

实战项目 2：毛笔书体检测与识别

目标

- 使用传统形态学方法（腐蚀、膨胀）检测目标
- 使用传统机器学习方法（HOG+SVM）图像分类

形态学变换（morphological）

形态学变换：基于图像形状的一些简单操作，一般基于单通道图处理（常用灰度图）；

一般有两个输入，一是要操作的图片，二是要变换的结构元素或核

两种基本的形态学变换是侵蚀和膨胀，他们的变种也有张开和闭合

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

img = cv2.imread('./test_imgs/j.png')

img.shape

>(150, 112, 3)
```

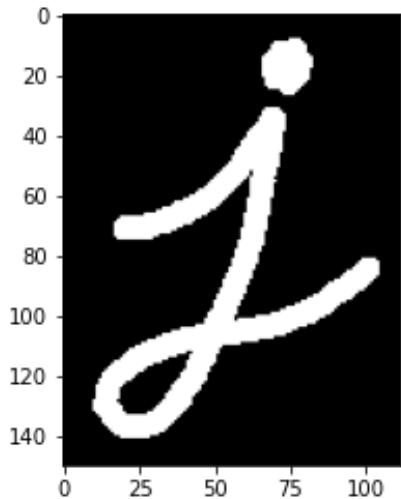
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

gray.shape

>(150, 112)
```

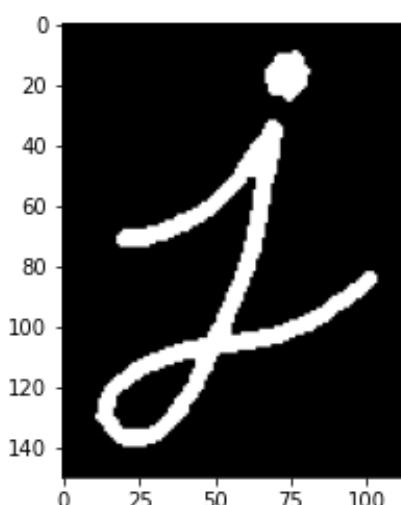
```
plt.imshow(gray, cmap='gray')

><matplotlib.image.AxesImage at 0x7fad7df3e490>
```

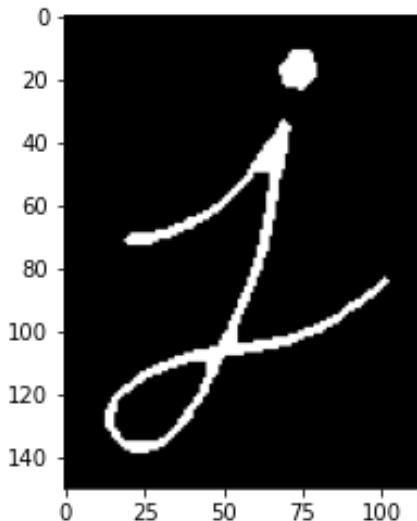


```
# 侵蚀 cv2.erode(图片, 内核, 迭代次数)  
# 作用: 去除白色噪点, 将两个连起来的形状打散  
# 核大小: 3x3, 迭代次数: 1
```

```
kernel = np.ones((3, 3), dtype=np.int8)  
  
ersion1 = cv2.erode(gray.copy(), kernel, iterations=1)  
  
plt.imshow(ersion1, cmap='gray')  
  
><matplotlib.image.AxesImage at 0x7fad7e1c9e50>
```

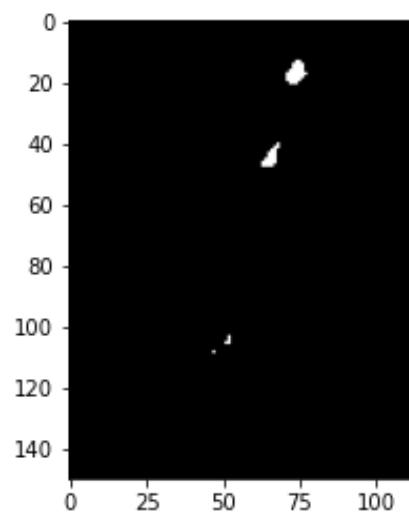


```
# 核大小: 5x5, 迭代次数: 1 (核大小越大, 侵蚀越严重)  
  
kernel = np.ones((5, 5), dtype=np.int8)  
  
ersion2 = cv2.erode(gray.copy(), kernel, iterations=1)  
  
plt.imshow(ersion2, cmap='gray')  
  
><matplotlib.image.AxesImage at 0x7fad7b99e690>
```



核大小: 5x5, 迭代次数: 2 (核大小不变, 迭代次数增加, 侵蚀越严重)

```
kernel = np.ones((5,5), dtype=np.int8)
erosion3 = cv2.erode(gray.copy(), kernel, iterations=2)
plt.imshow(erosion3, cmap='gray')
><matplotlib.image.AxesImage at 0x7fad7e293910>
```



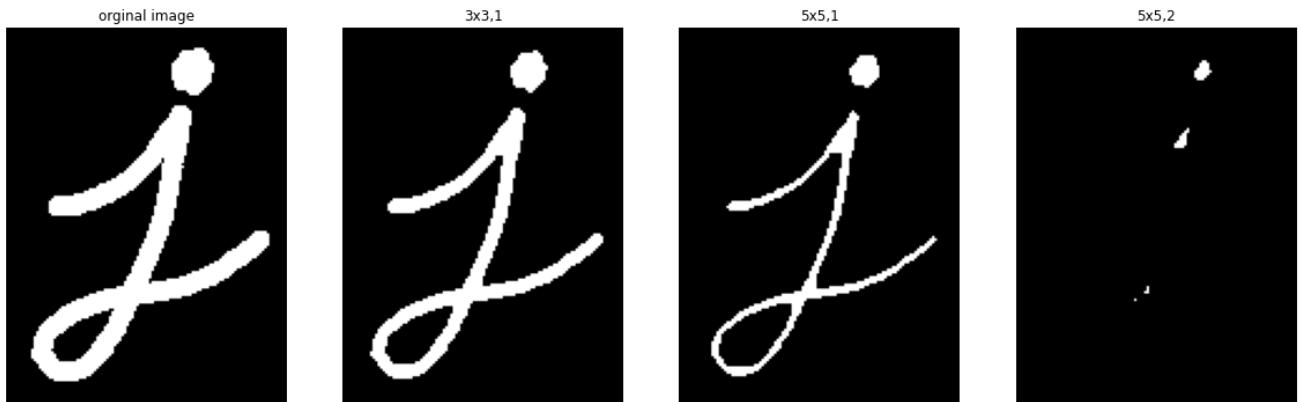
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20, 8), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(gray.copy(), cmap='gray')
ax1.set_title('original image')

ax2.axis('off')
```

```
ax2.imshow(ersion1,cmap='gray')
ax2.set_title('3x3,1')
```

```
ax3.axis('off')
ax3.imshow(ersion2,cmap='gray')
ax3.set_title('5x5,1')
```

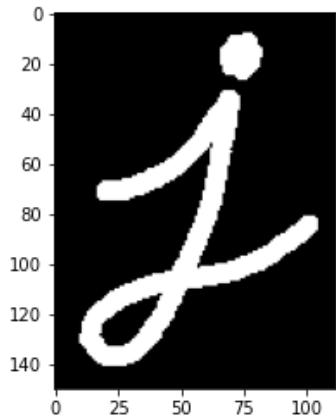
```
ax4.axis('off')
ax4.imshow(ersion3,cmap='gray')
ax4.set_title('5x5,2')
>Text(0.5, 1.0, '5x5,2')
```



膨胀 cv2.dilate(图片, 内核, 迭代次数)

作用：跟在侵蚀操作后去噪点，把两个分开的部分连接起来

```
plt.imshow(gray.copy(),cmap='gray')
><matplotlib.image.AxesImage at 0x7fad7eb67810>
```



核大小: 3x3, 迭代次数: 1

```
kernel = np.ones((3,3), dtype=np.int8)  
dilation1 = cv2.dilate(gray.copy(), kernel, iterations=1)
```

核大小: 5x5, 迭代次数: 1

```
kernel = np.ones((5,5), dtype=np.int8)  
dilation2 = cv2.dilate(gray.copy(), kernel, iterations=1)
```

核大小: 5x5, 迭代次数: 2

```
kernel = np.ones((5,5), dtype=np.int8)  
dilation3 = cv2.dilate(gray.copy(), kernel, iterations=2)
```

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20, 8), sharex=True, sharey=True)
```

```
ax1.axis('off')
```

```
ax1.imshow(gray.copy(), cmap='gray')
```

```
ax1.set_title('original image')
```

```
ax2.axis('off')
```

```
ax2.imshow(dilation1, cmap='gray')
```

```
ax2.set_title('3x3, 1')
```

```
ax3.axis('off')
```

```
ax3.imshow(dilation2, cmap='gray')
```

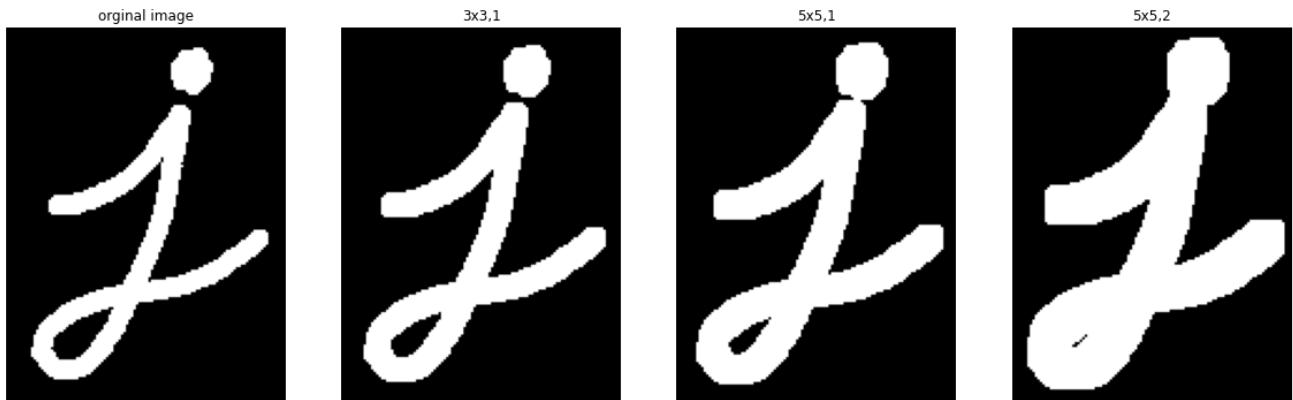
```
ax3.set_title('5x5, 1')
```

```
ax4.axis('off')
```

```
ax4.imshow(dilation3, cmap='gray')
```

```
ax4.set_title('5x5, 2')
```

```
>Text(0.5, 1.0, '5x5, 2')
```



opening, 张开 cv2.morphologyEx(图片, 模型: 张开, 内核大小)

