中山大學
SUN YAT-SEN UNIVERSITY

数据挖掘导论大作业

题目 Title：车牌识别算法

院 系
School (Department)：计算机学院

专 业
Major： 软件工程

学生姓名
Student Name：鲁沛 、马靖成、陆世炜

学 号
Student No.：18342066 、18342071 、18342068

时间： 2021 年 7 月 10 日

Date: Month 7 Day 10 Year 2021

【摘　要】

本文实现了一个基本的车牌识别算法，四个主要环节包括图像预处理、车牌定位、字符分割和字符识别。并在此基础上讨论了当前车牌识别面临的一些问题以及未来的发展趋势。

【关键词】：Machine learning; Intelligent transportation; License plate recognition; Character segmentation; SVM; opencv;

[ABSTRACT]

In this paper, a basic license plate recognition algorithm is implemented. The four main steps include image preprocessing, license plate location, character segmentation and character recognition. On this basis, some problems and future development trend of license plate recognition are discussed.
[Keywords]: Machine learning; License plate recognition; opencv；

目　录

第一章 概述/引言

1.1 研究基于机器学习的车牌识别算法的背景和意义

随着我国社会经济、公路运输的高速发展，以及汽车拥有量的急剧增加。采用先进、高效、准确的智能交通管理系统迫在眉睫，车辆监控和管理的自动化、智能化在交通系统中具有十分重要的意义。车辆臼动识别系统能广泛应用于公路和桥梁收费站、城市交通监控系统、港口、机场和停车场等车牌认证的实际交通系统中，以提高交通系统的车辆监控和管理的自动化程度。

1.2 问题的描述

使用 python+opencv 完成一个基本的车牌识别算法。

1.3 本文的工作

本文使用二值化、高斯滤波、开运算与闭运算对车辆照片进行预处理，找到若干可能是车牌的矩形区域。接着对倾斜的矩形区域进行角度矫正，使用颜色定位找到最可能是车牌的矩形（目前识别车牌的颜色只有蓝、绿、黄三种颜色），同时匹配出车牌的类型。然后用投影法分割字符，得到每个单独的字符。最后用支持向量机训练出每个字符的分类模型，将分割后的每个图像逐个与模型进行匹配，得到最终的识别结果。

1.4 论文结构简介

本文第一章主要是阐述研究基于机器学习的车牌识别算法的背景和意义以及本文所有大致工作内容；第二章主要综述本文在完成车牌识别算法的过程中所作的相关

工作以及选择的考量；第三章将会介绍我在解决问题过程中所作的工作以及所提出的方法和实现算法的步骤；第四章是实验结果的评估；第五章是对本文的总结以及对接下来研究方向的展望；最后是参考文献以及致谢内容。

第二章 相关工作综述

2.1 车牌识别相关发展

1990 年美国智能交通学会提出智能交通的概念，随即我国也开始对车牌识别技术进行研究。它融合了智能控制、计算机视觉、图像处理和通信技术等诸多电子技术为一体，使交通向着合理化、人性化和智能化的方向前进。[1]

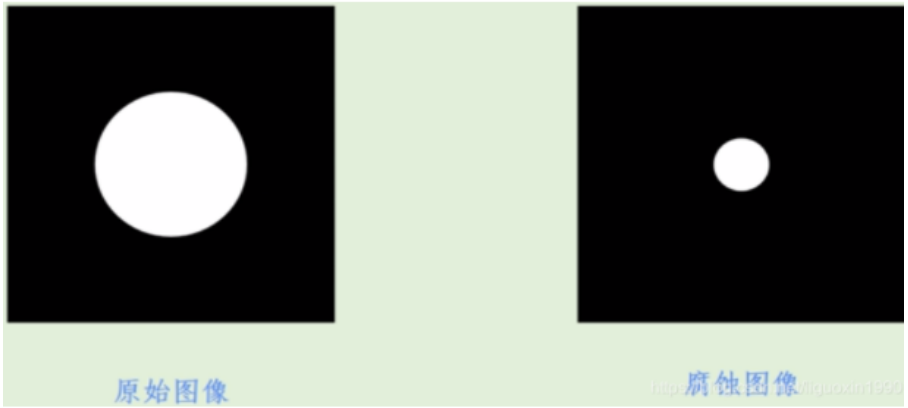
目前车牌识别吸引了众多的公司与学者对其进行研究现在就高校而言有北方交通大学，清华大学、上海交通大学，和浙江大学等高校;就公司而言，有亿阳，上海宝通，三联，上海高德威公司。车牌识别技术已在公安交警、交通运输、城市停车场、居民小区管理等领域开始应用，但应用环境情况复杂，车牌识别技术还不够完善，因此在应用中还存在诸多问题。[2]

2.2 开运算与闭运算

图像腐蚀（缩小 erode）

概念：腐蚀二值图像中，数值为 1 的前景色，使其丢失自己的边缘而变小。

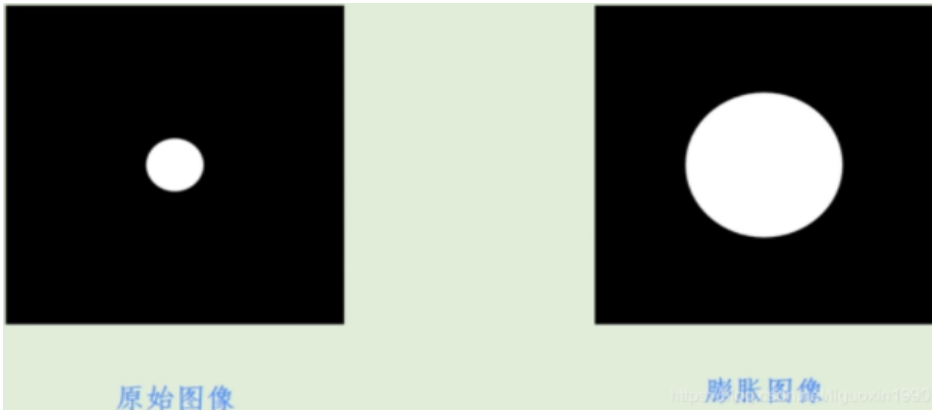
原理：卷积核中心点逐个像素扫描原始图像(遍历)，被扫描到的原始图像的所有像素点,只有当卷积核对应的区域所有元素均为 1 时,其核中心值才保留为 1,否则被置换为 0。



