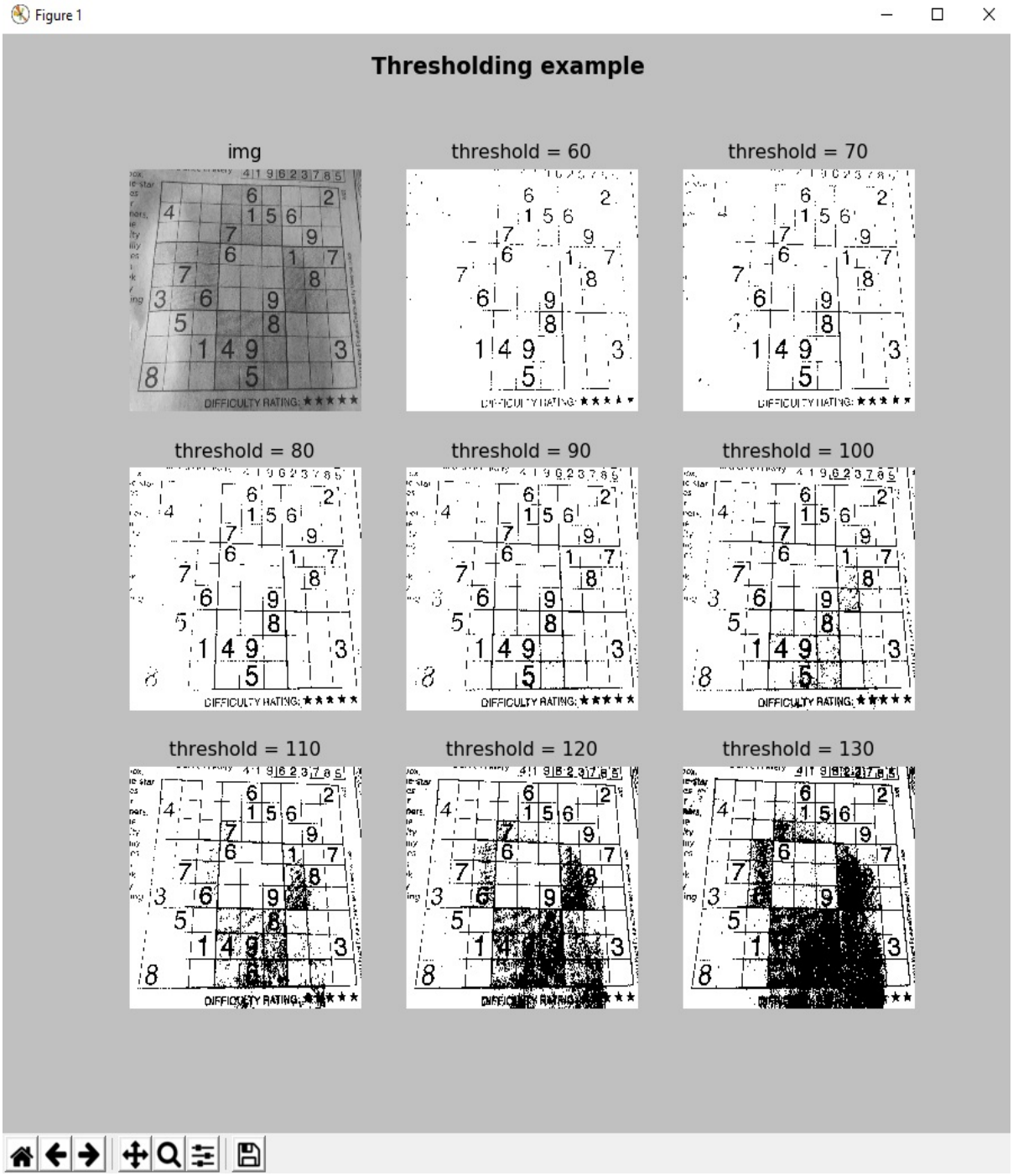
- ret1, thresh1 = cv2.threshold(gray_image, 60, 255, cv2.THRESH_BINARY)
- ret2, thresh2 = cv2.threshold(gray_image, 70, 255, cv2.THRESH_BINARY)
- ret3, thresh3 = cv2.threshold(gray_image, 80, 255, cv2.THRESH_BINARY)
- ret4, thresh4 = cv2.threshold(gray_image, 90, 255, cv2.THRESH_BINARY)
- ret5, thresh5 = cv2.threshold(gray_image, 100, 255, cv2.THRESH_BINARY)
- ret6, thresh6 = cv2.threshold(gray_image, 110, 255, cv2.THRESH_BINARY)
- ret7, thresh7 = cv2.threshold(gray_image, 120, 255, cv2.THRESH_BINARY)
- ret8, thresh8 = cv2.threshold(gray_image, 130, 255, cv2.THRESH_BINARY)

And finally, we show the thresholded images as follows:

最后，画出如下阈值图像

- show_img_with_matplotlib(cv2.cvtColor(thresh1, cv2.COLOR_GRAY2BGR), "threshold = 60", 2)
- show_img_with_matplotlib(cv2.cvtColor(thresh2, cv2.COLOR_GRAY2BGR), "threshold = 70", 3)
- show_img_with_matplotlib(cv2.cvtColor(thresh3, cv2.COLOR_GRAY2BGR), "threshold = 80", 4)
- show_img_with_matplotlib(cv2.cvtColor(thresh4, cv2.COLOR_GRAY2BGR), "threshold = 90", 5)
- show_img_with_matplotlib(cv2.cvtColor(thresh5, cv2.COLOR_GRAY2BGR), "threshold = 100", 6)
- show_img_with_matplotlib(cv2.cvtColor(thresh6, cv2.COLOR_GRAY2BGR), "threshold = 110", 7)
- show_img_with_matplotlib(cv2.cvtColor(thresh7, cv2.COLOR_GRAY2BGR), "threshold = 120", 8)
- show_img_with_matplotlib(cv2.cvtColor(thresh8, cv2.COLOR_GRAY2BGR), "threshold = 130", 9)

The output of this script can be seen in the following screenshot:
截图如下所示：



As you can see, the threshold value plays a critical role when thresholding images using cv2.threshold(). Suppose that your image-processing algorithm tries to recognize the digits inside the grid. If the threshold value is low (for example, threshold = 60), there are some digits missing in the thresholded image. On the other hand, if the threshold value is high (for example, threshold = 120), there are some digits occluded by black pixels. Therefore, establishing a global threshold value for the entire image is quite difficult. Moreover, if the image is affected by different illumination conditions, this task is almost impossible. This is why other thresholding algorithms can be applied to

threshold the images. In the next section, the adaptive thresholding algorithm will be introduced.

阈值在使用cv2.threshold()对图像进行阈值处理时起着关键作用。假设图像处理算法试图识别网格中的数字。
如果阈值很低(例如，阈值= 60)，则阈值图像中有一些数字缺失，另一方面，如果阈值很高(例如阈值= 120)，则会有一些数字被黑色遮挡。因此，为整个图像建立一个全局阈值是相当困难的。同时，如果图像受到不同光照条件的影响，这个任务几乎是不可能的。这就是为什么其他阈值算法可以应用到图像阈值化。
下一节将介绍自适应阈值算法。

Finally, you can see in the code snippet that we have created several thresholded images (one by one) with fixed threshold values. This can be optimized by creating an array containing the threshold values (using np.arange()) and iterating over the created array to call cv.threshold() for every value of the array. See the Questions section, because this optimization is proposed as an exercise.