蚀+膨

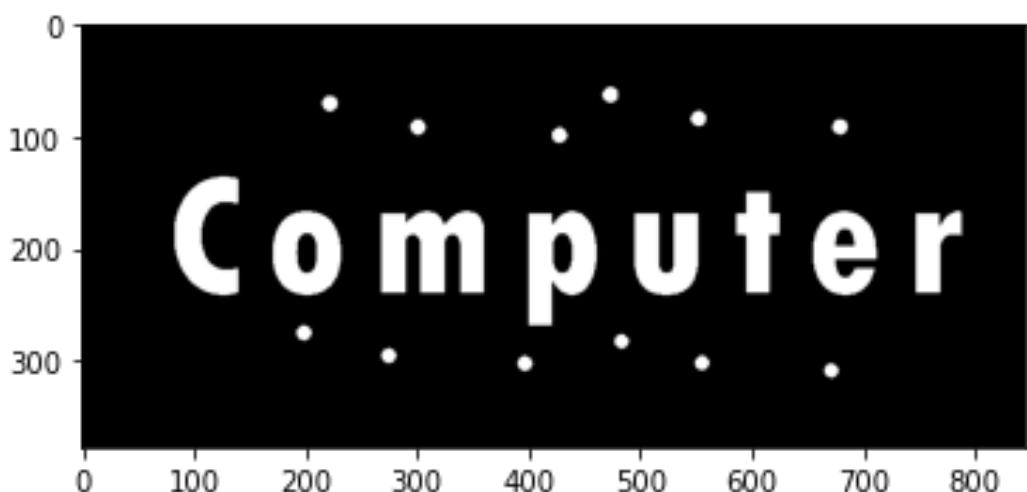
主要用于清除噪点

```
img = cv2.imread('./test_imgs/cv.png')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.imshow(gray, cmap='gray')

><matplotlib.image.AxesImage at 0x7fad7e945550>
```



```
kernel = np.ones((10,10), dtype=np.int8)

opening1 = cv2.morphologyEx(gray.copy(), cv2.MORPH_OPEN, kernel)

kernel = np.ones((12,12), dtype=np.int8)

opening2 = cv2.morphologyEx(gray.copy(), cv2.MORPH_OPEN, kernel)

kernel = np.ones((15,15), dtype=np.int8)
```

```
opening3 = cv2.morphologyEx(gray.copy(),cv2.MORPH_OPEN,kernel)
```

```
fig,(ax1,ax2,ax3,ax4) = plt.subplots(1,4,figsize=(20,8),sharex=True,sharey=True)

ax1.axis('off')

ax1.imshow(gray.copy(),cmap='gray')

ax1.set_title('orginal image')


ax2.axis('off')

ax2.imshow(opening1,cmap='gray')

ax2.set_title('10x10')


ax3.axis('off')

ax3.imshow(opening2,cmap='gray')

ax3.set_title('12x12')


ax4.axis('off')

ax4.imshow(opening3,cmap='gray')

ax4.set_title('15x15')

>Text(0.5, 1.0, '15x15')
```

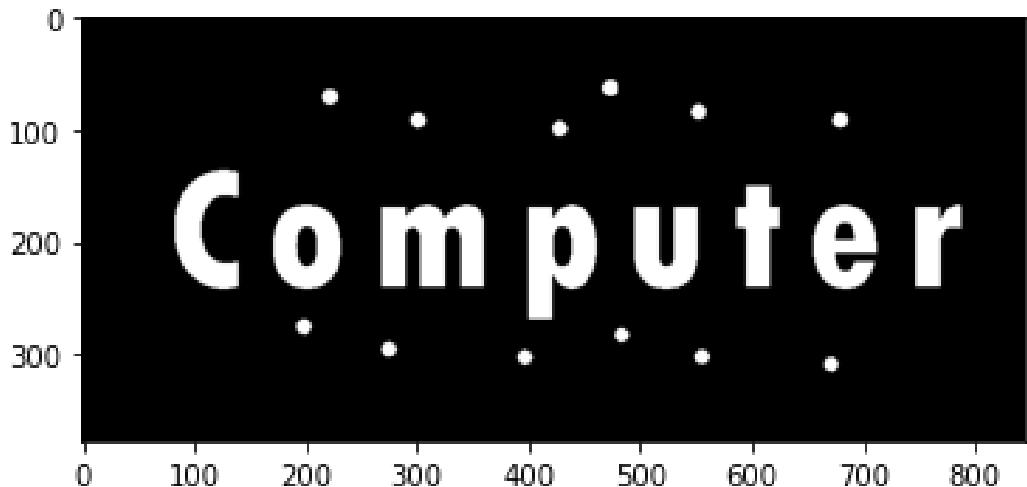


```
# closing, 闭合 cv2.morphologyEx(图片, 模型: 闭合, 内核大小)
```

```
# 先膨胀再侵蚀, 主要用于闭合主体内的小洞, 或者一些黑色的点
```

```
plt.imshow(gray.copy(),cmap='gray')
```

```
><matplotlib.image.AxesImage at 0x7fad7fa48890>
```

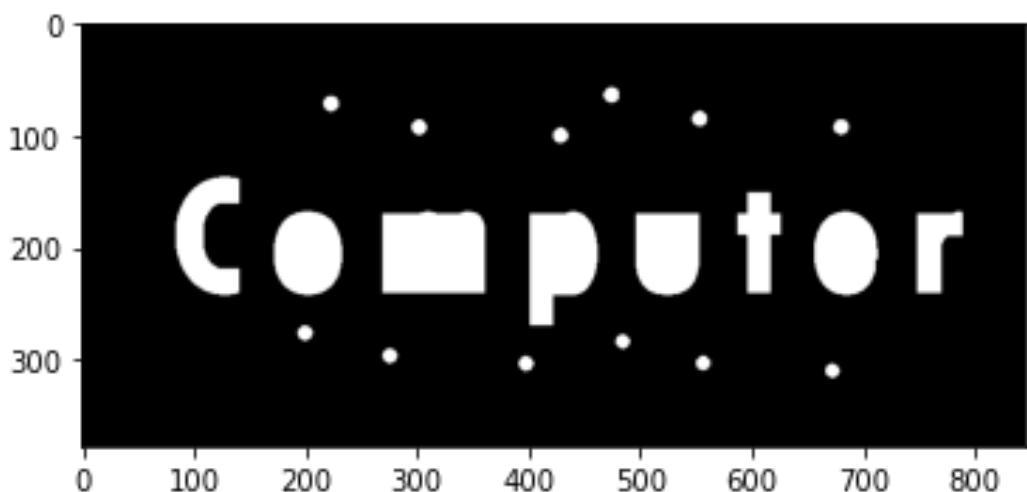


```
kernel = np.ones((20,20),dtype=np.int8)

closing1 = cv2.morphologyEx(gray.copy(),cv2.MORPH_CLOSE,kernel)

plt.imshow(closing1,cmap='gray')

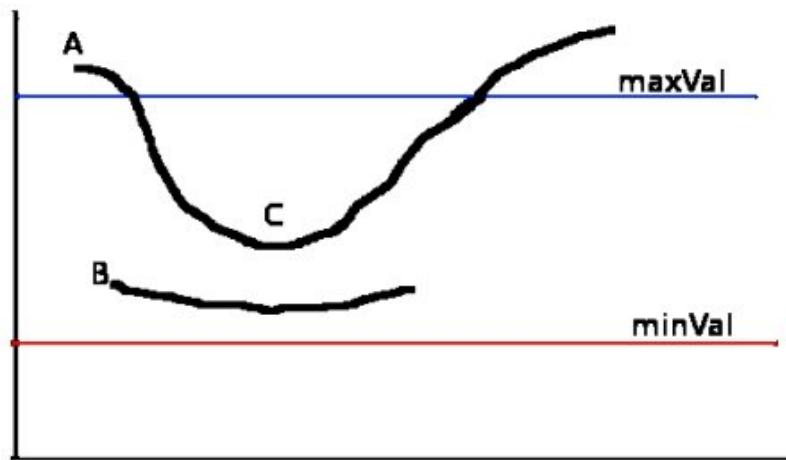
><matplotlib.image.AxesImage at 0x7fad7fc9990>
```



Canny 边缘检测算法

```
# canny 边缘检测算法是一种流行的边缘检测方法，由 John F. Canny in 发明
# 算法具体的推导超出本系列课程应用，但是他主要的步骤如下
# 1.高斯模糊降噪
# 2.使用 Sobel filter 计算图片像素梯度
# 3.NMS 非最大值抑制计算局部最大值
# 4.Hysteresis thresholding 滞后阈值法过滤
```

其中 canny 的两个参数 T_lower、T_upper 就是这里的



其实一般我们在使用时，要注意的就是这两个值得选择：

A 高于阈值 maxVal 所以是真正的边界点，C 虽然低于 maxVal 但高于 minVal 并且与 A 相连，所以也被认为是真正的边界点。

而 B 就会被抛弃，因为他不仅低于 maxVal 而且不与真正的边界点相连。

所以选择合适的 maxVal 和 minVal 对于能否得到好的结果非常重要。

#用法: cv2.Canny(图片, minVal, maxVal)

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

img = cv2.imread('./test_imgs/pumpkin.jpg')

img_fixed = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img_fixed)

><matplotlib.image.AxesImage at 0x7fc8de0dded0>
```



```
edges1 = cv2.Canny(img.copy(), 100, 200)  
  
edges2 = cv2.Canny(img.copy(), 50, 200)  
  
edges3 = cv2.Canny(img.copy(), 50, 100)
```

#最大值不变，最小值越小，图片细节越多；

#最小值不变，最大值越大，图片细节越多。

```
fig, (ax2, ax3, ax4) = plt.subplots(1, 3, figsize=(20, 8), sharex=True, sharey=True)  
  
ax2.axis('off')  
  
ax2.imshow(edges1, cmap='gray')  
  
  
ax3.axis('off')  
  
ax3.imshow(edges2, cmap='gray')  
  
  
ax4.axis('off')  
  
ax4.imshow(edges3, cmap='gray')  
  
><matplotlib.image.AxesImage at 0x7fc8db94fcd0>
```



检测书法文字

步骤：

1、读取图片，灰度、二值化处理

2、侵蚀去噪点

3、膨胀连接

4、闭合孔洞

5、边缘检测

6、画检测框

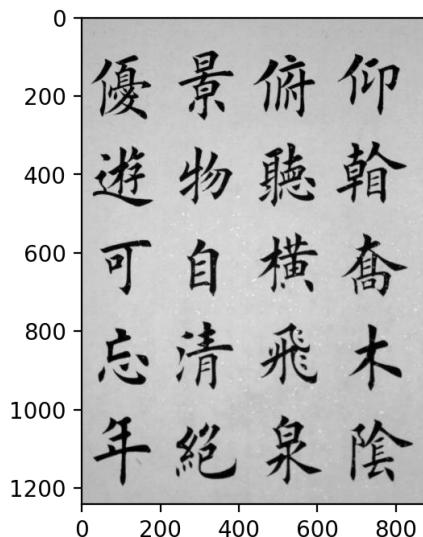
```
import cv2  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
# %matplotlib inline  
  
plt.rcParams['figure.dpi']=200 #控制显示图片的画面大小
```

读取

```
img = cv2.imread('./test_imgs/shufa.jpg')  
  