图像膨胀（扩张 dilate）

概念：腐蚀的逆操作。

原理：卷积核中心点逐个像素扫描原始图像(遍历)，被扫描到的原始图像的所有像素点,当卷积核对应的区域只要有一个为 1 时,核中心值就置换为 1,否则为 0。



开运算

原理：开运算=腐蚀（膨胀（img））

效果：开运算能够除去孤立的小点和毛刺，而总的位置和形状不变。

闭运算

原理：闭运算 = 腐蚀(膨胀(img))

效果：有助于去除前景物体内部的小孔,或者物体上的黑点

相关库函数：

cv2.morphologyEx(src, op, kernel)

- src：传入的图片
- op：进行变化的方式
- kernel：表示方框的大小

op = cv2.MORPH_OPEN 进行开运算，op = cv2.MORPH_CLOSE 进行闭运算

2.3 二值化

图像二值化就是将图像上的像素点的灰度值设置为 0 或 255，这样将使整个图像呈现出明显的黑白效果。灰度处理后就能够二值化了，这是方便图像处理的重要步骤，对轮廓有要求的很有效。在数字图像处理中，二值图像占有非常重要的地位，图像的二值化使图像中数据量大为减少，从而能凸显出目标的轮廓。

相关库函数：

cv2.threshold (src, thresh, maxval, type)

- src：源图片，必须是单通道
- thresh：阈值，取值范围 0～255
- maxval：填充色，取值范围 0～255
- type：阈值类型，具体见下表：

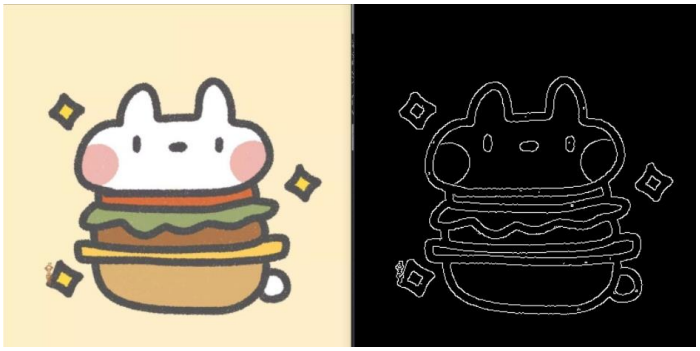
type	小于阈值的像素点	大于阈值的像素点
0	置 0	置填充色
1	置填充色	置 0
2	保持原色	置灰色
3	置 0	保持原色
4	保持原色	置 0

2.4 Canny 边缘检测

边缘是图像最基本的特征,它包含了用于识别的有用信息。因此边缘检测在图像理解、分析和识别领域有着重要的作用。边缘检测的效果会直接影响图像分割和识别的性能[3]。

Canny 边缘检测是一种非常流行的边缘检测算法，是 John Canny 在 1986 年提出的。它是一个多阶段的算法，即由多个步骤构成：

1) 图像降噪
2) 计算图形梯度
3) 非极大值抑制
4) 阈值筛选



相关库函数：

cv2.Canny(image,threshold1,threshold2[,edges[,apertureSize[,L2gradient]]])

- image：需要处理的原图像，该图像必须为单通道的灰度图
- threshold1：阈值 1，用于将间断的边缘连接起来
- threshold2：阈值 2，用于检测图像中明显的边缘

2.5 轮廓检测

轮廓检测也是图像处理中经常用到的。

相关库函数：

cv2.findContours(image, mode, method[, contours[, hierarchy[, offset]]])

- image：输入图像
- mode：轮廓的检索模式，共有 4 种：
  a) cv2.RETR_EXTERNAL：只检测外轮廓；
  b) cv2.RETR_LIST：检测的轮廓不建立等级关系；
  c) cv2.RETR_CCOMP：建立两个等级的轮廓，上面的一层为外边界，里面的一层为内孔的边界信息。如果内孔内还有一个连通物体，这个物体的边界也在顶层；
  d) cv2.RETR_TREE：建立一个等级树结构的轮廓。
- method：轮廓的近似办法

返回值：

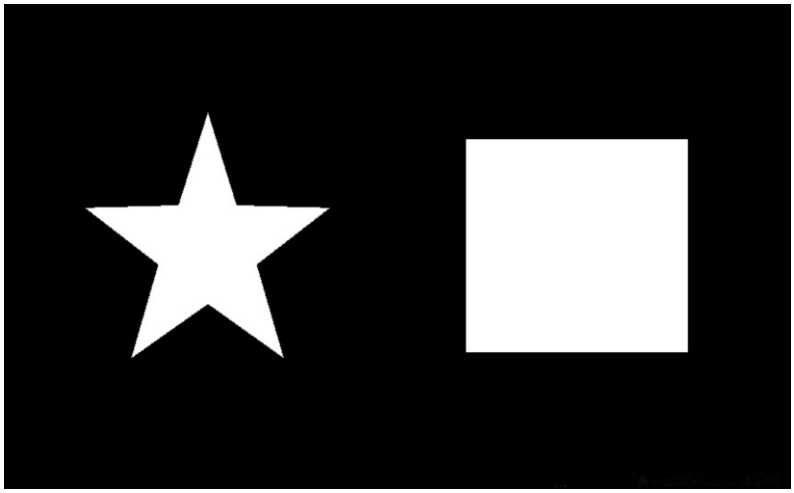
- contour：1 个 list，list 中每个元素都是图像中的一个轮廓，用 numpy 中的 ndarray 表示，每个 ndarray 是轮廓上的点的集合。

- hierarchy:一个 ndarray,其中的元素个数和轮廓个数相同,每个轮廓 contours[i]对应 4 个 hierarchy 元素 hierarchy[i][0] ~hierarchy[i][3],分别表示后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号，如果没有对应项，则该值为负数。

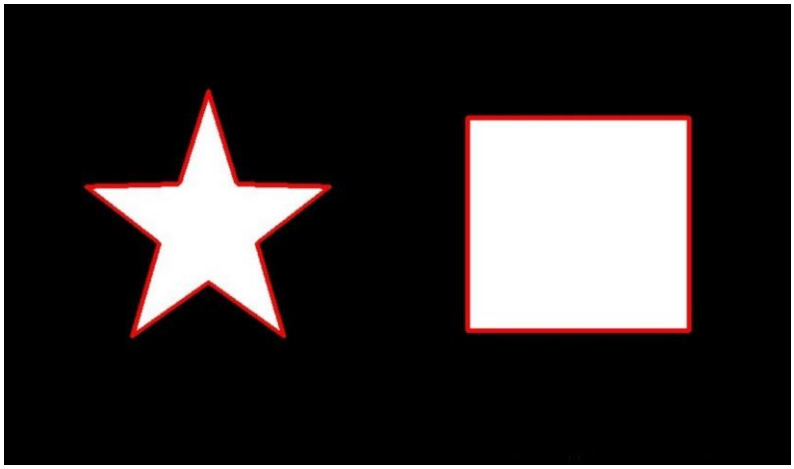
cv2.drawContours(image,contours,contourIdx,color[,thickness[,lineType[,hierarchy[,maxLevel[,offset]]]]])