代码中看到，我们已经一个接一个创建了几个具有固定阈值的图像，用np.range()创建包含阈值的数组，调用cv.threshold()遍历数组中的值来优化。请参阅问题部分，这个优化是作为练习提出的。

可在代码片段中看到，我们一个接一个创建具有固定阈值的图像。创建一个包含阈值的数组(使用np. range())，遍历该数组，为数组的每个值调用cv.threshold()进行优化。请参阅问题部分，优化是作为练习提出的。

7.8 Adaptive thresholding

自适应阈值分割

In the previous section, we have applied cv2.threshold() using a global threshold value. As we could see, the obtained results were not very good due to the different illumination conditions in the different areas of the image. In these cases, you can try adaptive thresholding. In OpenCV, the adaptive thresholding is performed by the cv2.adapativeThreshold() function. The signature for this method is as follows:

在前一节中，通过应用cv2.threshold()调用了全局阈值。可以看到，由于图像不同区域的光照条件不同，得到的结果不是很好。在这些情况下，可以尝试自适应阈值。在OpenCV中，自适应阈值由cv2.adapativeThreshold()函数执行。此方法的签名如下

- adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst]) -> dst

This function applies an adaptive threshold to the src array (8-bit single-channel image). The maxValue parameter sets the value for the pixels in the dst image for which the condition is satisfied. The adaptiveMethod parameter sets the adaptive thresholding algorithm to use:

该函数对src数组(8位单通道图像)应用自适应阈值。maxValue参数设置dst图像中满足条件的像素的值。adaptiveMethod参数设置要使用的自适应阈值算法

- cv2.ADAPTIVE_THRESH_MEAN_C: The T(x, y) threshold value is calculated as the mean of the blockSize x blockSize neighborhood of (x, y) minus the C parameter
- cv2.ADAPTIVE_THRESH_GAUSSIAN_C: The T(x, y) threshold value is calculated as the weighted sum of the blockSize x blockSize neighborhood of (x, y) minus the C parameter

需要改：

- cv2.ADAPTIVE_THRESH_MEAN_C: T(x, y) 阈值is calculated as the mean of the blockSize x blockSize neighborhood of (x, y) minus the C parameter
- cv2.ADAPTIVE_THRESH_GAUSSIAN_C: The T(x, y) threshold value is calculated as the weighted sum of the blockSize x blockSize neighborhood of (x, y) minus the C parameter

The blockSize parameter sets the size of the neighborhood area used to calculate a threshold value for the pixel, and it can take the values 3, 5, 7,... and so forth.

blockSize参数设置用于计算像素阈值的邻域区域大小，可以取3、5、7、…等等。

The C parameter is just a constant subtracted from the means or weighted means (depending on the adaptive method set by the adaptiveMethod parameter).

Commonly, this value is positive, but it can be zero or negative. Finally, the thresholdType parameter sets the cv2.THRESH_BINARY or cv2.THRESH_BINARY_INV thresholding types.C

C参数只是从平均值或加权平均值中减去的一个常数(取决于adaptiveMethod参数设置的自适应方法)。这个值通常是正的，但也可以是0或负的。阀值thresholdType参数设置cv2.HRESH_BINARY或cv2.THRESH_BINARY_INV。

According to the following formula where T(x, y) is the threshold calculated for each pixel, the thresholding_adaptive.py script applies adaptive thresholding to a test image using the cv2.ADAPTIVE_THRESH_MEAN_C and cv2.ADAPTIVE_THRESH_GAUSSIAN_C methods:

根据下面的公式，其中T(x, y)是为每个像素计算的阈值。ADAPTIVE_THRESH_MEAN_C cv2。ADAPTIVE_THRESH_GAUSSIAN_C方法

Here is the formula for cv2.THRESH_BINARY:

下面是cv2.threshold_binary的公式

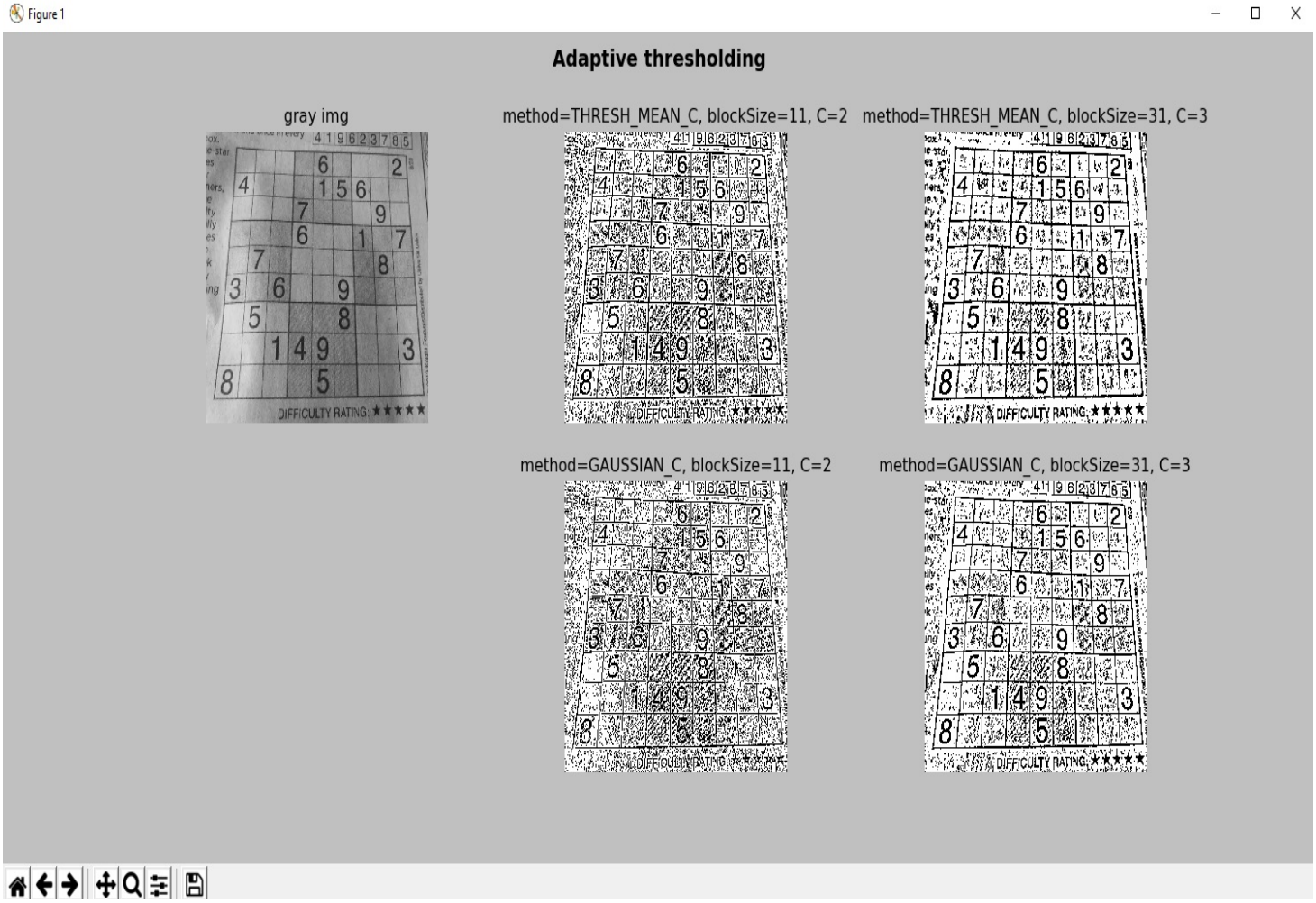
$$dst(x,y) = \begin{cases} maxValue & 如果src(x,y) > T(x,y) \\ 0 & 其他 \end{cases}$$

Here is the formula for cv2.THRESH_BINARY_INV:

下面是cv2.threshold_binary_inv的公式

$$dst(x,y) = \begin{cases} 0 & 如果src(x,y) > T(x,y) \\ maxValue & 其他 \end{cases}$$

The output of this script can be seen in the following screenshot:
脚本输出见屏幕截图:



In the previous screenshot, you can see the output after applying cv2.adaptiveThreshold() with different parameters. As previously mentioned, if your task is to recognize the digits, the adaptive thresholding can give you better thresholded images. However, as you can also see, a lot of noise appears in the image. In order to deal with it, you can apply some smoothing operations (see Chapter 5, Image Processing Techniques).