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

显示灰度图

```
plt.imshow(gray, cmap='gray')  
  
><matplotlib.image.AxesImage at 0x7fe1cb59ae10>
```



二值化（理解为非此即彼）

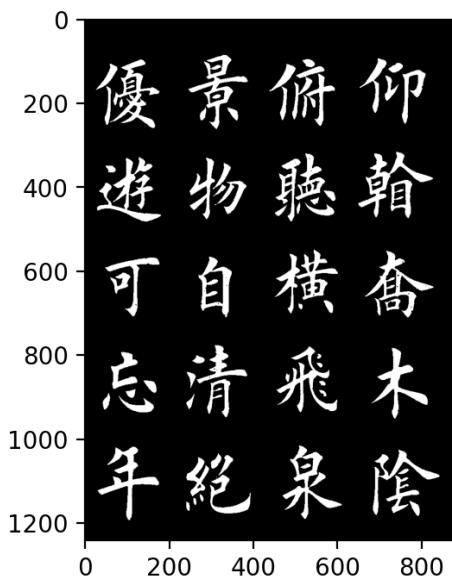
用法: `cv2.threshold(图片, 比较值/阈值, 目标值, 模式)`

作用: 将画面像素与比较阈值对比, 小于它则设为 0 (黑色), 大于它则设为目标值

```
r,black_img = cv2.threshold(gray,100,255, cv2.THRESH_BINARY_INV)

plt.imshow(black_img,cmap='gray')

><matplotlib.image.AxesImage at 0x7fe1cc145d50>
```

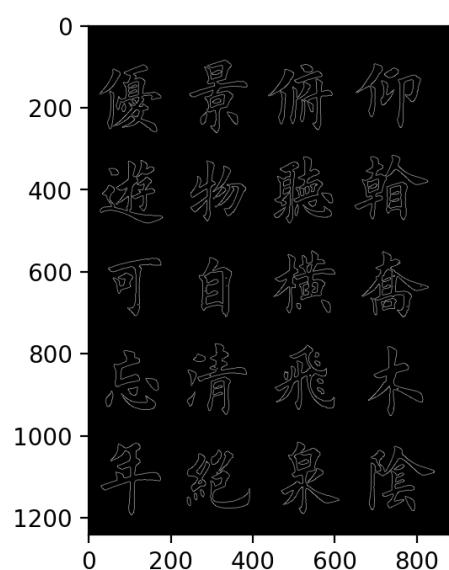


边缘检测

```
edges = cv2.Canny(black_img,30,200)

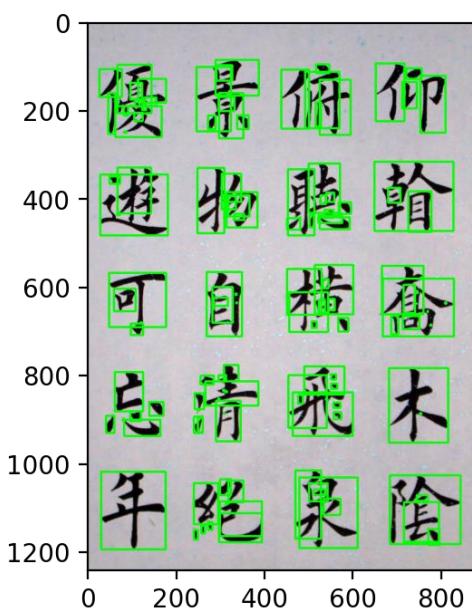
plt.imshow(edges,cmap='gray')

><matplotlib.image.AxesImage at 0x7fe1cd6c5190>
```



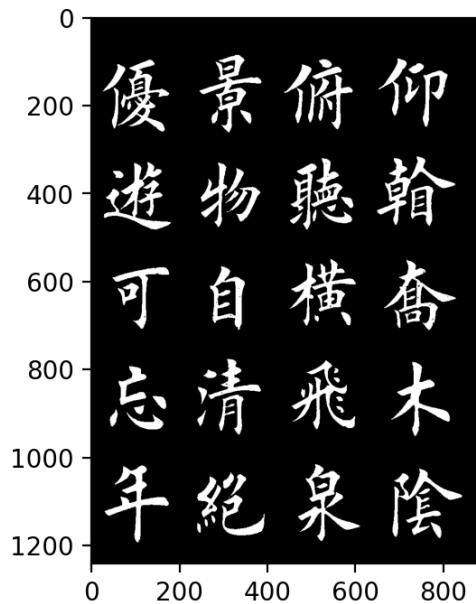
```
# 找轮廓 cv2.findContours(图片, cv2.RETR_EXTERNAL,  
cv2.CHAIN_APPROX_NONE)
```

```
coutours,h = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)  
  
img_copy = img.copy()  
  
for c in coutours:  
  
    x,y,w,h = cv2.boundingRect(c)  
  
    cv2.rectangle(img_copy, (x,y), (x+w,y+h), (0,255,0), 3) #绘制矩形, 用彩图, 因为要画绿色  
  
plt.imshow(img_copy)  
  
><matplotlib.image.AxesImage at 0x7fe1cd82e190>
```



#可以看到, 检测出来的边缘跟我们想要的整个文字的边缘的结果差距还是相当大的, 因此我们就需要使用形态学变换, 先将这些有笔画断开的文字变成连续的整体。

```
plt.imshow(black_img, cmap='gray') #使用二值化后的图片进行处理  
  
><matplotlib.image.AxesImage at 0x7fe1cd7ba850>
```



形态学变化

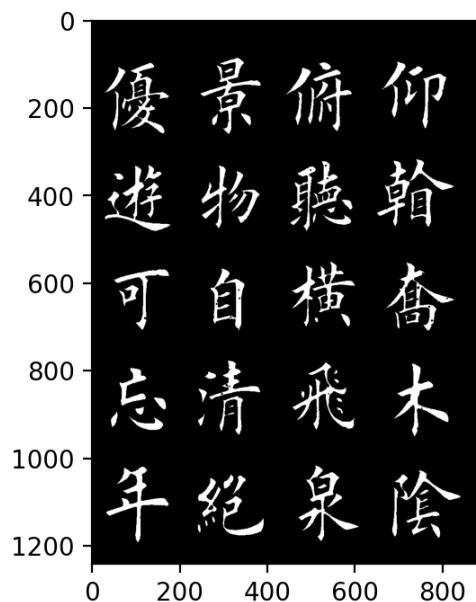
先侵蚀，去除噪点

```
kernel = np.ones((3,3), dtype=np.int8)

erosion1=cv2.erode(black_img,kernel,iterations=1)

plt.imshow(erosion1,cmap='gray')

><matplotlib.image.AxesImage at 0x7fe1cd8cd6d0>
```



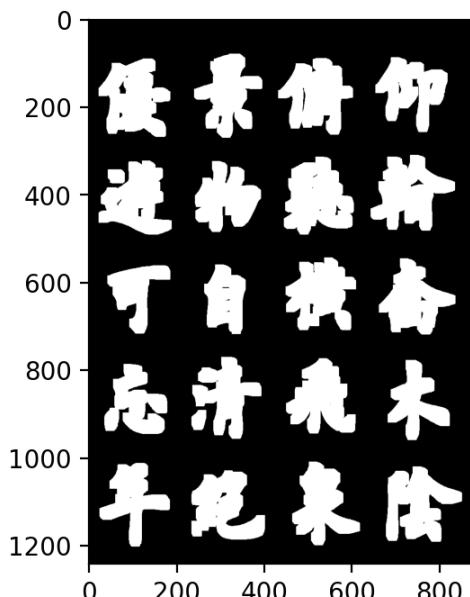
再膨胀

```
kernel = np.ones((10,10), dtype=np.int8)
```

```
dilation = cv2.dilate(erosion1,kernel,iterations=2)

plt.imshow(dilation,cmap='gray')

><matplotlib.image.AxesImage at 0x7fe1cde49690>
```



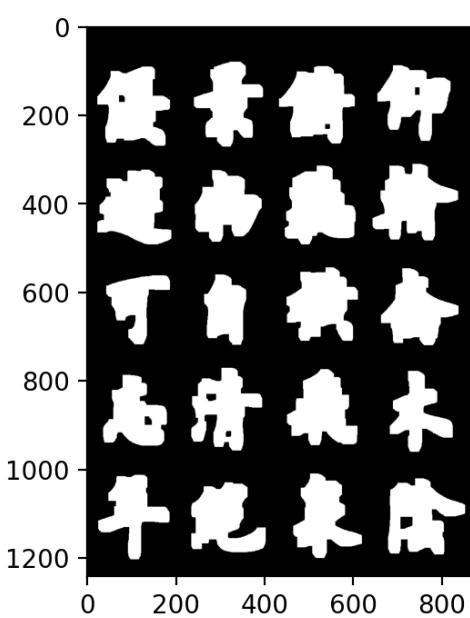
闭合

```
kernel = np.ones((10,10),dtype=np.int8)

closing = cv2.morphologyEx(dilation,cv2.MORPH_CLOSE,kernel)

plt.imshow(closing,cmap='gray')

><matplotlib.image.AxesImage at 0x7fe1cddd7450>
```

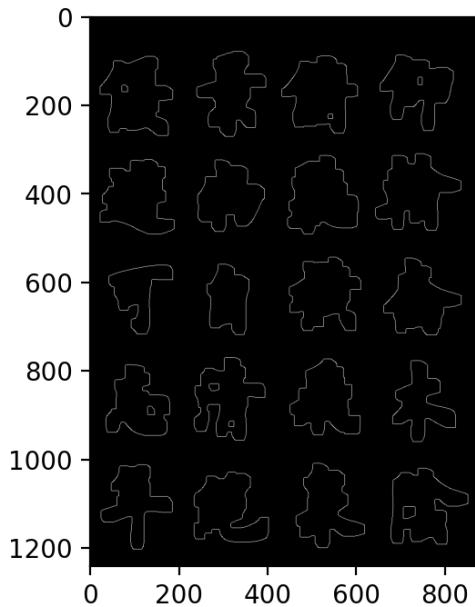


边缘检测

```
edges1 = cv2.Canny(closing, 30, 200)

plt.imshow(edges1, cmap='gray')

><matplotlib.image.AxesImage at 0x7fe1cc1370d0>
```



找轮廓

```
coutours1, h = cv2.findContours(edges1, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

img_copy = img.copy()

for c in coutours1:

    x, y, w, h = cv2.boundingRect(c) #可以增加矩形大小判断，过滤太小的矩形

    cv2.rectangle(img_copy, (x, y), (x+w, y+h), (0, 255, 0), 3)

plt.imshow(img_copy)

><matplotlib.image.AxesImage at 0x7fe1d0114150>
```



图片分类（HOG+SVM）

HOG：方向梯度直方图

方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。HOG 特征通过计算和统计图像局部区域的梯度方向直方图来构成特征。