- image：指明在哪幅图像上绘制轮廓
- contours：轮廓本身，是一个 list
- contourIdx：指定绘制轮廓 list 中的哪条轮廓，如果是-1，则绘制其中的所有轮廓

原图：



绘制轮廓如下：



2.6 生成点集的最小外接矩形

相关库函数：

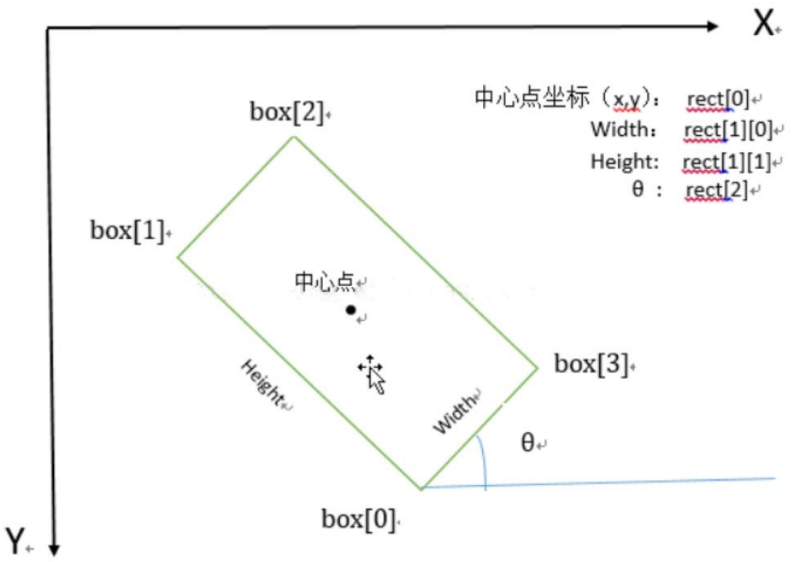
cv2.minAreaRect(cnt)

- cnt：点集数组或向量（里面存放的是点的坐标）

返回值：

- rect：（最小外接矩形的中心（x，y），（宽度，高度），旋转角度）

如果要将矩形绘制出来，需要矩形的四个顶点坐标 box，通过 cv2.BoxPoints 函数获得，返回形式[[x0,y0]，[x1,y1]，[x2,y2]，[x3,y3]]。得到的最小外接矩形的 4 个顶点顺序、中心坐标、宽度、高度、旋转角度（的对应关系如下：



2.7 投影法

垂直投影法是分割字符的一种常用方法。垂直投影法，简单来说就是将每一列上的符合要求的像素都“沉到”图片底部，并找到最薄弱的位置作为切割的位置。换一种说法就是统计出每一列上的符合要求的像素点的个数，并找到其中的薄弱点进行切割。

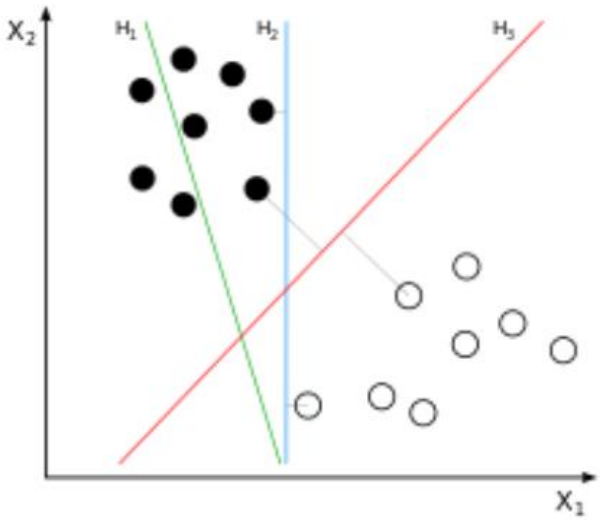
垂直投影法有一个重要的假设就是，两个字符的连接位置相对于字符本身位置的像素点的个数较少，也就是可以通过比较相对位置的像素点的个数得到图片的分割位置。而我国的标准车牌中，每个字符的宽度与高度都相同，分别为 45 毫米与 90 毫米，且除了第二个字符与第三个符的间隔为34毫米外，其余字符的间隔均为12毫米[3]。因此车牌识别中分割字符时很适合用投影法。

2.8 SVM（支持向量机）

支持向量机（support vector machines，SVM）是一种二分类模型，它将实例的特征向量映射为空间中的一些点，SVM 的目的就是想要画出一条线，以 “最好地” 区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。SVM 适合中小型数据样本、非线性、高维的分类问题[4]。

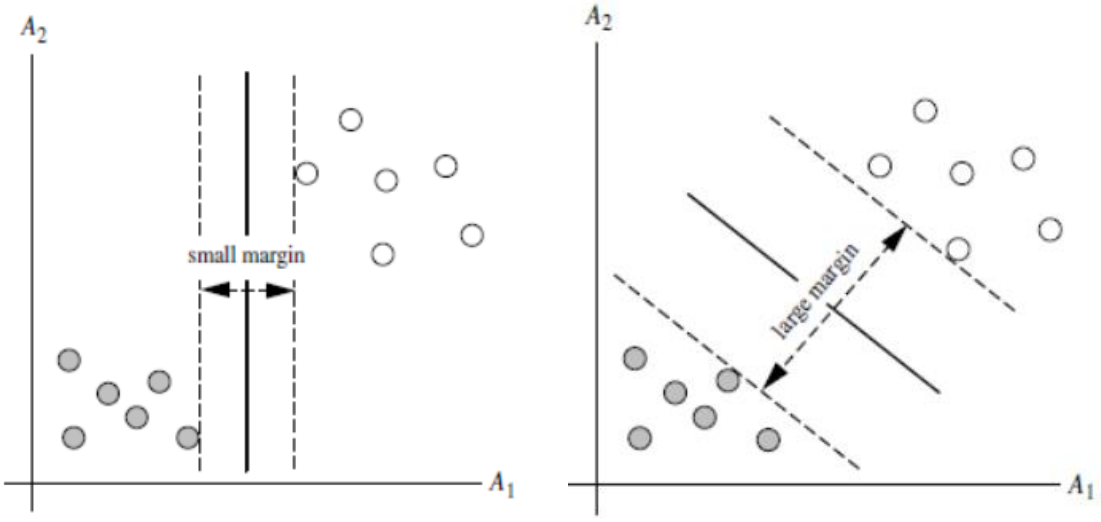
基本概念

将实例的特征向量（以二维为例）映射为空间中的一些点，如下图的实心点和空心点，它们属于不同的两类。SVM 的目的就是想要画出一条线，以“最好地”区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。