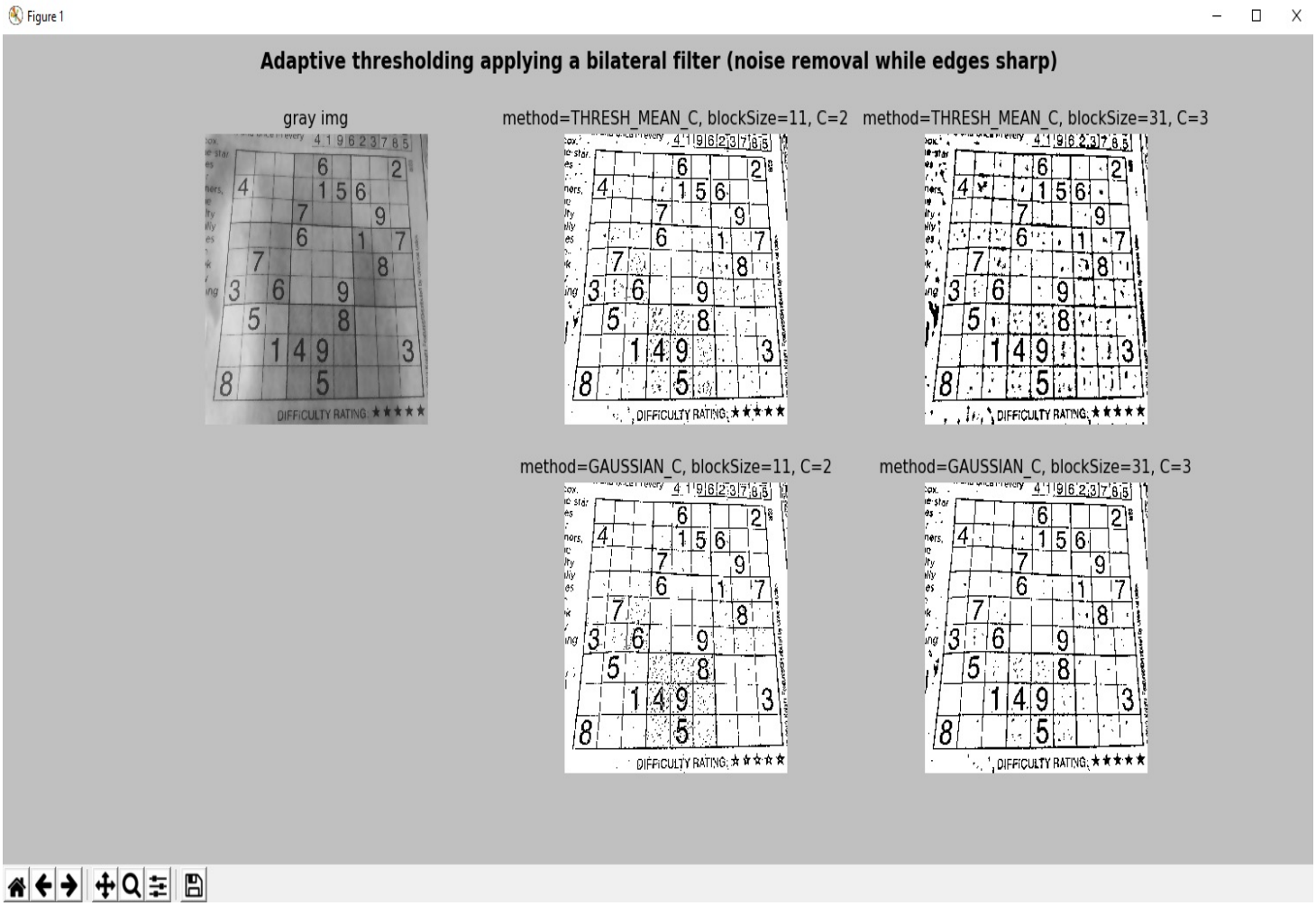
屏幕截图中，可以看到应用cv2.adaptiveThreshold()不同参数的输出。如前所述，如果任务是识别数字，自适应阈值可提供更好的阈值图像。也可以看到，图像中出现很多噪音。可应用平滑操作处理噪声(见第5章，图像处理技术)。

In this case, we can apply a bilateral filter, because it is highly useful in noise removal while maintaining sharp edges. In order to apply a bilateral filter, OpenCV provides the cv2.bilateralFilter() function. Therefore, we can apply the function before thresholding the image as follows:
在这种情况下，可用双边滤波器，因为它在去除噪声的同时保持锐边。为了调用双边过滤器，OpenCV提供了cv2.bilateralFilter()函数。可以在对图像进行阈值化之前应用该函数，如下所示

```
gray_image = cv2.bilateralFilter(gray_image, 15, 25, 25)
```

The code for this example can be seen in the thresholding_adaptive_filter_noise.py script. The output can be seen in the following screenshot:

此示例的代码为thresholding_adaptive_filter_noise.py 。输出的屏幕截图如下



7.9 Otsu's thresholding algorithm

Otsu阈值算法

As we saw in previous sections, the simple thresholding algorithm applies an arbitrary global threshold value. In this case, what we need to do is experiment with different thresholding values and look at the thresholded images in order to see if the result satisfies our necessities. However, this approach can be very tedious.

在前面几节可看到，简单阈值算法需要一个全局阈值。在这种情况下，我们需要做的是用不同的阈值进行实验，检查经过阈值处理的图像，查看结果是否满足需要。这种方法可能非常繁琐。

One solution is to use the adaptive thresholding that OpenCV provides by means of the cv2.adapativeThreshold() function. When applying adaptive thresholding in OpenCV, there is no need to set a thresholding value, which is a good thing.

一种解决方案是使用OpenCV提供的cv2.adapativeThreshold()函数的自适应阈值。在OpenCV中调用自适应阈值时，不需要设置阈值，是一件好事。

However, two parameters should be established correctly: the blockSize parameter and the C parameter. Another approach is to use Otsu's binarization algorithm, which is a good approach when dealing with bimodal images. A bimodal image can be characterized by its histogram containing two peaks. Otsu's algorithm automatically calculates the optimal threshold value that separates both peaks by maximizing the variance between two classes of pixels. Equivalently, the optimal threshold value minimizes the intra-class variance. Otsu's binarization algorithm is a statistical method, because it relies on statistical information derived from the histogram (for example, mean, variance, or entropy). In order to compute Otsu's binarization in OpenCV, we make use of the cv2.threshold() function as follows:

但要确立两个参数:blockSize参数和C参数。另一种方法是用Otsu的二值化算法，这是处理双峰图像的一种好方法。双峰图像的特征是它的直方图包含两个峰值。Otsu的算法通过最大化两类像素之间的方差来自动计算分离两个峰值的最佳阈值。同样，最优阈值最小化类内方差。Otsu的二值化算法是一种统计方法，它依赖于来自直方图的统计信息(例如均值、方差或熵)。在OpenCV中计算Otsu的二值化，使用cv2.threshold()函数

```
ret, th = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

In this case, there is no need to set a threshold value because Otsu's binarization algorithm calculates the optimal threshold value, that is why thresh = 0. The cv2.THRESH_OTSU flag indicates that Otsu's algorithm will be applied. Additionally, in this case, this flag is combined with cv2.THRESH_BINARY. In fact, it can be combined with cv2.THRESH_BINARY,

cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO, and cv2.THRESH_TOZERO_INV. This function returns the thresholded image, th, and the thresholded value, ret.

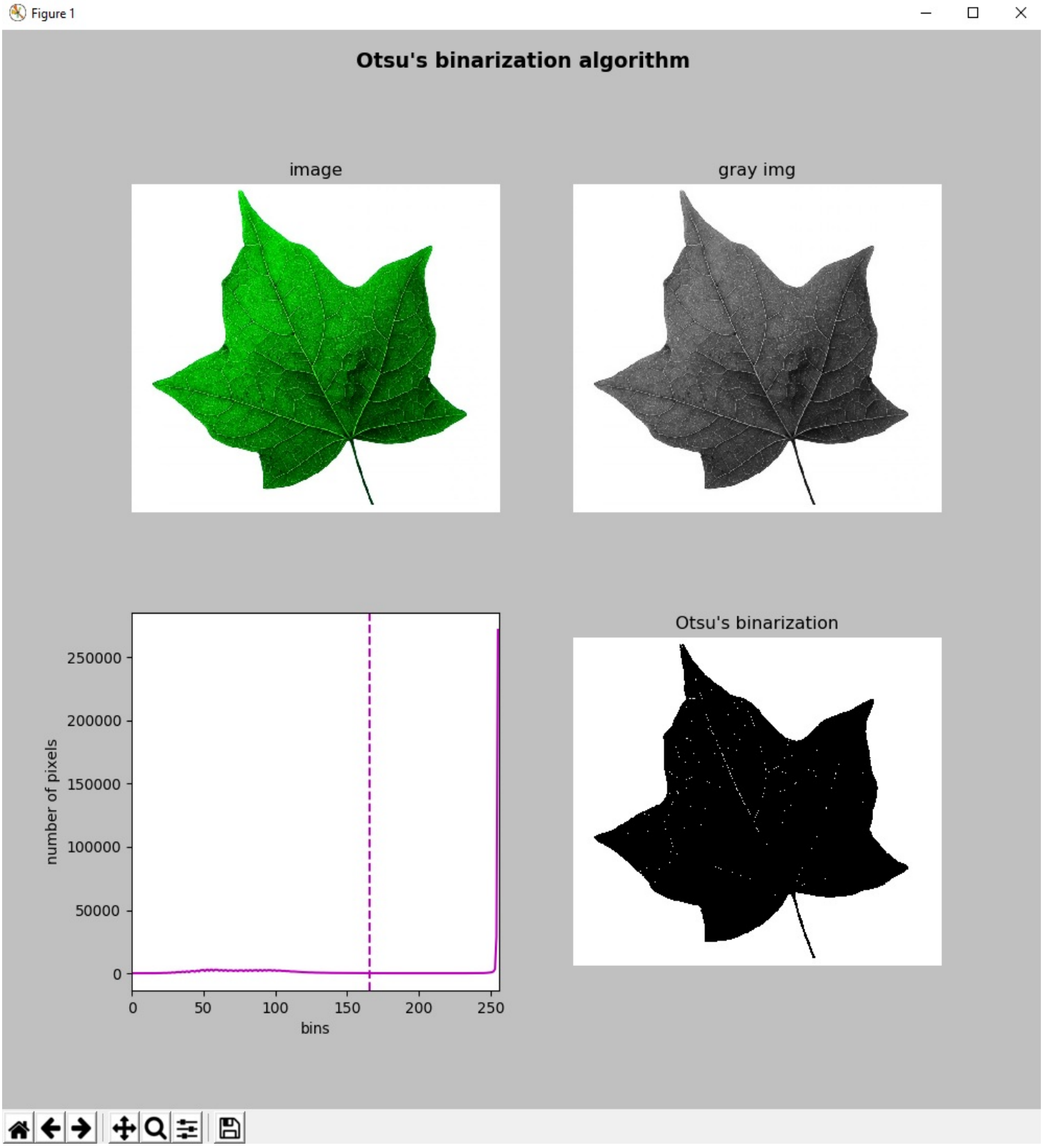
在这种情况下，不需要设置阈值，Otsu的二值化算法计算最优阈值，这就是为什么thresh = 0。cv2.THRESH_OTSU标志调用Otsu算法。在本例中，这个标志与cv2.threshold _binary结合在一起。事实上，还可以和cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO, and cv2.THRESH_TOZERO_INV结合在一起。函数返回阈值图像th和阈值值ret。

In the thresholding_otsu.py script, we have applied this algorithm to a sample image. The output can be seen in the following screenshot. We have modified the show_hist_with_matplotlib_gray() function to add an extra parameter that corresponds to the optimal threshold value calculated by Otsu's algorithm. In order to draw this threshold value, we draw a line establishing the x coordinate with the t threshold value as follows:

在threshold ding_otsu.py脚本中，该算法应用于样本图像。见下面的屏幕截图。我们修改了show_hist_with_matplotlib_gray()函数，添加一个额外的参数，该参数对应于Otsu算法计算出的最佳阈值。为了画出这个阈值，画了一条线来建立x坐标和t阈值，如下所示