1、主要思想：

特征描述符就是通过提取图像的有用信息，并且丢弃无关信息来简化图像的表示。

HOG 特征描述符可以将 3 通道的彩色图像转换成一定长度的特征向量。

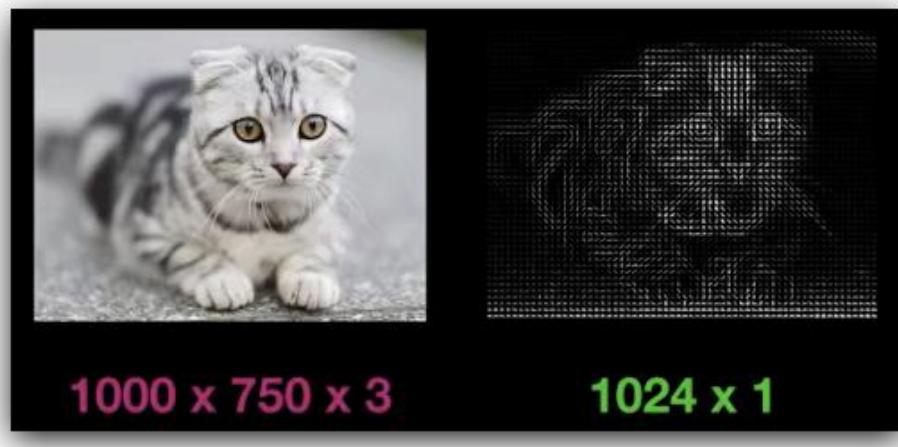
那么我们就需要定义什么是“有用的”，什么是“无关的”。这里的“有用”，是指对于什么目的有用，显然特征向量对于观察图像是没有用的，但是它对于像图像识别和目标检测这样的任务非常有用。当将这些特征向量输入到类似支持向量机（SVM）这样的图像分类算法中时，会得到较好的结果。

那什么样的“特征”对分类任务是有用，比如我们想检测出马路上的车道线，那么我们可以通过边缘检测来找到这些车道线，在这种情况下，边缘信息就是“有用的”，而颜色信息是无关的。

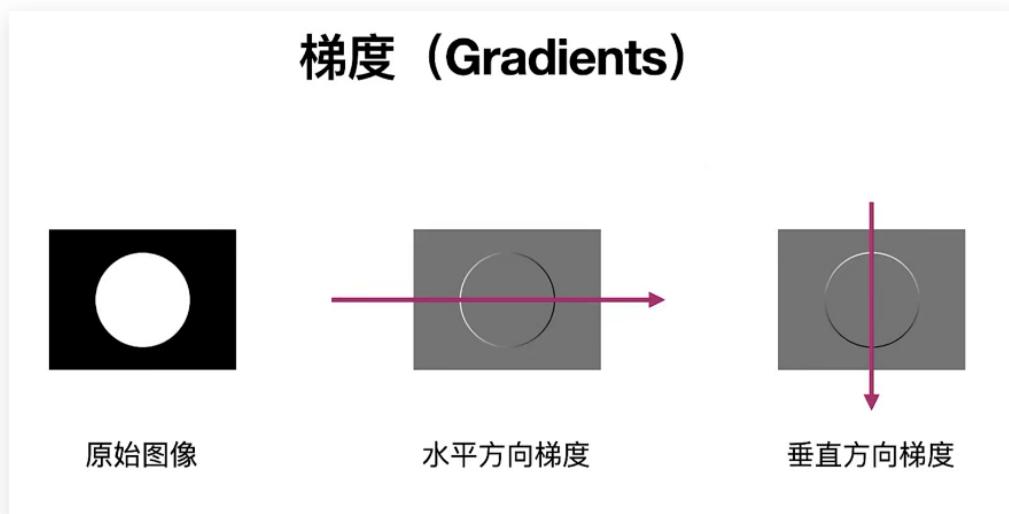
在 HOG 特征描述符中，梯度方向的分布，也就是梯度方向的直方图被视作特征。图像

的梯度(x 和 y 导数)非常有用，因为边缘和拐角(强度突变的区域)周围的梯度幅度很大，并且边缘和拐角比平坦区域包含更多关于物体形状的信息。

方向梯度直方图(HOG)特征描述符常和线性支持向量机(SVM)配合使用，用于训练高精度的目标分类器。

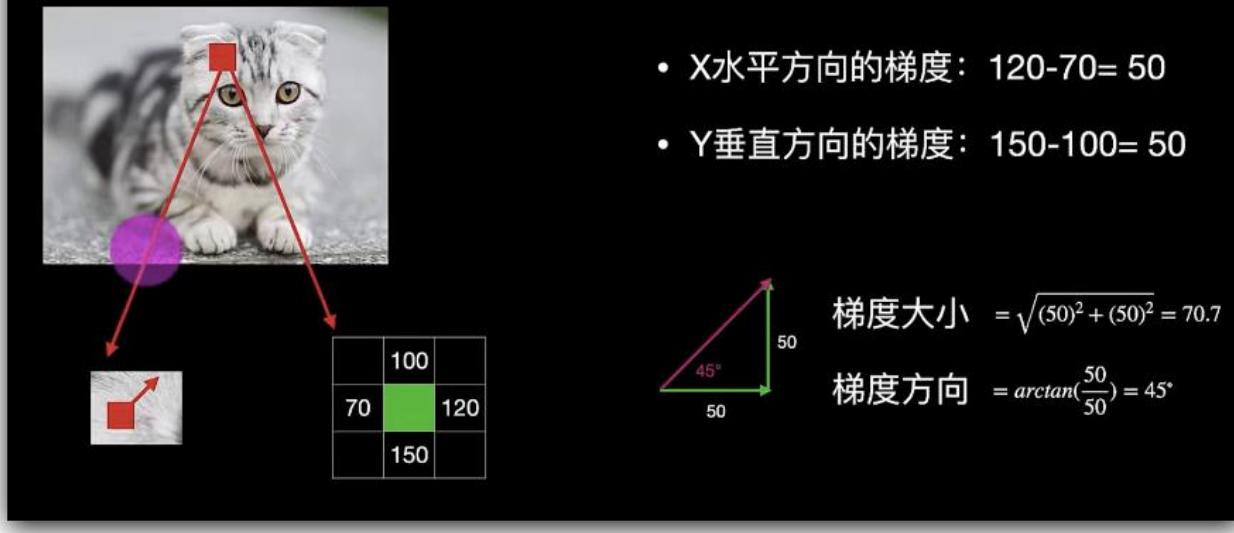


梯度



图像梯度计算的是图像变化的速度。对于图像的边缘部分，其灰度值变化较大，梯度值也较大；对于图像中比较平滑的部分，其灰度值变化较小，相应的梯度值也小。一般情况下，图像梯度计算的是图像的边缘信息。

梯度大小和方向



对于彩色图像，先对三通道颜色值分别计算梯度，然后取梯度值最大的那个作为该像素的梯度。

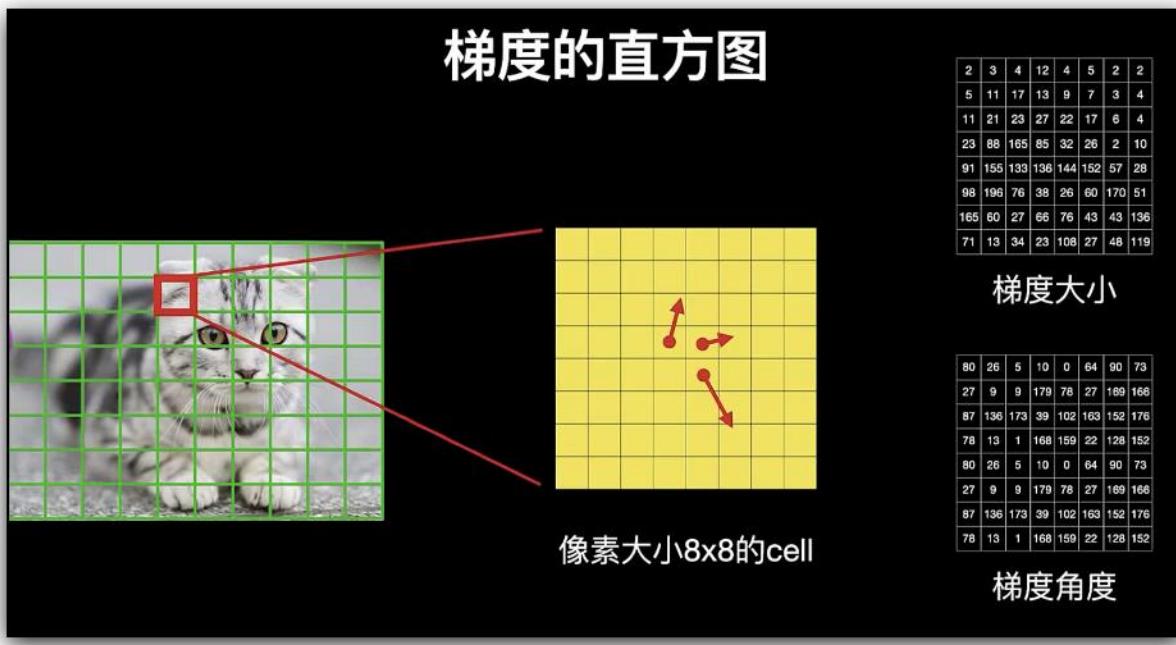
计算梯度直方图

在这一步，我们先把整个图像划分为若干个 8×8 的小单元，称为 **cell**，并计算每个 **cell** 的梯度直方图。这个 **cell** 的尺寸也可以是其他值，根据具体的特征而定。

为什么我们要把图像分成若干个 8×8 的小单元？

这是因为对于一整张梯度图，其中的有效特征是非常稀疏的，不但运算量大，而且效果可能还不好。于是我们就使用特征描述符来表示一个更紧凑的特征。一个 8×8 的小单元就包含了 $8\times 8\times 2 = 128$ 个值，因为每个像素包括梯度的大小和方向。

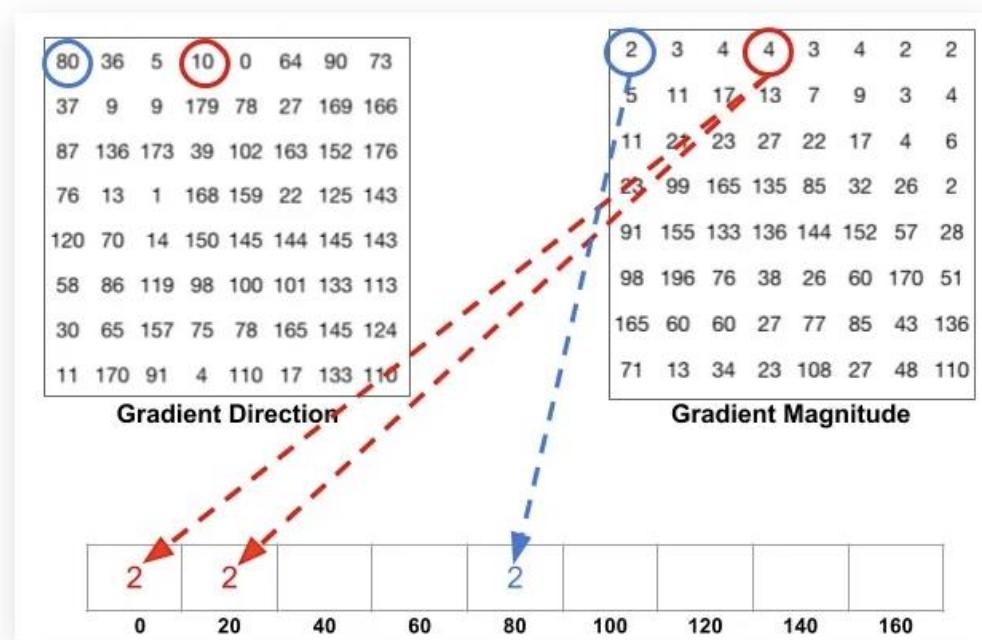
现在我们要把这个 8×8 的小单元用长度为 9 的数组来表示，这个数组就是梯度直方图。这种表示方法不仅使得特征更加紧凑，而且对单个像素值的变化不敏感，也就是能够抗噪声干扰。