线是有无数条可以画的，区别就在于效果好不好，每条线都可以叫做一个划分超平面。比如上面的绿线就不好，蓝线还凑合，红线看起来就比较好。我们所希望找到的这条效果最好的线就是具有 “最大间隔的划分超平面”。

对于任意一个超平面，其两侧数据点都距离它有一个最小距离（垂直距离），这两个最小距离的和就是间隔。比如下图中两条虚线构成的带状区域就是 margin，虚线是由距离中央实线最近的两个点所确定出来的（也就是由支持向量决定）。但此时 margin 比较小，如果用第二种方式画，margin 明显变大也更接近我们的目标。大的 margin 犯错的几率比较小，具有更高的鲁棒性。



SVM 可扩展为多分类问题

1) 对于每个类，有一个当前类和其他类的二类分类器；
2) 将多分类问题转化为 n 个二分类问题，n 就是类别个数。

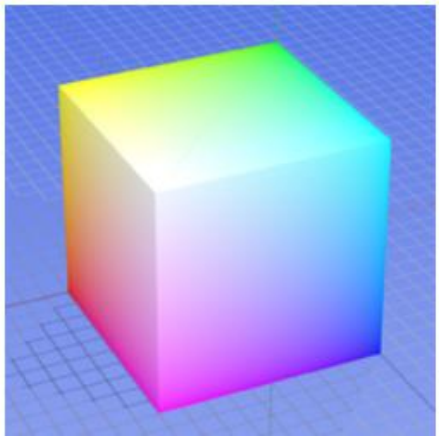
SVM 算法特性

- 训练好的模型的算法复杂度是由支持向量的个数决定的，而不是由数据的维度决定的。所以 SVM 不容易产生 overfitting；
- SVM 训练出来的模型完全依赖于支持向量，即使训练集里面所有非支持向量的点都被去除，重复训练过程，结果仍然会得到完全一样的模型。
- 一个 SVM 如果训练得出的支持向量个数比较少，那么 SVM 训练出的模型比较容易被泛化。

2.9 HSV 颜色模型

RGB 颜色模型

在图像处理中，最常用的颜色空间是 RGB 模型，常用于颜色显示和图像处理，三维坐标的模型形式，非常容易被理解：

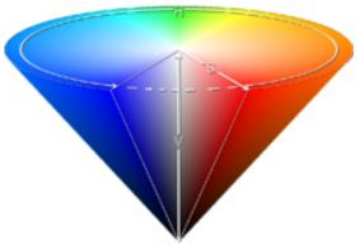


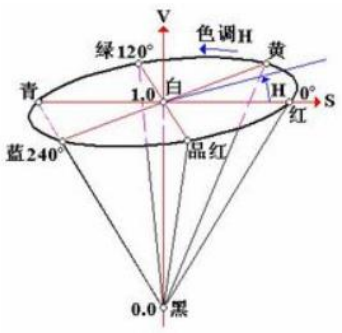
缺点：

1) RGB 空间是颜色显示空间, 并不适合人的视觉，对目标物体的颜色描述相对复杂, 各个分量之间冗余信息多；
2) RGB 空间中两点的欧式距离与实际颜色距离不是线性关系, 在颜色分离中极易引起误分离, 使有用信息漏掉或夹杂其他无用信息；
3) 图像太容易受图像明暗的影响.要对彩色车牌进行彩色分割, 就要将 RGB 颜色空间转化为不太容易受明暗影响且对色彩识别更为方便的 HSV 模型中去

HSV 颜色模型

HSV 颜色模型是一个倒锥模型，这个模型就是按色彩、深浅、明暗来描述的。模型中颜色的参数分别是：色彩（H），纯度（S），明度（V）：





- H 参数表示色彩信息，即所处的光谱颜色的位置。该参数用角度量来表示，红、

绿、蓝分别相隔 120 度。互补色分别相差 180 度；

- 纯度 S 为为比例值，范围从 0 到 1，它表示成所选颜色的纯度和该颜色最大的纯度之间的比率。S=0 时，只有灰度；
- V 表示色彩的明亮程度，范围从 0 到 1。有一点要注意：它和光强度之间并没有直接的联系。

HSV 模型是面向彩色图像处理的最常用的颜色空间, HSV 三维空间坐标系中各坐标之间均有独立的色彩的信息.能较好的反映人对颜色的感知和鉴别能力,非常适合对色彩的图像进行颜色比较.亮度和色度的分离有力于对图象的分割处理,其中 H 和 S 分量可以提取图象中的蓝色和黄色区域.V 分量可以提取白色和黑色区域.为了更有效进行色彩分割,需将原始捕获图像从 RGB 模型转换到 HSV 模型,再在 HSV 模型上进行色彩分割[5]。

相关库函数：

image_hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

可以将原始的 RGB 图像转换成 HSV 图像。

第三章 完成车牌识别算法的方法

3.1 基于机器学习的车牌识别算法的问题描述

将算法分为 4 个子模块：

- 图像预处理
- 车牌定位
- 字符分割
- 字符识别

3.2 逐步完成算法

环境配置：
处理器：i5-8300H CPU 1050Ti GPU
RAM：8G
操作系统：64 位 Windows 10
语言：python 3.7.0 + opencv 3.4.2.16

3.2.1 参数设置

设置原始图片最大宽度，车牌区域允许最大面积等参数：

```
# 创建参数的解析对象
parser = argparse.ArgumentParser(description='PyTorch garbage Training ')

# 参数列表
parser.add_argument('--Size', default=20, type=int, metavar='SZ', help='学图片长宽')
parser.add_argument('--MAX_WIDTH', default=1000, type=int, metavar='MAX', help='原始图片最大宽度')
parser.add_argument('--Min_Area', default=2000, type=int, metavar='Min', help='车牌区域允许最大面积')
parser.add_argument('--PROVINCE_START', default=1000, type=int, metavar='START', help='省份字符开始位置')
parser.add_argument('--provinces', default=provinces, help='标签对应字符')
parser.add_argument('--cardtype', default=cardtype, help='标签对应车牌类型')
parser.add_argument('--Prefecture', default=Prefecture, help='省份字符开始位置')
parser.add_argument('--Pic_size', default=pic_size, help='')

# 解析参数
args = parser.parse_args()
```