```
plt.axvline(x=t, color='m', linestyle='--')
```

The output of the thresholding_otsu.py script can be seen in the following screenshot:
屏幕截图中可以看到threshold ding_otsu.py脚本的输出



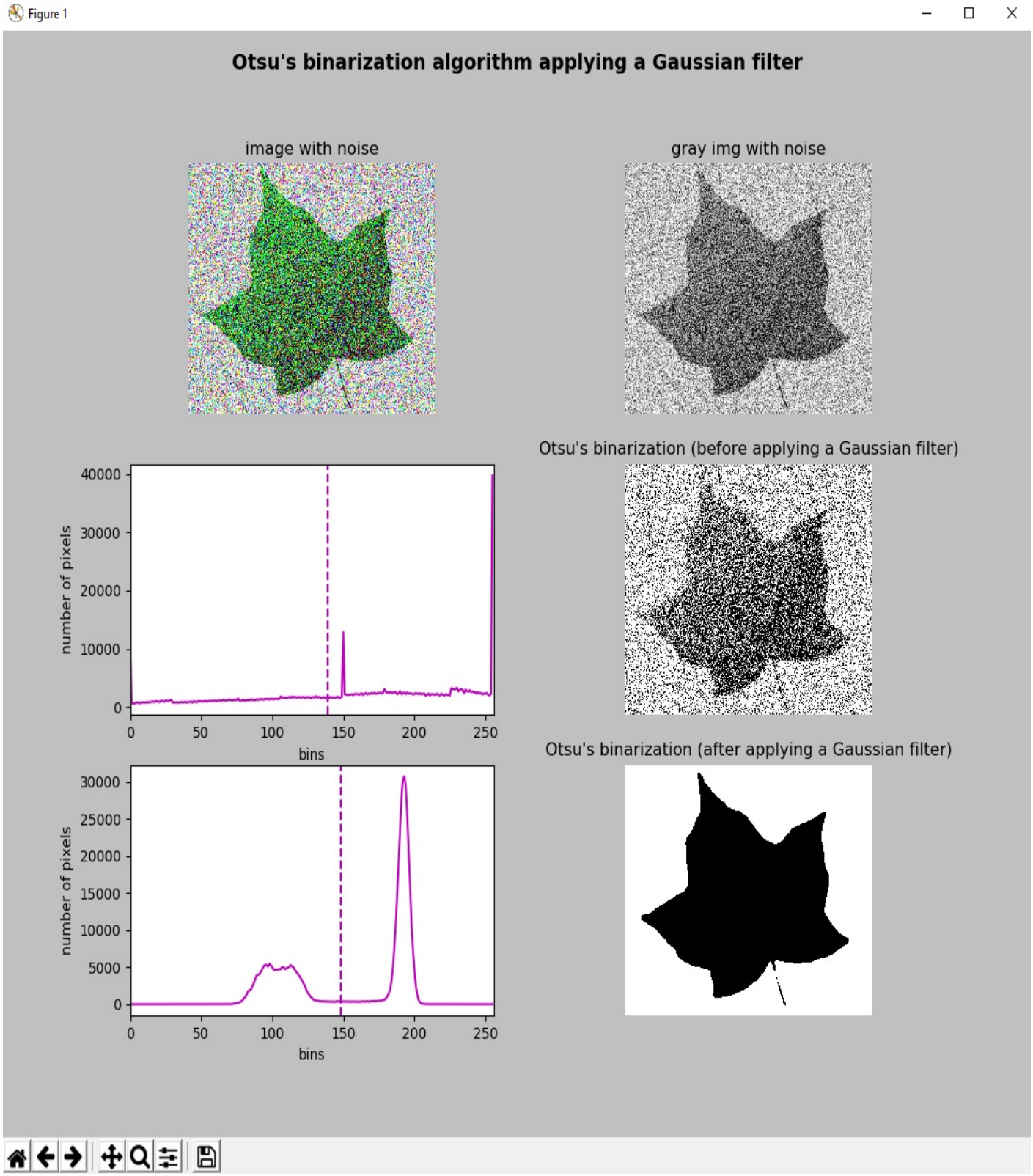
In the previous screenshot, we can see that the image is free of noise with a white background and a very well-defined green leaf. However, noise can affect the thresholding algorithm and we should deal with it properly. For example, in the previous section, we perform a bilateral filter in order to filter out some noise and preserve the edges. In the next example, we are going to add some noise to the leaf image, in order to see how the thresholding algorithm can be

affected. This can be seen in the thresholding_otsu_filter_noise.py script. In this script, we apply Otsu's binarization algorithm before and after applying a Gaussian filter in order to see how the thresholded image changes drastically.

在之前的截图中，我们可以看到图像是没有噪声的，背景是白色的，叶子是清晰的绿色。噪声会影响阈值算法，我们应该正确处理。例如，在前一节中，执行双边滤波器过滤掉噪声并保留边缘。在下一个示例中，我们将向叶片图像添加噪声，以便了解如何影响阈值算法。这可以在threshold ding_otsu_filter_noise.py脚本中看到。在这个脚本中，调用高斯滤波器之前和之后应用了Otsu的二值化算法，观察阈值化后的图像是如何急剧变化的。

This can be seen in the following screenshot:

这可以在下面的截图中看到



As we can see, if we do not apply a smoothing filter (a Gaussian filter, in this case), the thresholded image is also full of noise. However, applying a Gaussian filter is a good solution to filter the noise properly. Moreover, the filtered image is bimodal. This fact can be seen in the histogram corresponding to the filtered image. In this case, Otsu's binarization algorithm can segment the leaf properly.

可以看到，如果不用平滑滤波器，在这种情况下是高斯滤波器，阈值化的图像也会充满噪声。应用高斯滤波器可过滤噪声。此外，滤波后的图像是双峰的。可以从滤波后图像的直方图中看出。在这种情况下，Otsu的二值化算法可以对叶子进行适当的分割。

7.10 The triangle binarization algorithm

三角形二值化算法

Another automatic thresholding algorithm is the triangle algorithm, which is considered a shape-based method because it analyzes the structure (or shape) of the histogram (for example, trying to find valleys, peaks, and other shape histogram features). This algorithm works in three steps. In the first step, a line is calculated between the maximum of the histogram at bmax on the gray level axis and the lowest value bmin on the gray level axis. In the second step, the distance from the line (calculated in the first step) to the histogram for all the values of b [bmin-bmax] is calculated. Finally, in the third step, the level where the distance between the histogram and the line is maximal is chosen as the threshold value.

另一种自动阈值算法是三角形算法，是一种基于形状的方法，因为它分析直方图的结构或形状，例如，试图找到谷、峰和其他形状直方图特征。该算法分三步进行。第一步，计算灰度轴上bmax处直方图的最大值与灰度轴上bmin的最小值之间的直线。在第二步，计算所有b [bmin-bmax]值到直方图的距离(第一步计算)。在第三步中，选择直方图与直线距离最大的那一层作为阈值。

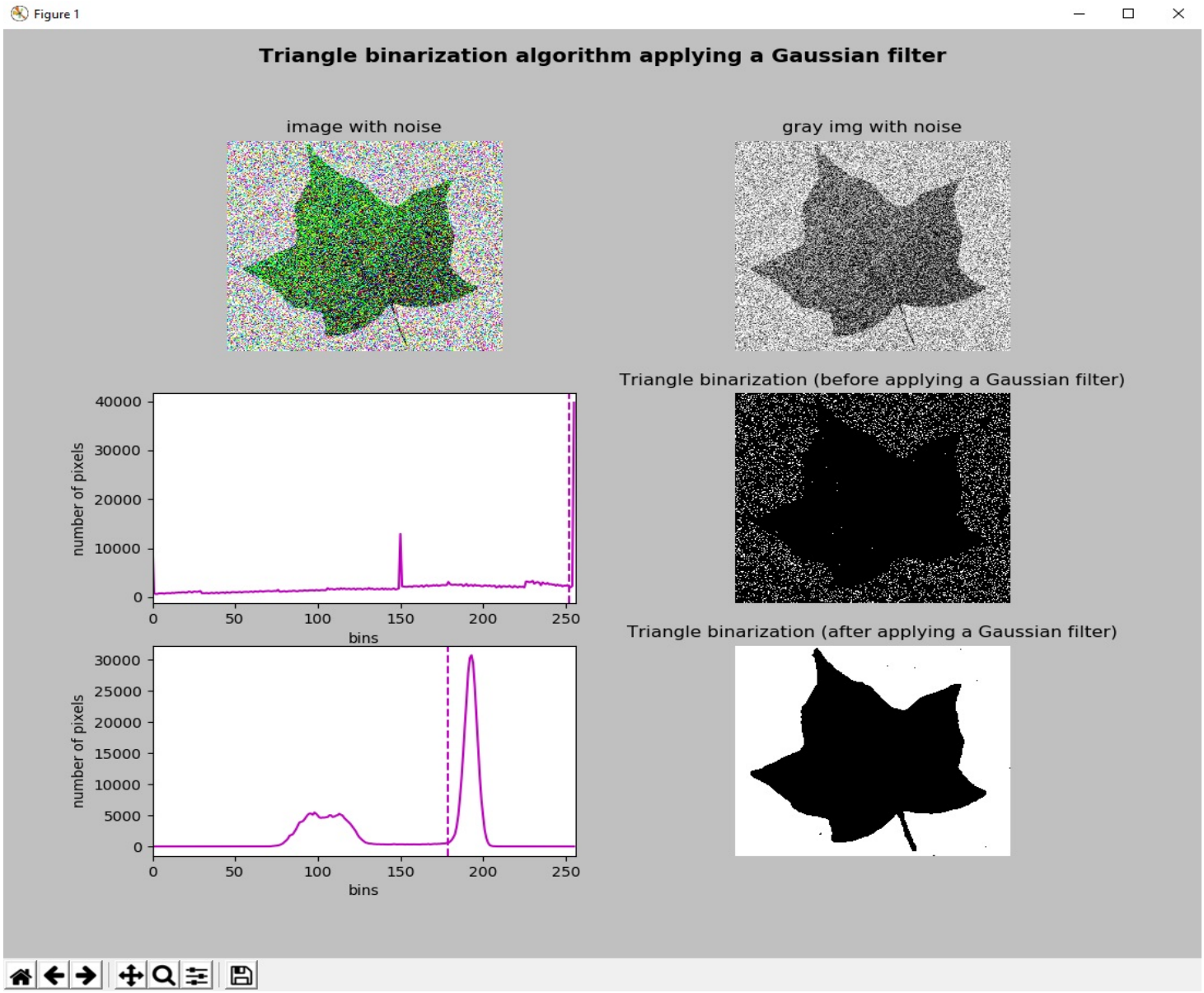
The way to use the triangle binarization algorithm in OpenCV is very similar to Otsu's algorithm. In fact, only one flag should be changed properly. In case of Otsu's binarization, the cv2.THRESH_OTSU flag was set. In the case of the triangle binarization algorithm, the flag is cv2.THRESH_TRIANGLE, as follows:

OpenCV中使用的三角形二值化算法与Otsu算法非常相似。事实上，只有一个标志应该被正确地改变。在 Otsu二值化中，设置了cv2.THRESH_OTSU。三角二值化算法中，设置cv2.THRESH_TRIANGLE,如下所示