中间那张图中的箭头表示梯度，箭头方向表示梯度方向，箭头长度表示梯度大小。

右图是 8×8 的 cell 中表示梯度的原始数字，注意角度的范围介于 0 到 180 度之间，而不是 0 到 360 度，这被称为“无符号”梯度，因为两个完全相反的方向被认为是相同的。

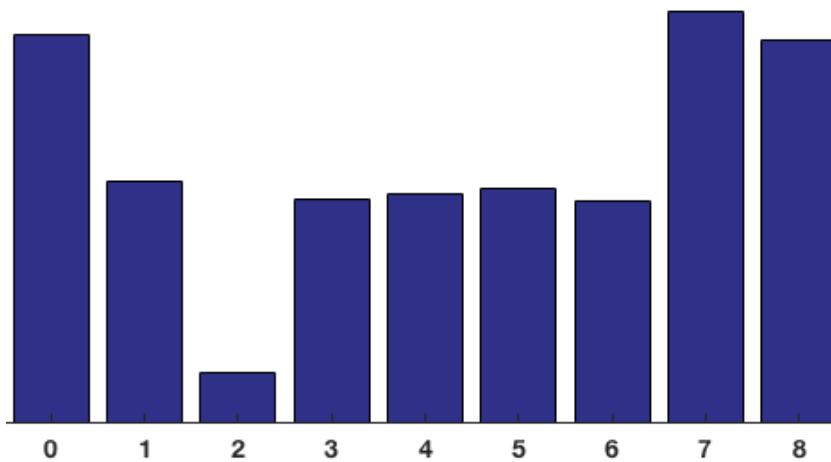
现在我们来计算 cell 中像素的梯度直方图，先将角度范围分成 9 份，也就是 9 bins，每 20° 为一个单元，也就是这些像素可以根据角度分为 9 组。将每一份中所有像素对应的梯度值进行累加，可以得到 9 个数值。直方图就是由这 9 个数值组成的数组，对应于角度 0、 20° 、 40° 、 60° ... 160° 。



比如上面方向图中蓝圈包围的像素，角度为 80 度，这个像素对应的幅值为 2，所以在直方图 80 度对应的 bin 加上 2。红圈包围的像素，角度为 10 度，介于 0 度和 20 度之间，其幅值为 4，那么这个梯度值就被按比例分给 0 度和 20 度对应的 bin，也就是各加上 2。

还有一个细节需要注意，如果某个像素的梯度角度大于 160 度，也就是在 160 度到 180 度之间，那么把这个像素对应的梯度值按比例分给 0 度和 160 度对应的 bin。

将这 8×8 的 cell 中所有像素的梯度值加到各自角度对应的 bin 中，就形成了长度为 9 的直方图：

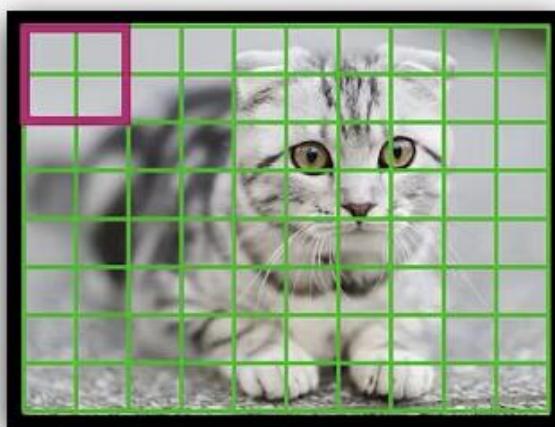


可以看到直方图中，0 度和 160 附近有很大的权重，说明了大多数像素的梯度向上或者向下，也就是这个 cell 是个横向边缘。

现在我们就可以用这 9 个数的梯度直方图来代替原来很大的三维矩阵，即代替了 $8 \times 8 \times 2$ 个值。

Block 归一化

HOG 将 8×8 的一个区域作为一个 cell，再以 2×2 个 cell 作为一组，称为 block。由于每个 cell 有 9 个值， 2×2 个 cell 则有 36 个值，HOG 是通过滑动窗口的方式来得到 block 的，如下图所示：



在前面的步骤中，我们基于图像的梯度对每个 `cell` 创建了一个直方图。

但是图像的梯度对整体光照非常敏感，比如通过将所有像素值除以 2 来使图像变暗，那么梯度幅值将减小一半，因此直方图中的值也将减小一半。理想情况下，我们希望我们的特征描述符不会受到光照变化的影响，那么我们就需要将直方图“归一化”。

$$\text{向量 } \vec{a} = [64, 32, 16]$$

$$\text{向量 } \vec{b} = 2 \times \vec{a} = [128, 64, 32]$$

$$\text{向量长度 } L_{\vec{a}} = \sqrt{64^2 + 32^2 + 16^2} = 73.32$$

$$\text{向量长度 } L_{\vec{b}} = \sqrt{128^2 + 64^2 + 32^2} = 146.64$$

$$\text{向量归一化后 } \frac{[64, 32, 16]}{L_{\vec{a}}} = [0.87, 0.43, 0.22]$$

$$\text{向量归一化后 } \frac{[128, 64, 32]}{L_{\vec{b}}} = [0.87, 0.43, 0.22]$$

在说明如何归一化直方图之前，先看看长度为 3 的向量是如何归一化的。

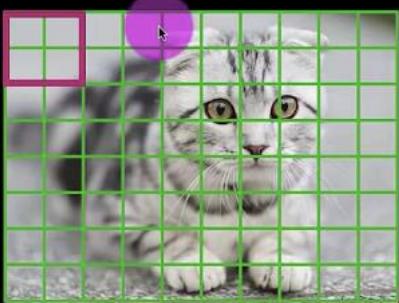
假设我们有一个向量 $[128, 64, 32]$ ，向量的长度为 $\sqrt{128^2 + 64^2 + 32^2} = 146.64$ ，这叫做向量的 L2 范数。将这个向量的每个元素除以 146.64 就得到了归一化向量 $[0.87, 0.43, 0.22]$ 。

现在有一个新向量，是第一个向量的 2 倍 $[128 \times 2, 64 \times 2, 32 \times 2]$ ，也就是 $[256, 128, 64]$ ，我们将这个向量进行归一化，你可以看到归一化后的结果与第一个向量归一化后的结果相同。所以，对向量进行归一化可以消除整体光照的影响。

知道了如何归一化，现在来对 `block` 的梯度直方图进行归一化（注意不是 `cell`），一个 `block` 有 4 个直方图，将这 4 个直方图拼接成长度为 36 的向量，然后对这个向量进行归一化。

因为使用的是滑动窗口，滑动步长为 8 个像素，所以每滑动一次，就在这个窗口上进行归一化计算得到长度为 36 的向量，并重复这个过程。

总结



1. 将图像分成像素 8×8 的cells，整图图分成了 10×8 个cells；
2. 计算每个Cell内的各个像素的梯度大小和方向，每个Cell的特征有 $8\times 8\times 2=128$ 个值；
3. 计算每个Cell的直方图，每个Cell的特征由128个值降到9个值；
4. 用 2×2 个Cell大小的Block进行归一化，每个Block特征有 $9\times 4=36$ 个值；
5. Block使用滑动窗口，所以整图共有 $(10-1)\times(8-1)=63$ 个Block，所以整图的HOG特征有 $63\times 36=2268$ 个值；
6. 这个HOG特征送入SVM等分类器就能训练分类模型。

代码实现

```
# 导入必要包
```

```
import cv2  
  
import matplotlib.pyplot as plt  
  
import numpy as np  
  
%matplotlib inline
```

```
# 安装 skimage (用于计算 HOG)
```

```
# Conda 未换源: conda install -c anaconda scikit-image
```

```
# Conda 换源: conda install scikit-image
```

```
# pip: pip install scikit-image
```

```
# 导入 skimage
```

```
from skimage.feature import hog  
  
from skimage import data, exposure  
  
  
img = cv2.imread('./test_imgs/cat1.jpg')  
  
img = cv2.resize(img, (500, 370))
```

```
img_fixed = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

<https://scikit-image.org/docs/dev/api/skimage.feature.html?highlight=hog#skimage.feature.hog>

```
fd, hog_image = hog(image=img_gray, orientations=9, pixels_per_cell=(8, 8),
                     cells_per_block=(2, 2), visualize=True)
```

image: 输入图像

orientations: 把 180 度分成几份, bin 的数量

pixels_per_cell : 元组形式, 一个 Cell 内的像素大小

cells_per_block: 元组形式, 一个 Block 内的 Cell 大小

visualize: 是否需要可视化, 如果 True, hog 会返回 numpy 图像

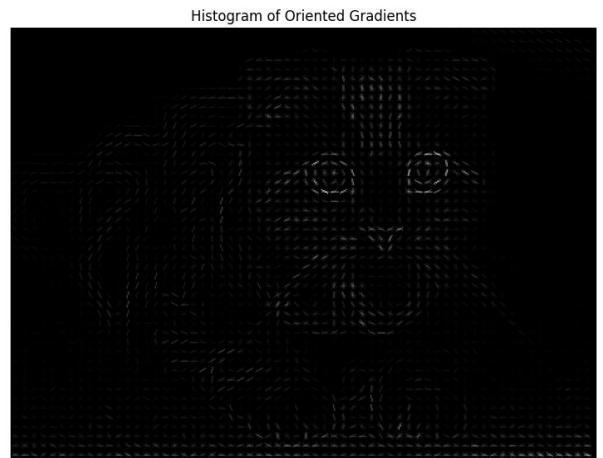
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10), sharex=True, sharey=True)

ax1.axis('off')

ax1.imshow(img_fixed)

ax1.set_title('Input image')

plt.show()
```