3.2.2 图像预处理

输入图片如下：

如果图片过大，则根据预先设置好的参数调整图片分辨率

```
if pic_width > self.MAX_WIDTH:
            resize_rate = self.MAX_WIDTH / pic_width
            img = cv2.resize(img, (self.MAX_WIDTH, int(pic_hight * resize_rate)),
                             interpolation=cv2.INTER_AREA)  # 图片分辨率调整
```

使用 cv2.filter2D 函数和锐化内核来将图片锐化，然后用 cv2.GaussianBlur 函数使图片高斯去噪，再将图像转化为灰度图

```
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], np.float32)  # 定义一个核
img = cv2.filter2D(img, -1, kernel=kernel) # 锐化
blur = self.cfg["blur"]
# 高斯去噪
if blur > 0:
    img = cv2.GaussianBlur(img, (blur, blur), 0)
oldimg = img
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

此时得到的图像如下：

对图像进行一次开运算，将原图与开运算结果融合，然后用 cv2.threshold 函数将图像二值化，再用 cv2.canny 函数检测图像边缘，最后对检测出边缘的图像使用开运算和闭运算让图像边缘成为一个整体

```
kernel = np.ones((20, 20), np.uint8)
img_opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
img_opening = cv2.addWeighted(img, 1, img_opening, -1, 0);
ret, img_thresh = cv2.threshold(img_opening, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
img_edge = cv2.Canny(img_thresh, 100, 200)
kernel = np.ones((self.cfg["morphologyr"], self.cfg["morphologyc"]), np.uint8)
img_edge1 = cv2.morphologyEx(img_edge, cv2.MORPH_CLOSE, kernel)
img_edge2 = cv2.morphologyEx(img_edge1, cv2.MORPH_OPEN, kernel)
```

此时得到的图像如下：

3.2.3 车牌定位

使用 cv2.findContours 函数查找图像边缘整体形成的矩形区域，可能有很多，车牌就在其中一个矩形区域中

```
try:
    image, contours, hierarchy = cv2.findContours(img_edge2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
except ValueError:
    contours, hierarchy = cv2.findContours(img_edge2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = [cnt for cnt in contours if cv2.contourArea(cnt) > self.Min_Area]
```

接下来开始逐个排除不是车牌的矩形区域。

遍历之前得到的矩形区域，车牌形成的矩形区域长宽比在 2 到 5.5 之间，因此使用 cv2.minAreaRect()函数框选矩形区域计算长宽比，长宽比在 2 到 5.5 之间的可能是车牌，其余的矩形排除。然后使用 cv2.drawContours()函数将可能是车牌的区域在原图中框选出来，得到的图像如下：

矩形区域可能是倾斜的矩形，因此需要矫正，并且需要扩大矩形范围，避免车牌边缘被排除。然后得到左、高、右、低四个顶点的坐标

```
for rect in car_contours:
    # 创造角度，使得左、高、右、低拿到正确的值
    if rect[2] > -1 and rect[2] < 1:
        angle = 1
    else:
        angle = rect[2]
    # 扩大范围，避免车牌边缘被排除
    rect = (rect[0], (rect[1][0] + 5, rect[1][1] + 5), angle)
    box = cv2.boxPoints(rect)
    heigth_point = right_point = [0, 0]
    left_point = low_point = [pic_width, pic_hight]
    for point in box:
        if left_point[0] > point[0]:
            left_point = point
        if low_point[1] > point[1]:
            low_point = point
        if heigth_point[1] < point[1]:
            heigth_point = point
        if right_point[0] < point[0]:
            right_point = point
```

如果是正角度：

```
if left_point[1] <= right_point[1]:
    new_right_point = [right_point[0], heigth_point[1]]
    # 字符只是高度需要改变
    pts2 = np.float32([left_point, heigth_point, new_right_point])
    pts1 = np.float32([left_point, heigth_point, right_point])
    M = cv2.getAffineTransform(pts1, pts2)
    dst = cv2.warpAffine(oldimg, M, (pic_width, pic_hight))
    self.__point_limit(new_right_point)
    self.__point_limit(heigth_point)
    self.__point_limit(left_point)
    card_img = dst[int(left_point[1]):int(heigth_point[1]),
                               int(left_point[0]):int(new_right_point[0])]
    card_imgs.append(card_img)
```

如果是负角度：

```
elif left_point[1] > right_point[1]:
    new_left_point = [left_point[0], heigth_point[1]]
    # 字符只是高度需要改变
    pts2 = np.float32([new_left_point, heigth_point, right_point])
    pts1 = np.float32([left_point, heigth_point, right_point])
    M = cv2.getAffineTransform(pts1, pts2)
    dst = cv2.warpAffine(oldimg, M, (pic_width, pic_hight))
    self.__point_limit(right_point)
    self.__point_limit(heigth_point)
    self.__point_limit(new_left_point)
    card_img = dst[int(right_point[1]):int(heigth_point[1]),
                                int(new_left_point[0]):int(right_point[0])]
    card_imgs.append(card_img)
```

此时查看 card_imgs 中的第一个元素如下：

接下来开始使用**颜色定位**，将得到的每张矩形图像转化到 HSV 颜色空间，并且初始化每张图像的绿色、蓝色、黄色、黑色和白色的占有量为 0

```
for card_index, card_img in enumerate(card_imgs):
    green = yellow = blue = black = white = 0
    try:
        # 有转换失败的可能，原因来自于上面矫正矩形出错
        card_img_hsv = cv2.cvtColor(card_img, cv2.COLOR_BGR2HSV)

    except:
        card_img_hsv = None

    if card_img_hsv is None:
        continue
    row_num, col_num = card_img_hsv.shape[:2]
    card_img_count = row_num * col_num
```

基于 HSV 颜色模型可知色调 H 的取值范围为 0°～360°，从红色开始按逆时针方向计算，红色为 0°，绿色为 120°，蓝色为 240°。它们的补色是：黄色为 60°，青色为 180°，品红为 300°。查阅相关资料确定出下表：

	黄色	绿色	蓝色
H	11-34	35-99	100-124

根据此表计算每张矩形图像中各颜色的占有量：

