第 5 章 车牌分割定位识别

CHAPTER 5

车牌自动识别模块是现代社会智能交通系统(ITS)的重要组成部分,是图像处理和模式识别技术研究的热点,具有非常广泛的应用。车牌识别主要包括以下 4 个步骤:

- 车牌图像处理;
- 车牌定位处理;
- 车牌字符处理;
- 车牌字符识别。

本章通过对采集的车牌图像进行灰度变换、边缘检测、腐蚀及平滑等过程进行车牌图像预处理,并由此得到一种基于车牌颜色纹理特征的车牌定位方法,最终实现了车牌区域定位。车牌字符分割是为了方便后续对车牌字符进行匹配,从而对车牌进行识别。

5.1 基本概述

车牌定位与字符识别技术以计算机图像处理、模式识别等技术为基础,通过对原图像进行预处理及边缘检测等过程来实现对车牌区域的定位,然后对车牌区域进行图像裁剪、归一化、字符分割及保存,最后将分割得到的字符图像与模板库的模板进行匹配识别,输出匹配结果,流程图如图 5-1 所示。

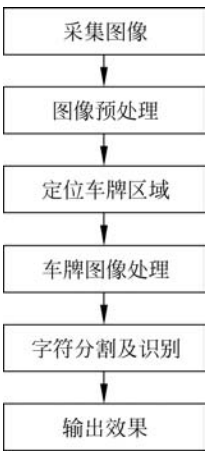


图 5-1 车牌定位与字符识别流程图

在进行车牌识别时首先要正确分割车牌区域,为此人们已经提出了很多方法:使用霍夫变换检测直线来定位车牌边界获取车牌区域;使用灰度阈值分割、区域生长等方法进行区域分割;使用纹理特征分析技术检测车牌区域等。霍夫变换对图像噪声比较敏感,因此在检测车牌边界直线时容易受到车牌变形或噪声等因素的影响,具有较大的误检测概率。灰度阈值分割、区域生长等方法则比霍夫直线检测方法稳定,但当图像中包含某些与车牌灰度非常相似的区域时,便不再适用了。同理,纹理特征分析方法在遇到与车牌纹理特征相似的区域或其他干扰时,车牌定位的正确性也会受到影响。因此,仅采用单一的方法难以达到实际应用的要求。

车牌分割定位识别主要包括 3 个步骤:一是车牌图像处理;

二是车牌定位处理；三是车牌字符处理；四是车牌字符识别。下面分别进行介绍。

5.2 车牌图像处理

原本的图像每个像素点都是 RGB 定义的，或者称为有 R/G/B 3 个通道。在这种情况下，很难区分谁是背景，谁是字符，所以需要对图像进行一些处理，把每个 RGB 定义的像素点都转化成一个 bit 位(即 0-1 代码)，具体方法如下。

5.2.1 图像灰度化

RGB 图像根据三基色原理，每种颜色都可以由红、绿、蓝 3 种基色按不同的比例构成，所以车牌图像的每个像素都由 3 个数值来指定红、绿、蓝的颜色分量。灰度图像实际上是一个数据矩阵 $\boldsymbol{I}$，该矩阵中每个元素的数值都代表一定范围内的亮度值，矩阵 $\boldsymbol{I}$ 可以是整型、双精度，通常 0 代表黑色，255 代表白色。

在 RGB 模型中，如果 $R=G=B$，则表示一种灰度颜色。其中 $R=G=B$ 的值叫作灰度值，由彩色转为灰度的过程称为图像灰度化处理。因此，灰度图像是指只有强度信息而没有颜色信息的图像。一般而言，可采用加权平均值法对原始 RGB 图像进行灰度化处理，该方法的主要思想是，从原始图像中取 R、G、B 各层的像素值经加权求和得到灰度图的亮度值。在现实生活中，人眼对绿色(G)敏感度最高，对红色(R)敏感度次之，对蓝色(B)敏感度最低，因此为了选择合适的权值对象输出合理的灰度图像，权值系数应该满足 $G>R>B$。实验和理论证明，当 R、G、B 的权值系数分别选择 0.299、0.587 和 0.114 时，能够得到最适合人眼观察的灰度图像。