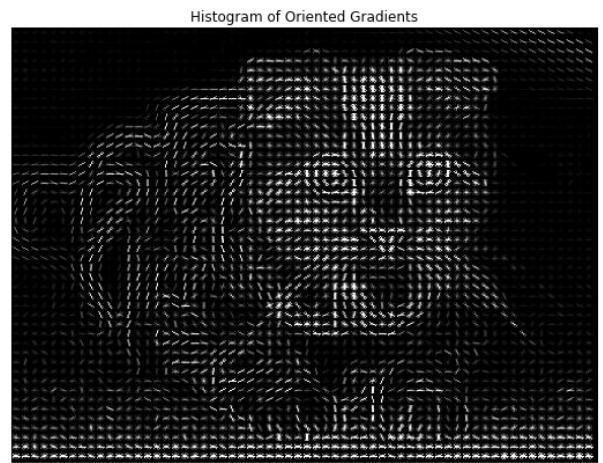
```
# Rescale histogram for better display
```

```
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
```

```
ax2.axis('off')

ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
```

```
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```



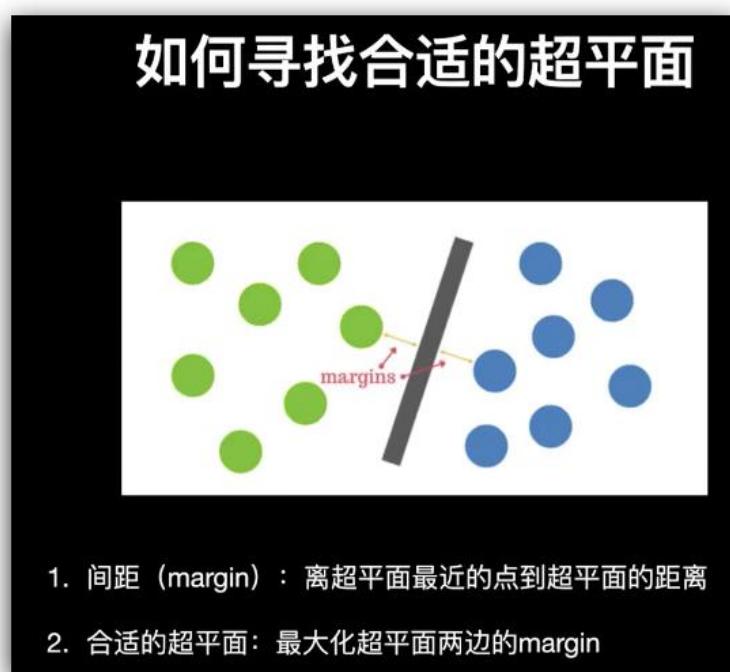
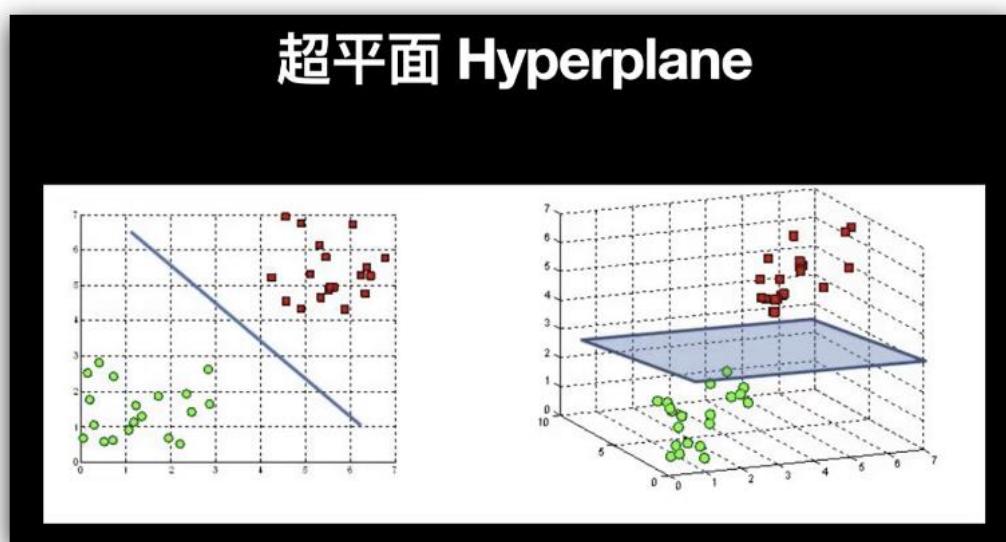
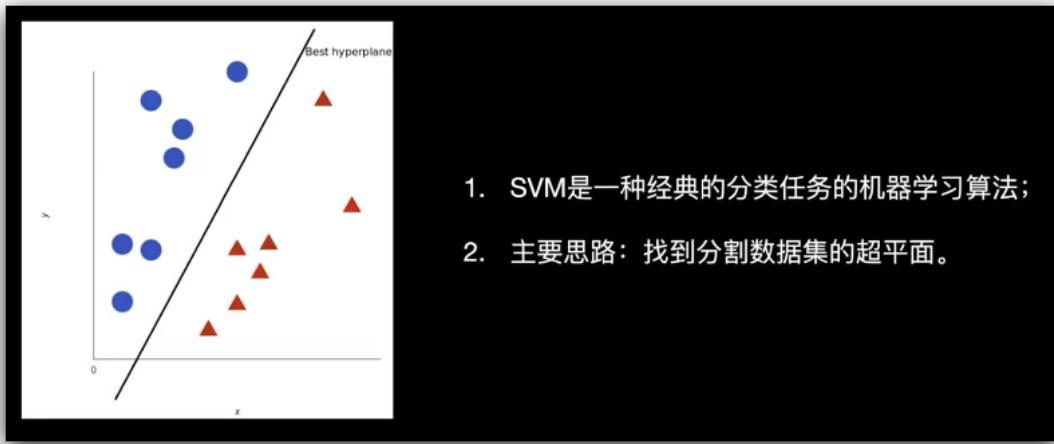
查看一下 HOG 特征大小