```
ret1, th1 = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_TRIANGLE
```

In the next screenshot, you can see the output when applying the triangle binarization algorithm to a noise image (the same image that was used in Otsu's binarization example in the previous section). The full code for this example can be seen in
the thresholding_triangle_filter_noise.py script:

在下一屏幕截图中，可以看到将三角形二值化算法应用到噪声图像(与上一节Otsu二值化示例中的图像相同)时的输出。本例的完整代码见thresholding_triangle_filter_noise.py



You can see that applying a Gaussian filter is a good solution to filter the noise. This way, the triangle binarization algorithm can segment the leaf properly.

可以看到，应用高斯滤波器可以过滤噪声。三角形二值化算法就可以对叶子进行适当的分割。

7.11 Thresholding color images

阈值彩色图像

The cv2.threshold() function can also be applied to multi-channel images. This can be seen in the thresholding_bgr.py script. In this case, the cv2.threshold() function applies the thresholding operation in each of the channels of the BGR image. This produces the same result as applying this function in each channel and merging the thresholded channels:

cv2.threshold()函数也可以应用于多通道图像。见threshold ding_bgr.py。在这种情况下，cv2.threshold()函数在BGR图像的每个通道进行阈值操作。与每个通道中应用此函数并合并阈值化通道有相同的结果

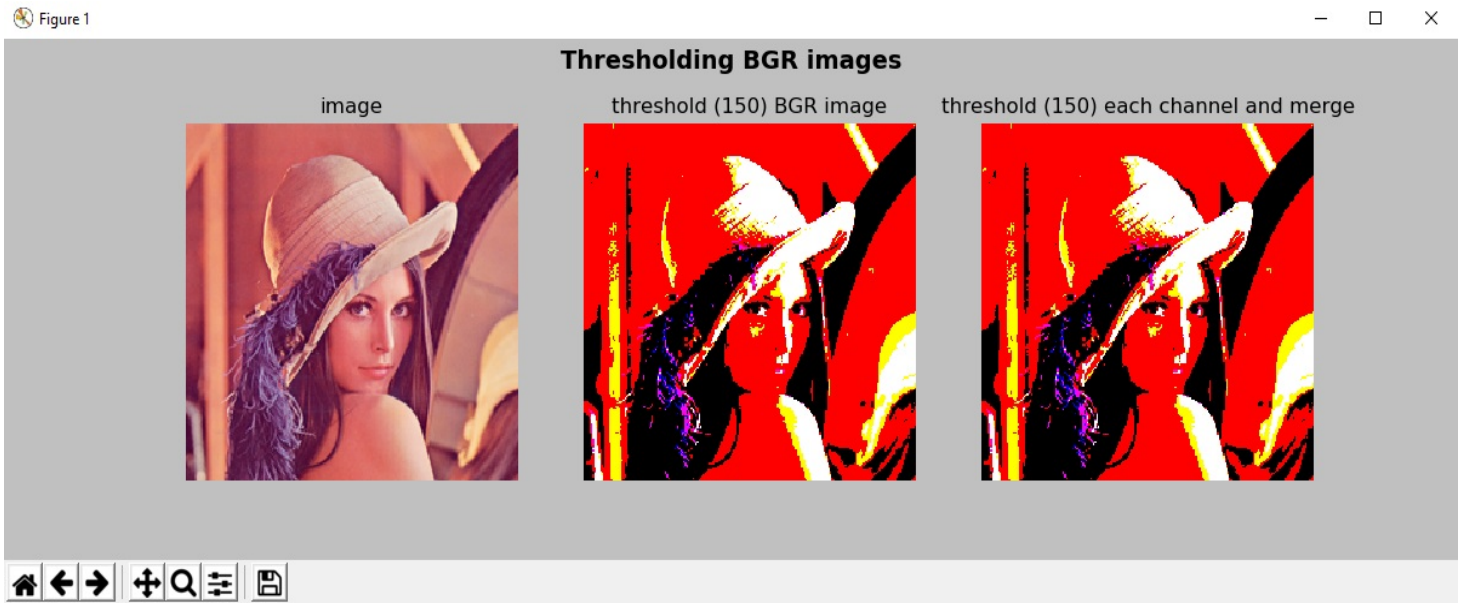
```
ret1, thresh1 = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY)
```

Therefore, the preceding line of code produces the same result as performing the following:

因此，前一行代码产生的结果与执行以下代码的结果相同

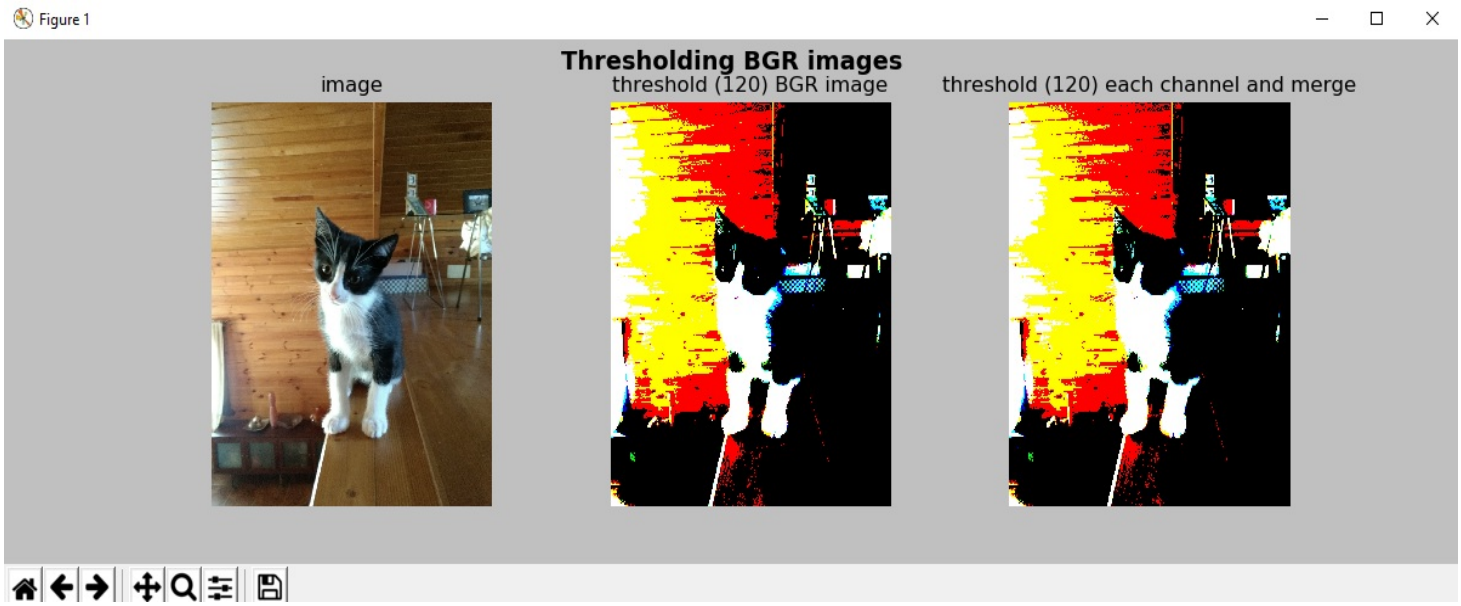
```
(b, g, r) = cv2.split(image)
ret2, thresh2 = cv2.threshold(b, 150, 255, cv2.THRESH_BINARY)
ret3, thresh3 = cv2.threshold(g, 150, 255, cv2.THRESH_BINARY)
ret4, thresh4 = cv2.threshold(r, 150, 255, cv2.THRESH_BINARY)
bgr_thresh = cv2.merge((thresh2, thresh3, thresh4))
```

The result can be seen in the following screenshot:
结果如下图所示



Although you can perform cv2.threshold() on multi-channel images (for example, BGR images), this operation can produce weird results. For example, in the resulting image, as each channel can take only two values (0 and 255 in this case), the final image has only 23 possible colors. In the next screenshot, we also apply this color thresholding to another test image:

虽然可以在多通道图像(例如BGR图像)上执行cv2.threshold()，但此操作可能会产生奇怪的结果。例如，在生成的图像中，由于每个通道只能接受两个值(本例中为0和255)，因此最终图像只有23种可能的颜色。在下一个屏幕快照中，我们还将此颜色阈值应用于另一个测试图像



As shown in the previous screenshot, the output image has only 23 possible colors. Therefore, you should take this into account when performing thresholding operations in BGR images.

如前面的屏幕截图所示，输出图像只有23种可能的颜色。因此，在BGR映像中执行阈值操作时应该考虑这一点

7.12 Thresholding algorithms using scikit- image

使用scikit-image的阈值算法

As we mentioned in Chapter 1, Setting Up OpenCV, there are other packages that can be used for scientific computing, data science, machine learning, deep learning, and computer vision. In connection with computer vision, scikit-image is a collection of algorithms for image processing (https://scikit-image.org/). Images manipulated by scikit- image are NumPy arrays.

正如我们在第1章“设置OpenCV”中提到的，还有其他一些包可以用于科学计算、数据科学、机器学习、深度学习和计算机视觉。关于计算机视觉，scikit-image是一个图像处理的算法集合(https://scikit-image.org/)。由scikit- image操作的图像是数字数组。

In this section, we will make use of scikit-image capabilities in connection with thresholding techniques. So, if you want to replicate the results obtained here, the first step is to install it. See https://scikit-image.org/download.html in order to install scikit-image properly on your operating system. Here, we are going to install it with conda, which is an open source package management system (and also an environment management system). See Chapter 1, Setting Up OpenCV, in order to see how to install Anaconda/Miniconda distributions and conda. To install scikit-image for conda-based distributions (Anaconda, Miniconda), execute the following code:

在本节将使用与阈值技术相关的scikit-image功能。若想复制这里的结果，第一步是安装。请参阅https://scikit-image.org/download.html，安装scikit-image。在这里，我们将使用conda安装，它是开源包管理系统(也是一个环境管理系统)。参见第一章，设置OpenCV，以了解如何安装Anaconda/Miniconda发行版和conda。基于conda的发行版(Anaconda、Miniconda)安装scikit-image，执行以下代码