5.2.2 二值化

灰度图像二值化在图像处理的过程中有着很重要的作用，图像二值化处理不仅能使数据量大幅减少，还能突出图像的目标轮廓，便于进行后续的图像处理和分析。对车牌灰度图像而言，所谓的二值化处理就是将其像素点的灰度值设置为 0 或 255，从而让整幅图片呈现黑白效果。因此，对灰度图像进行适当的阈值选取，可以在图像二值化的过程中保留某些关键的图像特征。在车牌图像二值化的过程中，灰度大于或等于阈值的像素点被判定为目标区域，其灰度值用 255 表示；否则这些像素点被判定为背景或噪声而排除在目标区域以外，其灰度值用 0 表示。

图像二值化是指在整幅图像内仅保留黑、白二值的数值矩阵，每个像素都取两个离散数值(0 或 1)之一，其中 0 代表黑色，1 代表白色。在车牌图像处理系统中，进行图像二值化的关键是选择合适的阈值，使得车牌字符与背景能够得到有效分割。采用不同的阈值设定方法对车牌图像进行处理也会产生不同的二值化处理结果：阈值设置得过小，则容易误分割，产生噪声，影响二值变换的准确度；阈值设置得过大，则容易过分割，降低分辨率，使噪声信号被视为噪声而被过滤，造成二值变换的目标损失。

5.2.3　边缘检测

边缘是指图像局部亮度变化最显著的部分，主要存在于目标与目标、目标与背景、区域与区域、颜色与颜色之间，是图像分割、纹理特征提取和形状特征提取等图像分析的重要步骤之一。在车牌识别系统中，边缘提取对于车牌位置的检测有很重要的作用，常用的边缘检测算子有很多，如 Roberts、Sobel、Prewitt、Log 及 Canny 等。据实验分析，Canny 算子于边缘的检测相对精确，能更多地保留车牌区域的特征信息。

Canny 算子在边缘检测中有以下明显的判别指标。

1. 信噪比

信噪比越大，提取的边缘质量越高。信噪比(SNR)的定义为：

$$\mathrm{SRN}=\frac{\left|\int_{-W}^{+W}G(-x)h(x)\mathrm{d}x\right|}{\sigma\sqrt{\int_{-W}^{+W}h^{2}(x)\mathrm{d}x}}$$

式中，$G(x)$代表边缘函数，$h(x)$代表宽度为 W 的滤波器的脉冲响应，σ 代表高斯噪声的均方差。

2. 定位精度

边缘的定位精度 L 的定义为：

$$L=\frac{\left|\int_{-W}^{+W}G'(-x)h'(x)\mathrm{d}x\right|}{\sigma\sqrt{\int_{-W}^{+W}h'^{2}(x)\mathrm{d}x}}$$

式中，$G'(x)$、$h'(x)$分别是 $G(x)$、$h(x)$的导数。L 越大，定位精度越高。

3. 单边缘响应

为了保证单边缘只有一个响应，检测算子的脉冲响应导数的零交叉点的平均距离 $D(f')$ 应满足：

$$D(f')=\pi\left\{\frac{\int_{-\infty}^{+\infty}h'^{2}(x)\mathrm{d}x}{\int_{-\infty}^{+\infty}h''(x)\mathrm{d}x}\right\}^{\frac{1}{2}}$$