```
for i in range(row_num):
    for j in range(col_num):
        H = card_img_hsv.item(i, j, 0)
        S = card_img_hsv.item(i, j, 1)
        V = card_img_hsv.item(i, j, 2)
        if 11 < H <= 34 and S > 34:
            yellow += 1
        elif 35 < H <= 99 and S > 34:
            green += 1
        elif 99 < H <= 124 and S > 34:
            blue += 1

        if 0 < H < 180 and 0 < S < 255 and 0 < V < 46:
            black += 1
        elif 0 < H < 180 and 0 < S < 43 and 221 < V < 225:
            white += 1
```

比较每个矩形颜色绿色、蓝色、黄色、黑色和白色的占有量，即可确定每一个矩形的颜色，将每一个矩形的颜色存进对应的 color 列表中

```
limit1 = limit2 = 0
if yellow * 2 >= card_img_count:
    color = "yellow"
    limit1 = 11
    limit2 = 34  # 有的图片有色偏偏绿
elif green * 2 >= card_img_count:
    color = "green"
    limit1 = 35
    limit2 = 99
elif blue * 2 >= card_img_count:
    color = "blue"
    limit1 = 100
    limit2 = 124  # 有的图片有色偏偏紫
elif black + white >= card_img_count * 0.7:
    color = "bw"
colors.append(color)
if limit1 == 0:
    continue
```

调用__accurate_place 函数，根据矩形的颜色再次进行定位，缩小边缘非车牌的边界

```
def __accurate_place(self, card_img_hsv, limit1, limit2, color):
    row_num, col_num = card_img_hsv.shape[:2]
    xl = col_num
    xr = 0
    yh = 0
    yl = row_num
    row_num_limit = self.cfg["row_num_limit"]
    # 绿色有渐变
    col_num_limit = col_num * 0.8 if color != "green" else col_num * 0.5
    for i in range(row_num):
        count = 0
        for j in range(col_num):
            H = card_img_hsv.item(i, j, 0)
            S = card_img_hsv.item(i, j, 1)
            V = card_img_hsv.item(i, j, 2)
            if limit1 < H <= limit2 and 34 < S and 46 < V:
                count += 1
        if count > col_num_limit:
            if yl > i:
                yl = i
            if yh < i:
                yh = i
    for j in range(col_num):
        count = 0
        for i in range(row_num):
            H = card_img_hsv.item(i, j, 0)
            S = card_img_hsv.item(i, j, 1)
            V = card_img_hsv.item(i, j, 2)
            if limit1 < H <= limit2 and 34 < S and 46 < V:
                count += 1
        if count > row_num - row_num_limit:
            if xl > j:
                xl = j
            if xr < j:
                xr = j
    return xl, xr, yh, yl
```

此时查看 card_imgs 中的第一个元素如下：

可以明显看到，比起之前的结果，车牌周围的非车牌区域更少，定位更加准确。至此车牌定位结束。

3.2.4 字符分割

总体而言，字符分割的过程如下：根据设定的阈值和图片直方图，找出波峰，利用找出的波峰，分隔图片。因为车牌中“ • ”也会产生一组波峰，因此将八组波峰中的第三组去除掉，即可得到每个字符的波峰，再根据每组波峰的宽度分割牌照图像得到每个字符的图像。

1、预先处理

首先我们将图片锐化，然后转为灰度图。但是注意到，车牌有黄、绿、蓝三种颜，其中黄、绿车牌字符比背景暗，与蓝车牌刚好相反。为了最后二值化时字体是白色，背景为黑色，则需要对黄绿车牌的图像进行反向，之后再进行二值化。

```
def __identification(self, card_imgs, colors,model,modelchinese):
        # 识别车牌中的字符
        result = {}
        predict_result = []
        roi = None
        card_color = None
        for i, color in enumerate(colors):
            if color in ("blue", "yellow", "green"):
                card_img = card_imgs[i]
                # old_img = card_img
                # 做一次锐化处理
                kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], np.float32)  # 锐化
                card_img = cv2.filter2D(card_img, -1, kernel=kernel)
                # cv2.imshow("custom_blur", card_img)

                # RGB转GARY
                gray_img = cv2.cvtColor(card_img, cv2.COLOR_BGR2GRAY)
                # cv2.imshow('gray_img', gray_img)

                # 黄、绿车牌字符比背景暗、与蓝车牌刚好相反，所以黄、绿车牌需要反向
                if color == "green" or color == "yellow":
                    gray_img = cv2.bitwise_not(gray_img)
                # 二值化
                ret, gray_img = cv2.threshold(gray_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
                # cv2.imshow('gray_img', gray_img)
```

2、使用直方图法在垂直和水平方向（x 和 y 方向）都找出波峰，用于分割字符，将车牌图片分割为一个个包含单个字符的图像。需要注意的是，设置的阈值不可以太大，不然会将 0 和 U 中间的间隔也作为峰值，将 U 或者 0 分成两半。

```
    # 查找水平直方图波峰
    x_histogram = np.sum(gray_img, axis=1)
    # 最小值
    x_min = np.min(x_histogram)
    # 均值
    x_average = np.sum(x_histogram) / x_histogram.shape[0]
    x_threshold = (x_min + x_average) / 2
    wave_peaks = self.__find_waves(x_threshold, x_histogram)
    if len(wave_peaks) == 0:
        continue

    # 认为水平方向，最大的波峰为车牌区域
    wave = max(wave_peaks, key=lambda x: x[1] - x[0])
    gray_img = gray_img[wave[0]:wave[1]]
    # cv2.imshow('gray_img', gray_img)

    # 查找垂直直方图波峰
    row_num, col_num = gray_img.shape[:2]
    # 去掉车牌上下边缘1个像素，避免白边影响阈值判断
    gray_img = gray_img[1:row_num - 1]
    # cv2.imshow('gray_img', gray_img)
    y_histogram = np.sum(gray_img, axis=0)
    y_min = np.min(y_histogram)
    y_average = np.sum(y_histogram) / y_histogram.shape[0]
    y_threshold = (y_min + y_average) / 5  # U和0要求阈值偏小，否则U和0会被分成两半

    wave_peaks = self.__find_waves(y_threshold, y_histogram)
    # print(wave_peaks)
```

find_waves 函数的实现：

```
def __find_waves(self, threshold, histogram):
        up_point = -1  # 上升点
        is_peak = False
        if histogram[0] > threshold:
            up_point = 0
            is_peak = True
        wave_peaks = []
        for i, x in enumerate(histogram):
            if is_peak and x < threshold:
                if i - up_point > 2:
                    is_peak = False
                    wave_peaks.append((up_point, i))
            elif not is_peak and x >= threshold:
                is_peak = True
                up_point = i
        if is_peak and up_point != -1 and i - up_point > 4:
            wave_peaks.append((up_point, i))
        return wave_peaks
```

3、后续处理以及分割

我们需要对得到的波峰进行一些处理，首先是判断分割是否正确，车牌字符数最少应该有 6 个，少于六个说明分割不成功。其次是判断左侧波峰是否是车牌边缘的部分，如果是则将其去掉。最后是因为车牌中“ • ”也会产生一组波峰，因此将八组波峰中的第三组去除掉。然后就可以根据波峰进行图像的分割。