```
conda install -c conda-forge scikit-image
```

7.13 Introducing thresholding with scikit- image

scikit-image阈值处理介绍

In order to test scikit-image, we are going to threshold a test image using Otsu's binarization algorithm. In order to try this method, the first step is to import the required packages. In this case, in connection with scikit-image as follows:

为了测试scikit-image，采用Otsu的二值化算法对检测图像进行阈值处理。首先导入所需的包。

```
from skimage.filters   import   threshold_otsu
from skimage import img_as_ubyte
```

The key code to apply Otsu's binarization algorithm with scikit-image is the following:

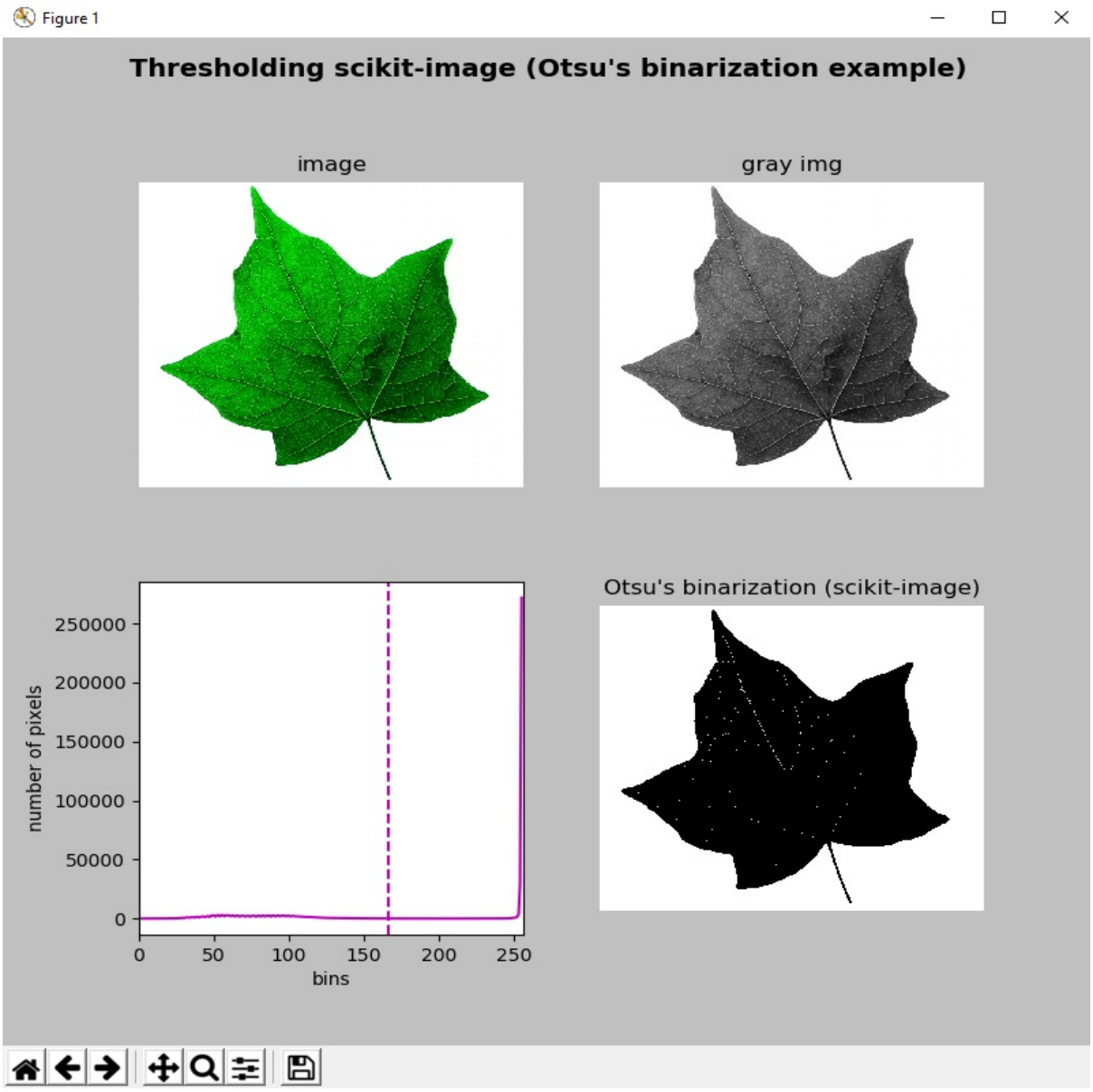
将Otsu二值化算法应用于scikit-image的关键代码如下

```
thresh    =    threshold_otsu(gray_image)
binary  =  gray_image  >  thresh
binary  =  img_as_ubyte(binary)
```

The threshold_otsu(gray_image) function returns the threshold value based on Otsu's binarization algorithm. Afterwards, with this value, the binary image is constructed (dtype= bool), which should be converted to 8-bit unsigned integer format (dtype= uint8) for proper visualization. The img_as_ubyte() function is used for this purpose. The full code for this example can be seen in the thresholding_scikit_image_otsu.py script.

threshold_otsu(gray_image)函数的作用是:返回基于Otsu二值化算法的阈值。然后，用这个值构造二进制图像(dtype= bool)，应该将其转换为8位无符号整数格式(dtype= uint8)，以便进行适当的可视化。img_as_ubyte()函数用于此目的。本例的完整代码可以在threshold _scikit_image_otsu.py脚本中看到。

The output can be seen in the following screenshot:
输出见屏幕截图



We will now try out some thresholding techniques with scikit-image.

现在试着试用scikit-image阈值技术

7.14 Trying out more thresholding techniques with scikit-image

更多的scikitimage阈值技术

We are going to threshold a test image comparing Otsu's, triangle, Niblack's, and Sauvola's thresholding techniques. Otsu and triangle are global thresholding techniques, while Niblack and Sauvola are local thresholding techniques. Local thresholding techniques are considered a better approach when the background is not uniform. For more information about Niblack's and Sauvola's thresholding algorithms, see An Introduction to Digital Image Processing (1986) and Adaptive

document image binarization (2000), respectively. The full code for this example can be seen in the thresholding_scikit_image_techniques.py script. In order to try these methods, the first step is to import the required packages. In this case, in connection with scikit-image as follows:

我们将阈值测试图像，对Otsu，三角形，Niblack，和Sauvola的阈值技术进行比较。Otsu和triangle是全局阈值技术，而Niblack和Sauvola是局部阈值技术。当背景不均匀时，局部阈值技术被认为是一种较好的方法。有关Niblack和Sauvola的阈值算法的更多信息，请参见数字图像处理导论(1986)和自适应文档图像二值化(2000)。本例的完整代码见threshold_scikit_image_technique.py。为了调用这些方法，首先导入所需的包。在这种情况下，关于scikit-image如下