式中，$h''(x)$是 $h(x)$的二阶导数。

以上述指标和准则为前提，采用 Canny 算子的边缘检测算法步骤为：

(1) 预处理。采用高斯滤波器进行图像平均。

(2) 梯度计算。采用一阶偏导的有限差分来计算梯度，获取其幅值和方向。

(3) 梯度处理。采用非极大值抑制方法对梯度幅值进行处理。

(4) 边缘提取。采用双阈值算法检测和连接边缘。

5.2.4　形态学运算

数学形态图像处理的基本运算有 4 个：膨胀(或扩张)、腐蚀(侵蚀)、开启和闭合。二值形态学中的运算对象是集合，通常给出了一个图像集合和一个结构元素集合，利用结构元素

对图像集合进行形态学操作。

膨胀运算符号为⊕，图像集合 A 用结构元素 B 来膨胀，记作 $A \oplus B$，定义为：

$$A \oplus B = \{x \mid [(\hat{B})_x \cap A] \neq \phi\}$$

式中，$\hat{B}$ 表示 B 的映像，即与 B 关于原点对称的集合。因此，用 B 对 A 进行膨胀的运算过程如下：首先做 B 关于原点的映射得到映像，再将其平移 x，当 A 与 B 映像的交集不为空时，B 的原点就是膨胀集合的像素。

腐蚀运算的符号是 Θ，图像集合 A 用结构元素 B 来腐蚀，记作 $A \Theta B$，定义为：

$$A \Theta B\{x \mid (B)_x \subseteq A\}$$

因此，A 用 B 腐蚀的结果是所有满足将 B 平移 x 后 B 仍旧被全部包含在 A 中的集合中，也就是结构元素 B 经过平移后全部被包含在集合 A 中原点所组成的集合中。

在一般情况下，由于受到噪声的影响，车牌图像在阈值化后得到的边界往往是不平滑的，在目标区域内部也有一些噪声孔洞，在背景区域会散布一些小的噪声干扰。通过连续的开运算和闭运算可以有效地改善这种情况，有时甚至需要经过多次腐蚀之后再加上相同次数的膨胀，才可以产生比较好的效果。

5.2.5 滤波处理

图像滤波能够在尽量保留图像细节特征的条件下对噪声进行抑制，是图像预处理中常用的操作之一，其处理效果的好坏将直接影响后续的图像分割和识别的有效性和稳定性。

均值滤波也被称为线性滤波，采用的主要方法为邻域平均法。该方法对滤波像素的位置(x,y)选择一个模板，该模板由其邻近的若干像素组成，求出模板中所包含像素的均值，再把该均值赋予当前像素点(x,y)，将其作为处理后的图像在该点上的灰度值 $g(x,y)$，即 $g(x,y)=\frac{1}{M}\sum f(x,y)$，$M$ 为该模板中包含当前像素在内的像素总个数。

在一般情况下，在研究目标车牌时所出现的图像噪声都是无用的信息，而且会对目标车牌的检测和识别造成干扰，极大地降低了图像质量，影响图像增强、图像分割、特征提取、图像识别等后继工作的进行。因此，在程序实现中为了能有效地进行图像去噪，并且能有效地保存目标车牌的形状、大小及特定的几何和拓扑结构特征，需要对车牌进行均值滤波去噪处理。

5.3 定位处理

车牌区域具有明显的特点，因此根据车牌底色、字色等有关知识，可采用彩色像素点统计的方法分割出合理的车牌区域。下面以蓝底白字的普通车牌为例说明彩色像素点统计的分割方法。假设经数码相机或 CCD 摄像头拍摄采集到了包含车牌的 RGB 彩色图像，将水平方向记为 y，将垂直方向记为 x，则：首先，确定车牌底色 RGB 各分量分别对应的颜色范围；其次，在 y 方向统计此颜色范围内的像素点数量，设定合理的阈值，确定车牌在 y 方向的合理区域；再次，在分割出的 y 方向区域内统计 x 方向上此颜色范围内的像素点数量，设定合理的阈值进行定位；最后，根据 x、y 方向的范围来确定车牌区域，实现定位。