```
# 车牌字符数应大于6
if len(wave_peaks) <= 6:
    #   print(wave_peaks)
    continue

wave = max(wave_peaks, key=lambda x: x[1] - x[0])
max_wave_dis = wave[1] - wave[0]
# 判断是否是左侧车牌边缘
if wave_peaks[0][1] - wave_peaks[0][0] < max_wave_dis / 3 and wave_peaks[0][0] == 0:
    wave_peaks.pop(0)


# 去除车牌上的分隔点
point = wave_peaks[2]
if point[1] - point[0] < max_wave_dis / 3:
    point_img = gray_img[:, point[0]:point[1]]
    if np.mean(point_img) < 255 / 5:
        wave_peaks.pop(2)

if len(wave_peaks) <= 6:
    # print("peak less 2:", wave_peaks)
    continue
# print(wave_peaks)
# 分割牌照字符
part_cards = self.__seperate_card(gray_img, wave_peaks)
```

Separate_card 函数的实现：

```
def __seperate_card(self, img, waves):
        part_cards = []
        for wave in waves:
            part_cards.append(img[:, wave[0]:wave[1]])
        return part_cards
```

3.2.5 字符识别

字符识别中用到了使用 SVM 训练出的识别模型。对于识别过程，首先对分割出的图片进行边缘填充和图片缩放（缩放为模型可以识别的标准大小）。

```
for i, part_card in enumerate(part_cards):
    # 可能是固定车牌的铆钉
    if np.mean(part_card) < 255 / 5:
        continue
    part_card_old = part_card
    w = abs(part_card.shape[1] - self.SZ) // 2

    # 边缘填充
    part_card = cv2.copyMakeBorder(part_card, 0, 0, w, w, cv2.BORDER_CONSTANT, value=[0, 0, 0])
    # cv2.imshow('part_card', part_card)

    # 图片缩放 (self.SZ为训练图片的标准长宽)
    part_card = cv2.resize(part_card, (self.SZ, self.SZ), interpolation=cv2.INTER_AREA)
    # cv2.imshow('part_card', part_card)
```

然后使用训练的模型进行识别：

```
part_card = SVM_Train.preprocess_hog([part_card])

if i == 0:  # 识别汉字
    resp = self.modelchinese.predict(part_card)  # 匹配样本
    charactor = self.provinces[int(resp[0]) - self.PROVINCE_START]
    # print(charactor)
else:  # 识别字母
    resp = self.model.predict(part_card)  # 匹配样本
    charactor = chr(resp[0])
    # print(charactor)
```

最后判断识别出的最后一个数字是否是车牌右边缘（也许将车牌右边缘识别为 1）

```
# 判断最后一个数是否是车牌边缘，假设车牌边缘被认为是1
if charactor == "1" and i == len(part_cards) - 1:
    if color == 'blue' and len(part_cards) > 7:
        if part_card_old.shape[0] / part_card_old.shape[1] >= 7:  # 1太细，认为是边缘
            continue
    elif color == 'blue' and len(part_cards) > 7:
        if part_card_old.shape[0] / part_card_old.shape[1] >= 7:  # 1太细，认为是边缘
            continue
    elif color == 'green' and len(part_cards) > 8:
        if part_card_old.shape[0] / part_card_old.shape[1] >= 7:  # 1太细，认为是边缘
            continue
```

3.2.6 字符识别的训练过程与程序主程序

训练过程中我们使用了 opencv 自带的 SVM 框架。从 github 上下载了一个数字与部分中文汉字的训练图片集。然后使用 opencv 的 SVM 框架进行训练和模型的保存

1、定义 SVM 模型
调用 opencv 的 model 来创建一个 SVM 模型，参数为 gamma 值和 C 值。

```
class SVM(StatModel):
    def __init__(self, C=1, gamma=0.5):
        self.model = cv2.ml.SVM_create()
        self.model.setGamma(gamma)
        self.model.setC(C)
        self.model.setKernel(cv2.ml.SVM_RBF)
        self.model.setType(cv2.ml.SVM_C_SVC)

    # 训练svm
    def train(self, samples, responses):
        self.model.train(samples, cv2.ml.ROW_SAMPLE, responses)

    # 字符识别
    def predict(self, samples):
        r = self.model.predict(samples)
        return r[1].ravel()
```

从 opencv 官网（https://github.com/opencv/opencv/blob/master/samples/python/digits.py）中得到了字符的 SVM 训练所需要做的预处理的方法如下：

抗色偏（deskew）：

```
# 来自opencv官方的sample，用于svm训练预处理
def deskew(img):
    m = cv2.moments(img)
    if abs(m['mu02']) < 1e-2:
        return img.copy()
    skew = m['mu11'] / m['mu02']
    M = np.float32([[1, skew, -0.5 * SZ * skew], [0, 1, 0]])
    img = cv2.warpAffine(img, M, (SZ, SZ), flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
    return img
```

计算 hog 特征：

```python
# 来自opencv官方的sample，用于svm训练预处理
def preprocess_hog(digits):
    samples = []
    for img in digits:
        gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
        gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
        mag, ang = cv2.cartToPolar(gx, gy)
        bin_n = 16
        bin = np.int32(bin_n * ang / (2 * np.pi))
        bin_cells = bin[:10, :10], bin[10:, :10], bin[:10, 10:], bin[10:, 10:]
        mag_cells = mag[:10, :10], mag[10:, :10], mag[:10, 10:], mag[10:, 10:]
        hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells, mag_cells)]
        hist = np.hstack(hists)

        # transform to Hellinger kernel
        eps = 1e-7
        hist /= hist.sum() + eps
        hist = np.sqrt(hist)
        hist /= norm(hist) + eps

        samples.append(hist)
    return np.float32(samples)
```

然后就是直接的模型训练过程，将英文字符和数字训练集保存在本地的 chars 文件夹，中文汉字保存在 charsChinese 文件夹中，训练时使用。

英文数字训练部分：