```
from from skimage.filters import (threshold_otsu, threshold_triangle, threshold_nibla
from skimage import img_as_ubyte
```

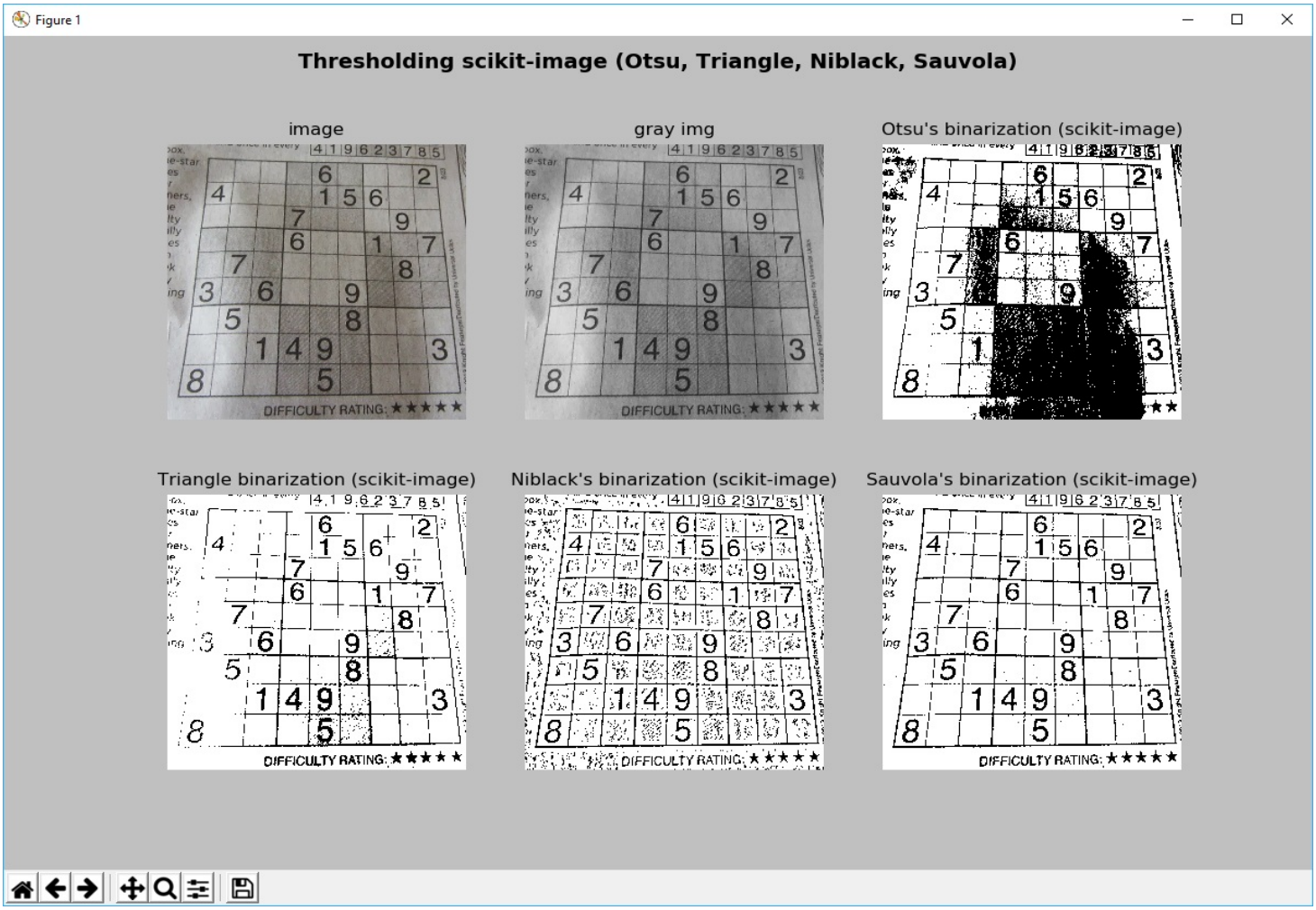
In order to perform the thresholding operations with scikit-image, we call each thresholding method (threshold_otsu(), threshold_niblack(), threshold_sauvola(),and threshold_triangle()):

为了使用scikit-image阈值操作，调用多个阈值方法threshold_otsu(), threshold_niblack(), threshold_sauvola() 和 threshold_triangle()):

```
# Trying Otsu's scikit-image algorithm:
thresh_otsu = threshold_otsu(gray_image)
binary_otsu = gray_image > thresh_otsu
binary_otsu = img_as_ubyte(binary_otsu)

# Trying Niblack's scikit-image algorithm:
thresh_niblack = threshold_niblack(gray_image, window_size=25, k=0.8)
binary_niblack = gray_image > thresh_niblack
binary_niblack = img_as_ubyte(binary_niblack)

# Trying Sauvola's scikit-image algorithm:
thresh_sauvola = threshold_sauvola(gray_image, window_size=25)
binary_sauvola = gray_image > thresh_sauvola
binary_sauvola = img_as_ubyte(binary_sauvola)

# Trying triangle scikit-image algorithm:
thresh_triangle = threshold_triangle(gray_image)
binary_triangle = gray_image > thresh_triangle
binary_triangle = img_as_ubyte(binary_triangle)
```

The output can be seen in the next screenshot:

输出见截图



As you can see, local thresholding methods can provide better results when the background is not uniform. Indeed, these methods can be applied for text recognition. Finally, scikit-image comes with more thresholding techniques that you

can try. If necessary, consult the API documentation to see all available methods at http:

//scikit-image.org/docs/dev/api/api.html.

可所看到的，当背景不一致时，局部阈值方法可以提供更好的结果。这些方法可以用于文本识别。最后，scikit-image提供了很多阈值技术。如果需要，请参考API文档，查看http://scikit-image.org/docs/dev/api/api.html中的所有可用方法。

7.15 Summary

In this chapter, we have reviewed the main thresholding techniques you can use to threshold your images. Thresholding techniques can be used for many computer vision tasks (for example, text recognition and image segmentation, among others). Both simple and adaptive thresholding techniques have been reviewed.

Additionally, we have seen how to apply Otsu's binarization algorithm and the triangle algorithm to automatically select a global threshold for thresholding your images. Finally, we have seen how to use different thresholding techniques using scikit-image. In this sense, two global thresholding techniques (Otsu's and triangle algorithms) and two local thresholding techniques (Niblack's and Sauvola's algorithms) have been applied to a test image.

总结

在这一章回顾了主要的阈值技术，用于阈值化图像。阈值技术可以用于许多计算机视觉任务，如文本识别和图像分割等。综述了简单阈值技术和自适应阈值技术，如何应用Otsu的二值化算法和三角形算法自动选择全局阈值对图像进行阈值化，及在scikit-image中如何使用不同的阈值技术使用。在这个意义上，两种全局阈值技术(Otsu和三角形算法)和两种局部阈值技术(Niblack和Sauvola算法)应用于测试图像。

In Chapter 8, Contour Detection, Filtering, and Drawing, we will see how to deal with contours, which are very useful tools for shape analysis and object detection and recognition.

在第8章为轮廓检测，滤波，和绘图，将看到如何处理轮廓，这是非常有用的形状分析和对象检测和识别工具

7.16 Questions

1. Apply a thresholding operation using cv2.threshold() with a threshold value of 100 and using the cv2.THRESH_BINARY thresholding type.
2. Apply an adaptive thresholding operation using cv2.adapativeThreshold(),cv2.ADAPTIVE_THRESH_MEAN_C, C=2 and blockSize=9.
3. Apply Otsu's thresholding using the cv2.THRESH_BINARY thresholding type.
4. Apply triangle thresholding using the cv2.THRESH_BINARY thresholding type.
5. Apply Otsu's thresholding using scikit-image.
6. Apply triangle thresholding using scikit-image.
7. Apply Niblack's thresholding using scikit-image.
8. Apply Sauvola's thresholding using scikit-image and a window size of 25.
9. Modify the thresholding_example.py script in order to make use of np.arange(), with the purpose of defining the threshold values to apply to

the cv2.threshold() function. Afterwards, call the cv2.threshold() function with the defined threshold values and store all the thresholded images in an array. Finally, show all the images in the array calling show_img_with_matplotlib() for each one. Rename the script to thresholding_example_arange.py.

问题

1. 应用cv2.threshold()操作阈值100，用cv2.THRESH_BINARY阈值类型。
2. 应用cv2.adapativethreshold()、cv2.ADAPTIVE_THRESH_MEAN_C，C=2 及 blockSize=9 进行自适应阈值操作
3. 应用Otsu阈值，cv2.THRESH_BINARY thresholding阈值类型。
4. 应用三角形阈值，cv2.THRESH_BINARY阈值类型。
5. 应用Otsu阈值，使用scikit-image
6. 应用三角形阈值，使用scikit-image
7. 应用Niblack阈值，使用scikit-image
8. 应用Sauvola阈值，窗口大小25，使用scikit-image
9. 修改threshold _example.py，使用np.arange()，目标为应用cv2.threshold()函数定义阈值。使用调用cv2.threshold()函数定义阈值，并将所有经过阈值处理的图像存储在数组中。调用show_img_with_matplotlib()显示数组中的所有图像。将脚本重命名为threshold _example_alan.py。