5.4　字符处理

5.4.1　阈值分割

阈值分割算法是图像分割中应用场景最多的算法之一。简单地说，对灰度图像进行阈值分割就是先确定一个处于图像灰度取值范围内的阈值，然后将图像中各个像素的灰度值与这个阈值进行比较，并根据比较的结果将对应的像素划分为两类：像素灰度大于阈值的一类和像素灰度小于阈值的另一类，灰度值等于阈值的像素可以被归入这两类之一。分割后的两类像素一般分属图像的两个不同区域，所以对像素根据阈值分类达到了区域分割的目标。由此可见，阈值分割算法主要有以下两个步骤。

(1) 确定需要分割的阈值。

(2) 将阈值与像素点的灰度值进行比较，以分割图像的像素。

在以上步骤中，如果能确定一个合适的阈值，就可以准确地将图像进行分割。在阈值确定后，将阈值与像素点的灰度值进行比较和分割，就可对各像素点并行处理，通过分割的结果直接得到目标图像区域。一般选用最常用的图像双峰灰度模型进行阈值分割：假设图像目标和背景直方图具有单峰分布的特征，且处于目标和背景内部相邻像素间的灰度值是高度相关的，但处于目标和背景交界处两边的像素在灰度值上有很大的差别。如果一幅图像满足这些条件，则它的灰度直方图基本上可看作由分别对应目标和背景的两个单峰构成。如果这两个单峰部分的大小接近且均值相距足够远，两部分的均方差也足够小，则直方图在整体上呈现较明显的双峰现象。同理，如果在图像中有多个呈现单峰灰度分布的目标，则直方图在整体上可能呈现较明显的多峰现象。因此，对这类图像可用取多级阈值的方法来得到较好的分割效果。

如果要将图像中不同灰度的像素分成多个类，则需要选择一系列的阈值将像素分到合适的类别中。如果只用一个阈值分割，则称为单阈值分割算法；如果用多个阈值分割，则称为多阈值分割算法。因此，单阈值分割可看作多阈值分割的特例，许多单阈值分割算法可被推广到多阈值分割算法中。同理，在某些场景下也可将多阈值分割问题转化为一系列的单阈值分割问题来解决。以单阈值分割算法为例，对一幅原始图像 $f(x,y)$ 取单阈值 T 分割得到二值图像可定义为：

$$g(x,y)=\begin{cases}1, & f(x,y)>T\\ 0, & f(x,y)\leqslant T\end{cases}$$

这样得到的 $g(x,y)$ 是一幅二值图像。

在一般的多阈值分割情况下，阈值分割输出的图像可表示为：

$$g(x,y)=k \quad T_{k-1}\leqslant f(x,y)<T_k \quad k=1,2,\cdots,K$$

式中，$T_0,T_1,\cdots,T_k,\cdots,T_K$ 是一系列分割阈值，k 表示赋予分割后图像的各个区域的不同标号。

5.4.2　阈值化分割

车牌字符图像的分割目的是将车牌的整体区域分割成单字符区域，以便后续识别。其分割难点在于受字符与噪声粘连以及字符断裂等因素的影响。均值滤波是典型的线性滤波

算法，指在图像上对图像进行模板移动扫描，该模板包括像素周围的近邻区域，通过模板与命中的近邻区域像素的平均值来代替原来的像素值，实现去噪的效果。为了从车牌图像中直接提取目标字符，最常用的方法是设定一个阈值 T，用 T 将图像的像素分成两部分：大于 T 像素集合和小于 T 的像素集合，得到二值化图像。

5.4.3 归一化处理

字符图像归一化是简化计算的方式之一，在车牌字符分割后往往会出现大小不一致的情况，因此可采用基于图像放缩的归一化处理方式将字符图像进行大小放缩，以得到统一大小的字符像素，便于后续的字符识别。