```
fd.shape
>(98820,)
```

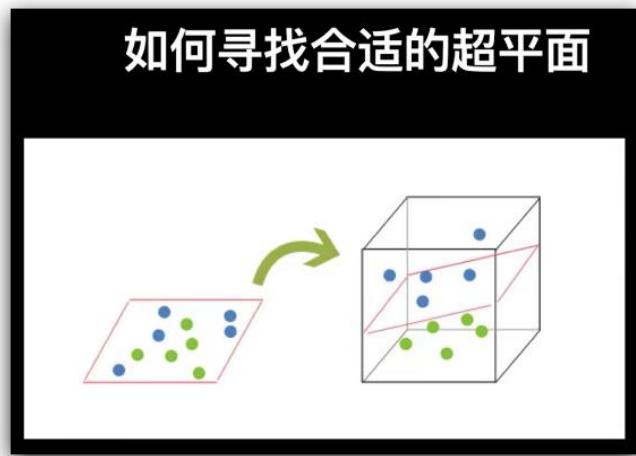
```
fd
>array([0.28139531, 0.          , 0.19306082, ..., 0.27718241, 0.13895444,
       0.12632074])
```

SVM（支持向量机）

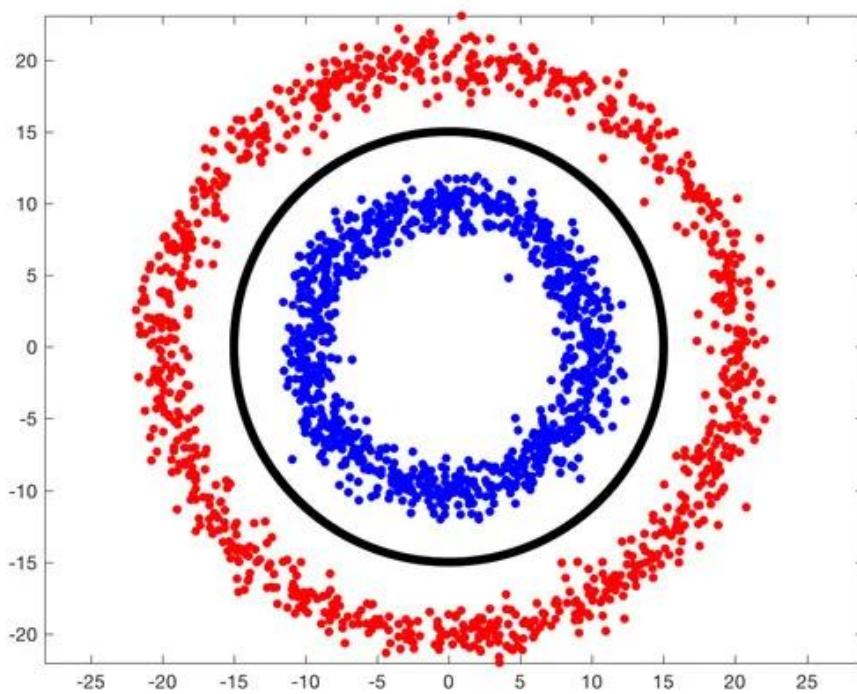
如果在 n 维空间中有点，并且点上有类标签，线性支持向量机将使用平面来划分空间，使得不同的类位于平面的不同侧面。在下图中，我们用红点和蓝点表示两个类。如果将这些数据输入到线性支持向量机中，它将很容易地通过找到清楚地分隔两个类的行来构建分类器。有很多行可以分隔这些数据。SVM 选择在任一类的最大距离数据点处的那个。



我们的图像描述符不是二维空间中的点，而是 81 维空间中的点，因为它们由一个 81×1 的矢量表示。附加在这些点上的类标签是图像中包含的数字，即 0,1,2, ... 9。支持向量机将在一个高维空间中寻找超平面来进行分类，而不是在二维空间中寻找直线。



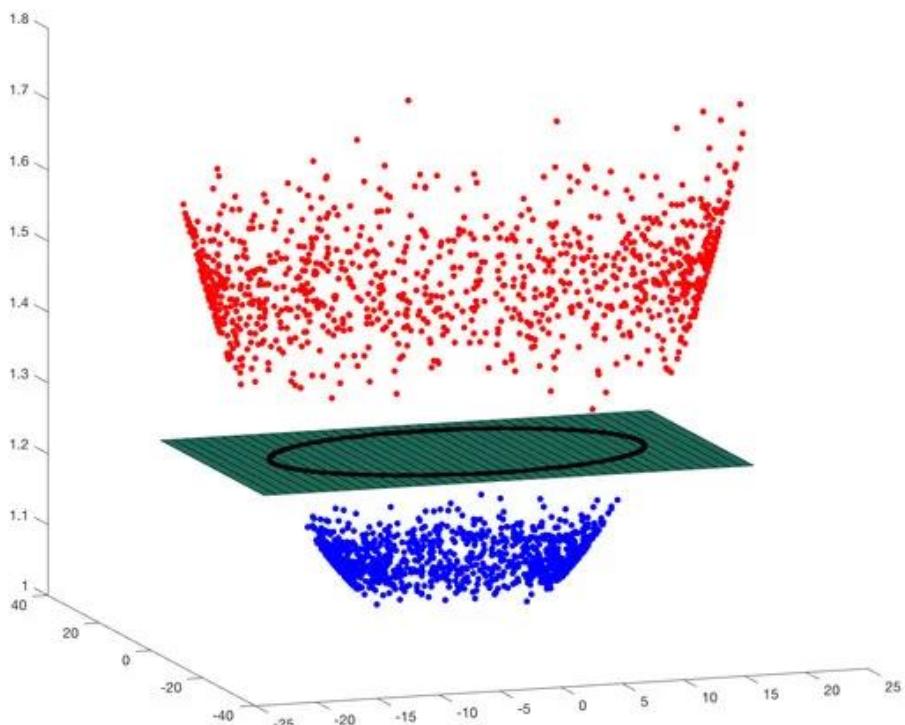
下图显示了使用不可线性分离的红点和蓝点的两个类。不能在平面上画一条线来分隔这两个类。一个好的分类器，用黑线表示，更像是一个圆。



在现实生活中，数据是混乱的，不能线性分离。为了实现使用 SVM 进行分类，您可以使用一种称为内核技巧的技术。在我们的例子中，红点和蓝点位于一个 2D 平面上。让我们使用以下方程为所有数据点添加第三维。

$$z = e^{-\gamma(x^2+y^2)}$$

如果你曾经听到人们用“径向基核函数”这个花哨的词来形容 Gaussian Kernel，他们只不过是在谈论上面的等式。径向基函数是一个简单的实值函数，它只取决于与原点的距离（即只取决于 $\sqrt{x^2 + y^2}$ ）。高斯核是指上述方程的高斯形式。更一般地说，RBF 可以有不同种类的内核。所以，我们根据另外两个维度的数据创造了一个第三维度。下图显示了这个三维 (x, y, z) 数据。我们可以看到它是可以被包含黑色圆圈的平面分开的！



参数 γ 控制第三维中的数据拉伸。它有助于分类，但它也扭曲了数据。像金发姑娘一样，你必须选择这个参数“恰到好处”。这是人们在训练支持向量机时选择的两个重要参数之一。

SVM 对多类数据的分类

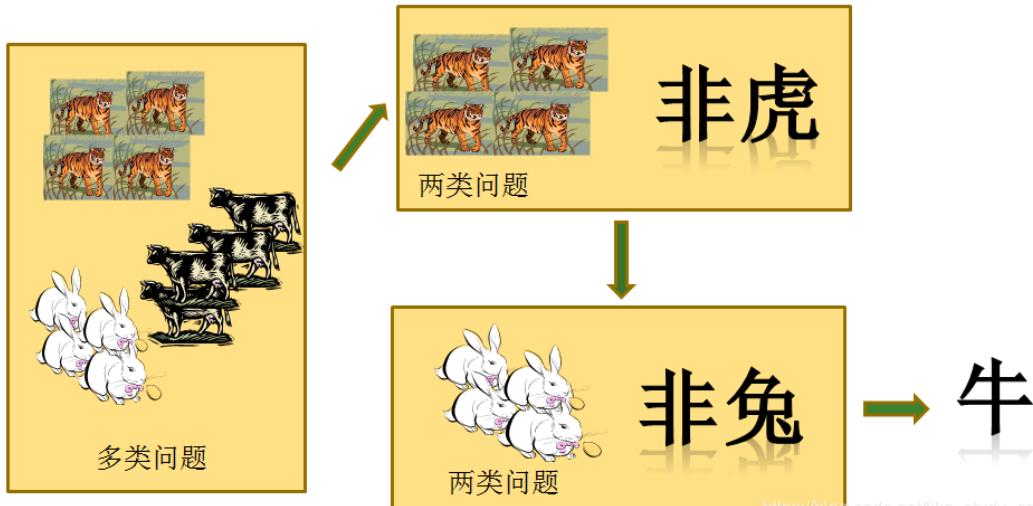
OVR (one-versus-rest)

一对多法，训练时依次把某个类别的样本归为一类，其他剩余的样本归为另一类，这样 k 个类别的样本就构造出了 k 个 SVM。分类时将未知样本分类为具有最大分类函数值的那类。

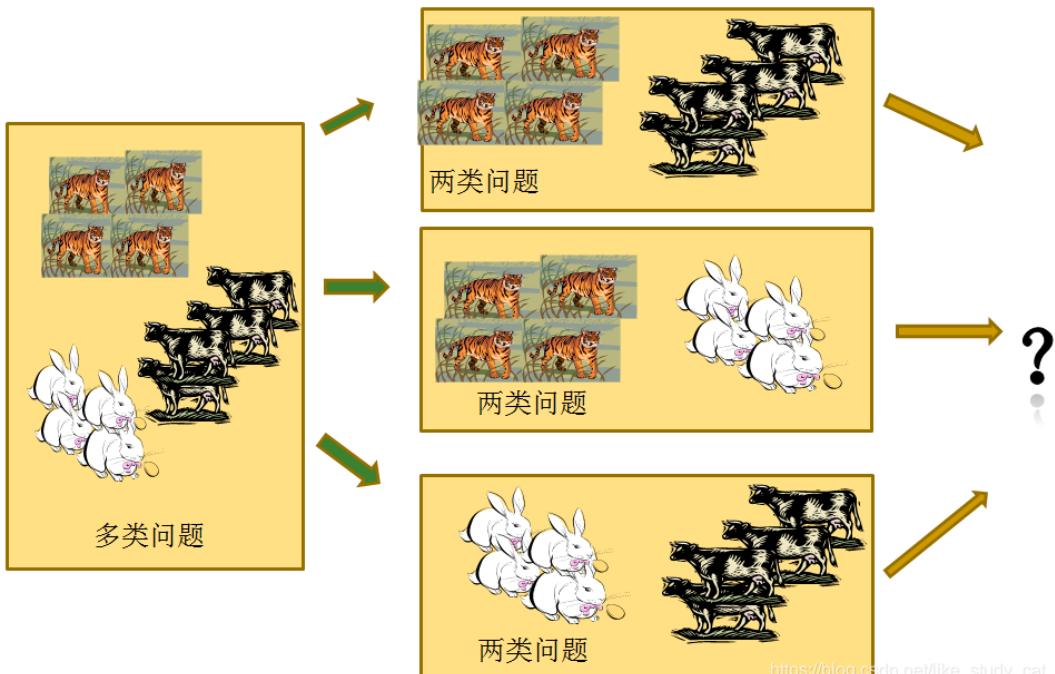
OVO (one-versus-one)

一对一法，在任意两类样本之间设计一个 SVM，因此 k 个类别的样本就需要设计 $k(k-1)/2$ 个 SVM。当对一个未知样本进行分类时，最后得票最多的类别即为该未知样本的类别。

OVR：将每一次的一个类作为正例，其余作为反例，总共训练 N 个分类器。测试的时候若仅有一个分类器预测为正的类别则对应的类别标记作为最终分类结果，若有多个分类器预测为正类，则选择置信度最大的类别作为最终分类结果。



OVO：给定数据集 D 这里有 N 个类别，这种情况下就是将这些类别两两配对，从而产生 $N(N-1)/2$ 个二分类任务，在测试的时候把样本交给这些分类器，然后进行投票。



代码实现

```
#pip install scikit-learn  
#conda install -c anaconda scikit-learn  
  
#从 sklearn 导入 SVM 包  
from sklearn import svm  
  
#先看二分类  
#训练样本特征，有两个样本：[0, 0]和[1, 1]  
X = [[0, 0], [1, 1]]  
  
#训练样本类别标签，样本[0, 0]的标签为 0，样本[1, 1]的标签为 1  
Y = [0, 1]  
clf = svm.SVC() #构造分类器  
  
#训练  
clf.fit(X, Y)  
  
#测试数据样本  
test = [[2, 2]]
```

#预测

```
clf.predict(test)  
>array([1]) #预测结果对应标签 1
```

#多分类问题

#训练样本特征

```
X = [[0], [1], [2], [3], [4]]
```

#训练样本类别标签

```
Y = [0, 1, 2, 3, 4]
```

#测试数据集

```
test = [[1]]
```

#选择一对一策略

```
clf = svm.SVC(decision_function_shape='ovo')
```

#训练

```
clf.fit(X, Y)
```

#查看投票函数

```
dec = clf.decision_function(test)
```

#查看筛选函数的大小，可以看到是 10，是因为 ovo 策略会设计 $5*4/2=10$ 个分类器，然后找出概率最大的

```
dec.shape
```

```
>(1, 10)
```

#选择一对多策略

```
clf1 = svm.SVC(decision_function_shape='ovr')
```

```
clf1.fit(X, Y)
```

#查看投票函数

```
dec1 = clf1.decision_function(test)
```

#查看筛选函数的大小，可以看到是 5，是因为 ovr 策略会设计 5 个分类器，然后找出概率最大的

```
dec1.shape
```

```
>(1, 5)
```

代码实现

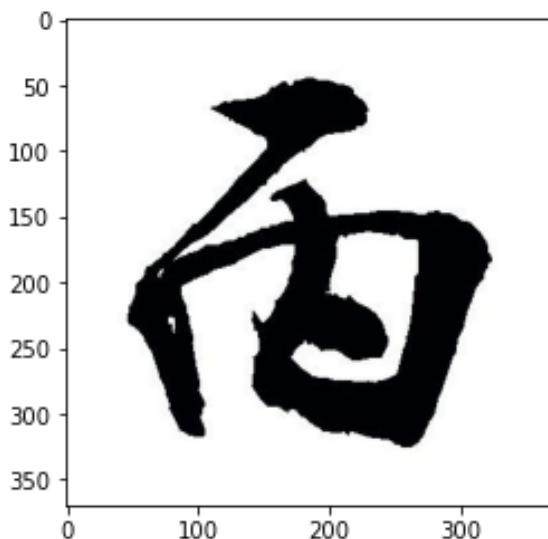
```
#采集图片位置: './images/行书/丙/敬世江  
_5945d30c02c1e30a89de9dc3920a0011adc9aa46.jpg'  
#任务: 将书法图片分成五类 (篆书、隶书、草书、行书、楷书)
```

```
#读取数据  
#提取 HOG 特征  
#送至 SVM 训练  
#评估模型  
#保存模型  
#可视化看一下训练效果  
# 导入必要包
```

```
import cv2  
  
import matplotlib.pyplot as plt  
  
import numpy as np  
  
%matplotlib inline
```

```
def readimg(filename, mode):  
  
    # 解决中文路径问题  
  
    raw_data = np.fromfile(filename, dtype=np.uint8)  
  
    img = cv2.imdecode(raw_data, mode)  
  
    return img  
  
  
test_file_dir = './images/行书/丙/敬世江_5945d30c02c1e30a89de9dc3920a0011adc9aa46.jpg'  
  
img = readimg(test_file_dir, -1)  
  
plt.imshow(img)
```

```
><matplotlib.image.AxesImage at 0x23943faeb50>
```



#计算梯度直方图

```
from skimage.feature import hog

from skimage import data, exposure


def resizeGray(img,new_size):

    img = cv2.resize(img,new_size)