5.4.4 字符分割经典应用

【例 5-1】 通过一个案例来分析利用字符分割实现车牌的分割处理。

```
import cv2
"""读取图像,并把图像转换为灰度图像并显示 """
img = cv2.imread("car.png")                          # 读取图片
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)     # 转换了灰度化
cv2.imshow('gray', img_gray)                         # 显示图片
cv2.waitKey(0)

"""将灰度图像二值化,设定阈值是 100 """
img_thre = img_gray
cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV, img_thre)
cv2.imshow('threshold', img_thre)
cv2.waitKey(0)

"""保存黑白图片 """
cv2.imwrite('thre_res.png', img_thre)

"""分割字符 """
white = []                                           #记录每一列的白色像素总和
black = []                                           #黑色
height = img_thre.shape[0]
width = img_thre.shape[1]
white_max = 0
black_max = 0
# 计算每一列的黑白色像素总和
for i in range(width):
  s = 0                                              # 这一列白色总数
  t = 0                                              # 这一列黑色总数
  for j in range(height):
    if img_thre[j][i] == 255:
      s += 1
    if img_thre[j][i] == 0:
      t += 1
  white_max = max(white_max, s)
  black_max = max(black_max, t)
```

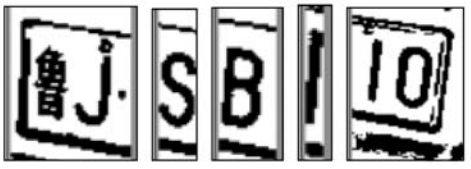
```
    white.append(s)
    black.append(t)
    print(s)
    print(t)

arg = False #False 表示白底黑字; True 表示黑底白字
if black_max > white_max:
    arg = True
#分割图像
def find_end(start_):
    end_ = start_ + 1
    for m in range(start_ + 1, width - 1):
        if (black[m] if arg else white[m]) > (0.95 * black_max if arg else 0.95 * white_max):
# 0.95 这个参数可调整,对应下面的 0.05
            end_ = m
            break
    return end_

n = 1
start = 1
end = 2
while n < width - 2:
    n += 1
    if (white[n] if arg else black[n]) > (0.05 * white_max if arg else 0.05 * black_max):
        #上面这些判断用来辨别是白底黑字还是黑底白字
        #0.05 这个参数可调整,对应上面的 0.95
        start = n
        end = find_end(start)
        n = end
        if end - start > 5:
            cj = img_thre[1:height, start:end]
            cv2.imshow('caijian', cj)
            cv2.waitKey(0)
```

运行程序,效果如图 5-2 所示。

(a) 原始图像　(b) 灰度图像　(c) 二值图像



(d) 分割后字符

图 5-2　字符分割车牌

由图 5-2 可看出，分割效果不是很好，当遇到干扰较多的图片，比如左右边框太大、噪点太多，这样就不能分割出来，读者可以试一下不同的照片。

5.5 字符识别

车牌字符识别方法基于模式识别理论，常用的有以下几类。

1. 结构识别

结构识别主要由识别及分析两部分组成：识别部分主要包括预处理、基元抽取（包括基元和子图像之间的关系）和特征分析；分析部分包括基元选择及结构推理。

2. 统计识别

统计识别用于确定已知样本所属的类别，以数学上的决策论为理论基础，并由此建立统计学识别模型。其基本方式是对所研究的图像实施大量的统计分析，寻找规律性认知，提取反映图像本质的特征并进行识别。

3. BP 神经网络

BP 神经网络以 B 神经网络模型为基础，属于误差后向传播的神经网络，是神经网络中使用最广泛的一类，采用了输入层、隐藏层和输出层 3 层网络的层间全互联方式，具有较高的运行效率和识别准确率。

4. 模板匹配

模板匹配是数字图像处理中最常用的识别方法之一，通过建立已知的模式库，再将其应用到输入模式中寻找与之最佳匹配模式的处理步骤，得到对应的识别结果，具有很高的运行效率。基于模板匹配的字符识别方法的过程为：

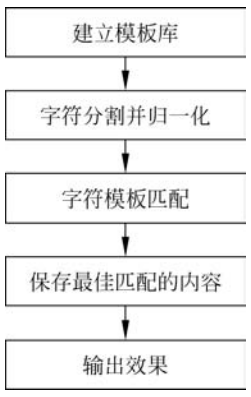


图 5-3 字符识别流程图

（1）建库。建立已标准化的字符模板库。

（2）对比。将归一化的字符图像与模板库中的字符进行对比，在实际实验中充分考虑了我国普通小汽车牌照的特点，即：第 1 位字符是汉字，分别对应各个省的简称；第 2 位是 A～Z 的字母；后 5 位则是数字和字母的混合搭配。因此，为了提高对比的效率和准确性，分别对第 1 位、第 2 位和后 5 位字符进行识别。

（3）输出。在识别完成后输出所得到的车牌字符结果。

其流程如图 5-3 所示。

5.5.1 模板匹配的字符识别

模板匹配是图像识别方法中最具有代表性的基本方法之一，该方法首先根据已知条件建立模板库 $T(i,j)$，然后从待识别的图像或图像区域 $f(i,j)$中提取若干特征量与 $T(i,j)$相应的特征量进行对比，分别计算它们之间归一化的互相关量。其中，互相关量最大的一个表示二者的相似程度最高，可将图像划到该类别。此外，也可以计算图像与模板特征量之间的距离，采用最小距离法判定所属类别。但是，在实际情况下，用于匹配的图像的采集成像

条件往往存在差异，可能会产生较大的噪声干扰。此外，图像经过预处理和归一化处理等步骤，其灰度或像素点的位置也可能会发生改变，进而影响识别效果。因此，在实际设计模板时，需要保持各区域形状的固有特点，突出不同区域的差别，并充分考虑处理过程可能会引起的噪声和位移等因素，按照基于图像不变的特性所对应的特征向量来构建模板，提高识别系统的稳定性。

本实例采用的特征向量距离计算的方法来求得字符与模板中字符的最佳匹配，然后找到对应的结果进行输出。首先，遍历字符模板；其次，依次将待识别的字符与模板进行匹配，计算其与模板字符的特征距离，得到的值越小就越匹配；再次，将每幅字符图像的匹配结果都进行保存；最后，有 7 个字符匹配识别结果即可作为车牌字符进行输出。

5.5.2　字符识别车牌经典应用

车牌自动识别系统以车牌的动态视频或静态图像作为输入，通过牌照颜色、牌照号码等关键内容的自动识别来提取车牌的详细信息。某些车牌识别系统具有通过视频图像判断车辆驶入监控区域的功能，一般被称为视频车辆检测，被广泛应用于道路车流量统计等方面。在现实生活中，一个完整的车牌识别系统应包括车辆检测、图像采集、车牌定位、车牌识别等模块。

车牌信息是一辆汽车独一无二的标识，所以车牌识别技术可以作为辨识一辆车最为有效的方法。车牌识别系统包括汽车图像的输入、车牌图像的预处理、车牌定位和字符检测、车牌字符的分割和车牌字符识别等部分，如图 5-4 所示。