    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    return img


image = resizeGray(img,(200,200))

fd, hog_image = hog(image, orientations=4, pixels_per_cell=(16, 16),

                    cells_per_block=(1, 1), visualize=True)
```

image: 输入图像

orientations: 把 180 度分成几份, bin 的数量

pixels_per_cell : 元组形式, 一个 Cell 内的像素大小

cells_per_block: 元组形式, 一个 Block 内的 Cell 大小

visualize: 是否需要可视化, 如果 True, hog 会返回 numpy 图像

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

ax1.axis('off')

ax1.imshow(image, cmap=plt.cm.gray)

ax1.set_title('Input image')

# Rescale histogram for better display

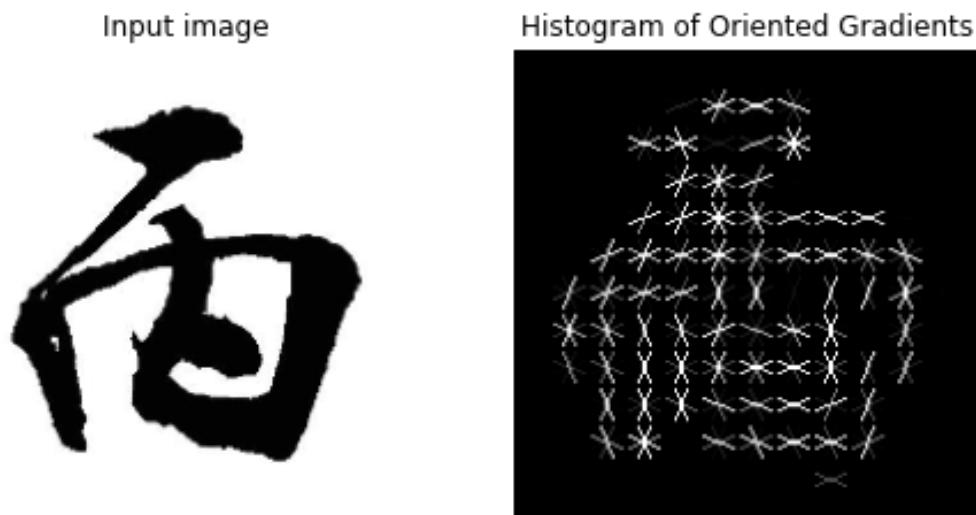
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

ax2.axis('off')

ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)

ax2.set_title('Histogram of Oriented Gradients')

plt.show()
```



#批量读取文件数据

#提取 HOG 特征

```
import os

import glob

import random
```

列出目录下的文件

```
def listdir_nohidden(path):  
    return glob.glob(os.path.join(path, '*')) #*表示通配符, 表全部
```

读取文件

```
def image_reader(file_name,new_size):  
  
    img = readimg(file_name,-1)  
  
    img = resizeGray(img,new_size)  
  
    return img
```

从文件中读取特征和标签

```
def get_file_hog_label_list_from_disk(selectNum=1000):  
  
    char_styles = ['篆书','隶书','草书','行书','楷书']  
  
    # 特征列表和标签列表  
  
    fileFeaturesList = []  
  
    fileLabelList = []  
  
    # 遍历各种风格  
  
    for style in char_styles:  
  
        file_list = glob.glob('./images/'+style+'/*/*')  
  
        print('风格: {}下共有{}张图片\n'.format(style,len(file_list)))  
  
        # 打乱顺序  
  
        random.shuffle(file_list)  
  
        # 挑选固定数量文件  
  
        select_files = file_list[:selectNum]  
  
  
        # 挑选指定数量文件  
  
        for file_item in select_files:  
  
            # 读取文件  
  
            img = image_reader(file_item,(100,100))  
  
            # 提取特征
```

```
features = hog(img, orientations=4,
pixels_per_cell=(6,6),cells_per_block=(2,2))

features = list(features)

fileFeaturesList.append(features)

# 提取标签，索引值对应不同的书法风格

fileLabelList.append(char_styles.index(style))

print('风格: {style}, 共挑选了{num}张图片\n\n'.format(style=style,num =
len(select_files)))

return fileFeaturesList,fileLabelList
```

#开始训练

```
from sklearn.model_selection import train_test_split #将样本分为训练和测试样本

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm #导入 SVM 训练器

from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix #导入融合矩阵，评价模型效果

# 每个字最多挑选 1 个

fileFeaturesList,fileLabelList = get_file_hog_label_list_from_disk(selectNum=1000)

# 将样本分为训练和测试样本，一般 30%设置为测试样本，剩下 70%用作训练

#x_train 训练样本

#x_test 测试样本

#y_train 训练样本标签

#y_test 测试样本标签

x_train,x_test,y_train,y_test = train_test_split(fileFeaturesList,fileLabelList,
test_size=0.25,random_state=42)
```

>风格：篆书下共有 9906 张图片

风格：篆书，共挑选了 1000 张图片

风格：隶书下共有 18337 张图片

风格：隶书，共挑选了 1000 张图片

风格：草书下共有 42917 张图片

风格：草书，共挑选了 1000 张图片

风格：行书下共有 43417 张图片

风格：行书，共挑选了 1000 张图片

风格：楷书下共有 20670 张图片

风格：楷书，共挑选了 1000 张图片

统计各种类别数量

```
from collections import Counter

Counter(fileLabelList)

>Counter({0: 1000, 1: 1000, 2: 1000, 3: 1000, 4: 1000})

len(fileLabelList)

>5000
```

SVM 分类器

```
cls = svm.SVC(kernel='rbf') #默认方法

cls.fit(x_train,y_train)

predictLabels = cls.predict(x_test)

print ("svm acc:%s" % accuracy_score(y_test,predictLabels))

cls1 = svm.SVC(kernel='linear')

cls1.fit(x_train,y_train)

predictLabels = cls1.predict(x_test)

print ("svm 1 acc:%s" % accuracy_score(y_test,predictLabels))
```

```
cls2 = svm.SVC(kernel='poly')

cls2.fit(x_train,y_train)

predictLabels = cls2.predict(x_test)

print ( "svm 2 acc:%s" % accuracy_score(y_test,predictLabels))

>svm acc:0.7568

svm 1 acc:0.6736

svm 2 acc:0.7536
```

KNN

```
neigh = KNeighborsClassifier(n_neighbors=3)

neigh.fit(x_train,y_train)

predictLabels = neigh.predict(x_test)

print ("KNN acc:%s" % accuracy_score(y_test,predictLabels))

>KNN acc:0.6384
```

保存模型

```
from joblib import dump, load

dump(cls, './models/svc.joblib')

dump(neigh, './models/neigh.joblib')

> ['./models/neigh.joblib']
```

调用模型

```
cls = load('./models/svc.joblib')

predictLabels = cls.predict(x_test)

print ( "svm acc:%s" % accuracy_score(y_test,predictLabels))

>svm acc:0.7568
```

#查看模型效果

```
cm = confusion_matrix(y_test, predictLabels)
```

```
cm

>array([[237,     5,     8,     2,    14],
       [   6,  214,     4,     1,    17],
       [   8,    10,  130,    67,    21],
       [   4,     4,    60,  177,    19],
       [   7,    23,    14,    10,  188]])
```

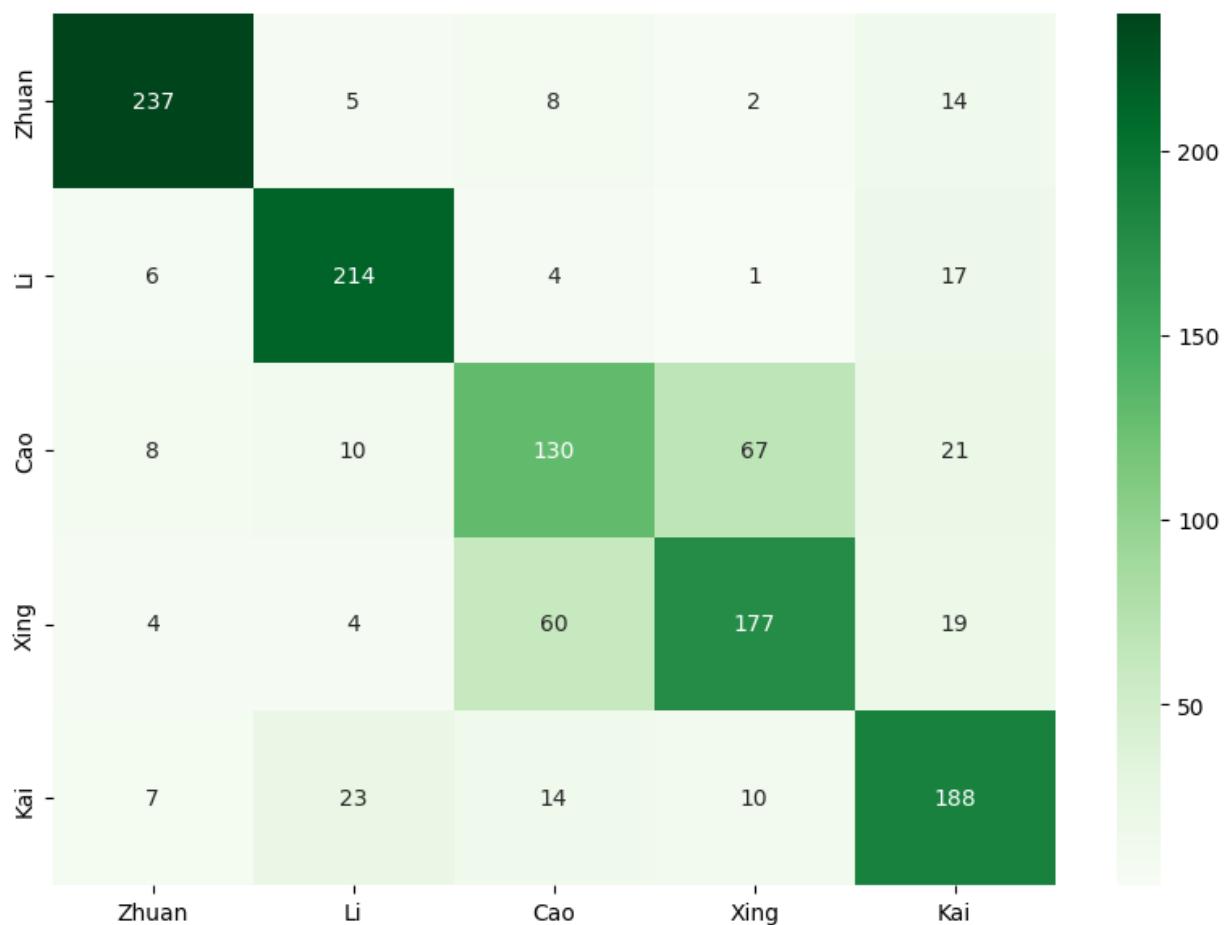
```
import seaborn as sn

import pandas as pd

df_cm = pd.DataFrame(cm, index = [i for i in ['Zhuan','Li','Cao','Xing','Kai']],
                      columns = [i for i in ['Zhuan','Li','Cao','Xing','Kai']])

plt.figure(figsize = (10,7))

sn.heatmap(df_cm, annot=True, cmap="Greens", fmt="d")
```



横向表示真实标签，纵向表示测试样本得到的标签。例如第一行第一列的 237，表示在篆书的测试样本中，有 237 个被识别为了篆书。即对角线的绿色越深，其他色块颜色越浅，识别的准确率越高。