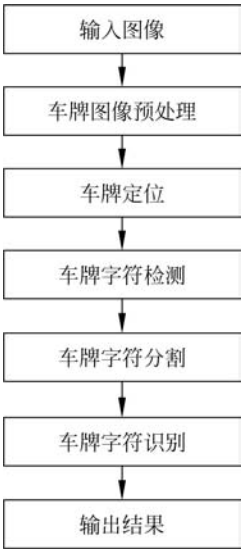


图 5-4　车牌识别流程图

【例 5-2】　利用字符识别车牌。

```
import cv2
import numpy as np
from PIL import Image
import os.path
from skimage import io,data
def stretch(img):
    '''
    图像拉伸函数
    '''
    maxi = float(img.max())
    mini = float(img.min())
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img[i,j] = (255/(maxi - mini) * img[i,j] - (255 * mini)/(maxi - mini))
    return img

def dobinaryzation(img):
    '''
    二值化处理函数
    '''
    maxi = float(img.max())
    mini = float(img.min())
```

```
    x = maxi - ((maxi - mini)/2)
    #二值化,返回阈值 ret 和二值化操作后的图像 thresh
    ret,thresh = cv2.threshold(img,x,255,cv2.THRESH_BINARY)
    #返回二值化后的黑白图像
    return thresh

def find_rectangle(contour):
    '''
    寻找矩形轮廓
    '''
    y,x = [],[]

    for p in contour:
        y.append(p[0][0])
        x.append(p[0][1])
    return [min(y),min(x),max(y),max(x)]

def locate_license(img,afterimg):
    '''
    定位车牌号
    '''
    img, contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
SIMPLE)
    block = []
    for c in contours:
        #找出轮廓的左上点和右下点,由此计算它的面积和长度比
        r = find_rectangle(c)
        a = (r[2] - r[0]) * (r[3] - r[1])              #面积
        s = (r[2] - r[0]) * (r[3] - r[1])              #长度比
        block.append([r,a,s])
    #选出面积最大的 3 个区域
    block = sorted(block,key = lambda b: b[1])[ - 3:]
    #使用颜色识别判断找出最像车牌的区域
    maxweight,maxindex = 0, - 1
    for i in range(len(block)):
        b = afterimg[block[i][0][1]:block[i][0][3],block[i][0][0]:block[i][0][2]]
        #BGR 转 HSV
        hsv = cv2.cvtColor(b,cv2.COLOR_BGR2HSV)
        #蓝色车牌的范围
        lower = np.array([100,50,50])
        upper = np.array([140,255,255])
        #根据阈值构建掩膜
        mask = cv2.inRange(hsv,lower,upper)
        #统计权值
        w1 = 0
        for m in mask:
            w1 += m/255
        w2 = 0
        for n in w1:
            w2 += n
        #选出最大权值的区域
```

```
            if w2 > maxweight:
                maxindex = i
                maxweight = w2
        return block[maxindex][0]

def find_license(img):
    '''
    预处理函数
    '''
    m = 400 * img.shape[0]/img.shape[1]
    #压缩图像
    img = cv2.resize(img,(400,int(m)),interpolation = cv2.INTER_CUBIC)
    #RGB转换为灰度图像
    gray_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    #灰度拉伸
    stretchedimg = stretch(gray_img)
    '''进行开运算,用来去除噪声'''
    r = 16
    h = w = r * 2 + 1
    kernel = np.zeros((h,w),np.uint8)
    cv2.circle(kernel,(r,r),r,1, - 1)
    #开运算
    openingimg = cv2.morphologyEx(stretchedimg,cv2.MORPH_OPEN,kernel)
    #获取差分图,两幅图像做差运算 cv2.absdiff('图像1','图像2')
    strtimg = cv2.absdiff(stretchedimg,openingimg)
    #图像二值化
    binaryimg = dobinaryzation(strtimg)
    #canny边缘检测
    canny = cv2.Canny(binaryimg,binaryimg.shape[0],binaryimg.shape[1])
    '''消除小的区域,保留大块的区域,从而定位车牌'''
    #进行闭运算
    kernel = np.ones((5,19),np.uint8)
    closingimg = cv2.morphologyEx(canny,cv2.MORPH_CLOSE,kernel)
    #进行开运算
    openingimg = cv2.morphologyEx(closingimg,cv2.MORPH_OPEN,kernel)
    #再次进行开运算
    kernel = np.ones((11,5),np.uint8)
    openingimg = cv2.morphologyEx(openingimg,cv2.MORPH_OPEN,kernel)
    #消除小区域,定位车牌位置
    rect = locate_license(openingimg,img)
    return rect,img

def cut_license(afterimg,rect):
    '''
    图像分割函数
    '''
    #转换为宽度和高度
    rect[2] = rect[2] - rect[0]
    rect[3] = rect[3] - rect[1]
    rect_copy = tuple(rect.copy())
    rect = [0,0,0,0]
```

```
    #创建掩膜
    mask = np.zeros(afterimg.shape[:2],np.uint8)
    #创建背景模型大小只能为13*5,行数只能为1,单通道浮点型
    bgdModel = np.zeros((1,65),np.float64)
    #创建前景模型
    fgdModel = np.zeros((1,65),np.float64)
    #分割图像
    cv2.grabCut(afterimg,mask,rect_copy,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
    mask2 = np.where((mask == 2)|(mask == 0),0,1).astype('uint8')
    img_show = afterimg * mask2[:,:,np.newaxis]
    return img_show

def deal_license(licenseimg):
    '''
    车牌图片二值化
    '''
    #车牌变为灰度图像
    gray_img = cv2.cvtColor(licenseimg,cv2.COLOR_BGR2GRAY)
    #均值滤波去除噪声
    kernel = np.ones((3,3),np.float32)/9
    gray_img = cv2.filter2D(gray_img, -1,kernel)
    #二值化处理
    ret,thresh = cv2.threshold(gray_img,120,255,cv2.THRESH_BINARY)
    return thresh

def find_end(start,arg,black,white,width,black_max,white_max):
    end = start + 1
    for m in range(start + 1,width - 1):
        if (black[m] if arg else white[m])>(0.98 * black_max if arg else 0.98 * white_max):
            end = m
            break
    return end

if __name__ == '__main__':
    img = cv2.imread('car1.jpg',cv2.IMREAD_COLOR)
    #预处理图像
    rect,afterimg = find_license(img)
    #框出车牌号
    cv2.rectangle(afterimg,(rect[0],rect[1]),(rect[2],rect[3]),(0,255,0),2)
    cv2.imshow('afterimg',afterimg)
    #分割车牌与背景
    cutimg = cut_license(afterimg,rect)
    cv2.imshow('cutimg',cutimg)
    #二值化生成黑白图
    thresh = deal_license(cutimg)
    cv2.imshow('thresh',thresh)
    cv2.waitKey(0)
    #分割字符
    '''
    判断底色和字色
    '''
```

```
#记录黑白像素总和
white = []
black = []
height = thresh.shape[0]
width = thresh.shape[1]
white_max = 0
black_max = 0
#计算每一列的黑白像素总和
for i in range(width):
    line_white = 0
    line_black = 0
    for j in range(height):
        if thresh[j][i] == 255:
            line_white += 1
        if thresh[j][i] == 0:
            line_black += 1
    white_max = max(white_max,line_white)
    black_max = max(black_max,line_black)
    white.append(line_white)
    black.append(line_black)
    print('white',white)
    print('black',black)
#arg 为 True 表示黑底白字,False 为白底黑字
arg = True
if black_max < white_max:
    arg = False
n = 1
start = 1
end = 2
s_width = 28
s_height = 28
while n < width - 2:
    n += 1
    #判断是白底黑字还是黑底白字 0.02 参数对应上面的 0.98,可做调整
    if(white[n] if arg else black[n])>(0.02 * white_max if arg else 0.02 * black_max):
        start = n
        end = find_end(start,arg,black,white,width,black_max,white_max)
        n = end
        if end - start > 5:
            cj = thresh[1:height,start:end]
            print("result/%s.jpg" % (n))
            #保存分割的图片 by cayden
            infile = "result/%s.jpg" % (n)
            io.imsave(infile,cj)
            cv2.imshow('cutlicense',cj)
            cv2.waitKey(0)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行程序，效果如图 5-5 所示。

(a) 原始图像

(b) 剪切后图像

(c) 灰度图像

(d) 二值图像

(e) 分割后字符

图 5-5　车牌字符识别效果