```python
def train_svm(path):
    # 识别英文字母和数字
    Model = SVM(C=1, gamma=0.5)
    # 识别中文
    Modelchinese = SVM(C=1, gamma=0.5)
    # 英文字母和数字部分训练
    chars_train = []
    chars_label = []

    for root, dirs, files in os.walk(os.path.join(path,'chars')):
        if len(os.path.basename(root)) > 1:
            continue
        root_int = ord(os.path.basename(root))
        for filename in files:
            print('input: {}'.format(filename))
            filepath = os.path.join(root, filename)
            digit_img = cv2.imread(filepath)
            digit_img = cv2.cvtColor(digit_img, cv2.COLOR_BGR2GRAY)
            chars_train.append(digit_img)
            chars_label.append(root_int)

    chars_train = list(map(deskew, chars_train))
    chars_train = preprocess_hog(chars_train)
    chars_label = np.array(chars_label)
    Model.train(chars_train, chars_label)

    if not os.path.exists("svm.dat"):
        # 保存模型
        Model.save("svm.dat")
    else:
        # 更新模型
        os.remove("svm.dat")
        Model.save("svm.dat")
```

汉字训练部分：

```
chars_train = []
    chars_label = []

    for root, dirs, files in os.walk(os.path.join(path,'charsChinese')):
        if not os.path.basename(root).startswith("zh_"):
            continue
        pinyin = os.path.basename(root)
        index = provinces.index(pinyin) + PROVINCE_START + 1  # 1是拼音对应的汉字
        for filename in files:
            print('input: {}'.format(filename))
            filepath = os.path.join(root, filename)
            digit_img = cv2.imread(filepath)
            digit_img = cv2.cvtColor(digit_img, cv2.COLOR_BGR2GRAY)
            chars_train.append(digit_img)
            chars_label.append(index)
    chars_train = list(map(deskew, chars_train))
    chars_train = preprocess_hog(chars_train)
    chars_label = np.array(chars_label)
    Modelchinese.train(chars_train, chars_label)

    if not os.path.exists("svmchinese.dat"):
        # 保存模型
        Modelchinese.save("svmchinese.dat")
    else:
        # 更新模型
        os.remove("svmchinese.dat")
        Modelchinese.save("svmchinese.dat")
```

主程序部分：

程序主程序读取输入的图片，并利用训练好的模型去识别车牌，最后输出计算用时，车牌的字符、车牌颜色、对应的地级市，车牌所在的图片坐标。

```
def VLPR(self, car_pic):
        result = {}
        start = time.time()
        # 初始化模型
        self.model = SVM(C=1, gamma=0.5)
        if os.path.exists("svm.dat"):
            self.model.load("svm.dat")
        else:
            raise FileNotFoundError('svm.dat')
        self.modelchinese = SVM(C=1, gamma=0.5)
        if os.path.exists("svmchinese.dat"):
            self.modelchinese.load("svmchinese.dat")
        else:
            raise FileNotFoundError('svmchinese.dat')

        card_imgs, colors = self.__preTreatment(car_pic)
        if card_imgs is []:
            return
        else:
            predict_result, roi, card_color = self.__identification(card_imgs, colors,self.model,self.modelchinese)
            if predict_result != []:
                result['UseTime'] = round((time.time() - start), 2)
                result['InputTime'] = time.strftime("%Y-%m-%d %H:%M:%S")
                result['Type'] = self.cardtype[card_color]
                result['List'] = predict_result
                result['Number'] = ''.join(predict_result[:2]) + '·' + ''.join(predict_result[2:])
                try:
                    result['From'] = ''.join(self.Prefecture[result['List'][0]][result['List'][1]])
                except:
                    result['From'] = '未知'
                result['Picture'] = roi
                return result
            else:
                return None
```

车牌对应的地级市不属于识别的内容，只要识别的省简称和对应代表城市的字母正确，则可以输出对应的地级市，这个内容是机械的对应。

```
Prefecture = {
    "冀": {"A": ["河北省", "石家庄市"], "B": ["河北省", "唐山市"], "C": ["河北省", "秦皇岛市"], "D": ["河北省", "邯郸市"], "E": ["河北省", "邢台市"],
         "F": ["河北省", "保定市"], "G": ["河北省", "张家口市"], "H": ["河北省", "承德市"], "J": ["河北省", "沧州市"], "R": ["河北省", "廊坊市"],
         "S": ["河北省", "沧州市"], "T": ["河北省", "衡水市"]},
    "辽": {"A": ["辽宁省", "沈阳市"], "B": ["辽宁省", "大连市"], "C": ["辽宁省", "鞍山市"], "D": ["辽宁省", "抚顺市"], "E": ["辽宁省", "本溪市"],
         "F": ["辽宁省", "丹东市"], "G": ["辽宁省", "锦州市"], "H": ["辽宁省", "营口市"], "J": ["辽宁省", "阜新市"], "K": ["辽宁省", "辽阳市"],
         "L": ["辽宁省", "盘锦市"], "M": ["辽宁省", "铁岭市"], "N": ["辽宁省", "朝阳市"], "P": ["辽宁省", "葫芦岛市"]},
    "皖": {"A": ["安徽省", "合肥市"], "B": ["安徽省", "芜湖市"], "C": ["安徽省", "蚌埠市"], "D": ["安徽省", "淮南市"], "E": ["安徽省", "马鞍山市"],
         "F": ["安徽省", "淮北市"], "G": ["安徽省", "铜陵市"], "H": ["安徽省", "安庆市"], "J": ["安徽省", "黄山市"], "K": ["安徽省", "阜阳市"],
         "L": ["安徽省", "宿州市"], "M": ["安徽省", "滁州市"], "N": ["安徽省", "六安市"], "P": ["安徽省", "宣城市"], "Q": ["安徽省", "巢湖市"],
         "R": ["安徽省", "池州市"], "S": ["安徽省", "亳州市"]},
```

第四章仿真/实验结果与分析

4.1 数据集

1、字符识别的数据集例子如下。

数字：

字母：

汉字：

2、含有车牌的汽车图片例子如下：


京A C9331

、


2013-04-10 12:02:31
西门出口
川A·A662F

4.2 评价与测试

4.2.1 评价标准

总分 10 分。若完全正确识别车牌的位置则为 10 分。每错误识别一个字符-1 分，颜色判断错误-3 分。

测试用例的选择：报告中选择的测试用例中包含了三种不同颜色的车牌和不同角度的车牌

4.2.2 测试与测试结果评价

1、测试样例一(特性：右倾斜、蓝色车牌)

识别结果：

```
{'UseTime': 0.35, 'InputTime': '2021-07-14 15:52:28', 'Type': '蓝色牌照',
'List': ['晋', 'A', 'A', '2', '6', '6', 'G'], 'Number': '晋A·A266G',
```

。

实际得分：9，将吉林的吉识别为了山西的晋。

2、测试样例二（特性：正面、黄色车牌）

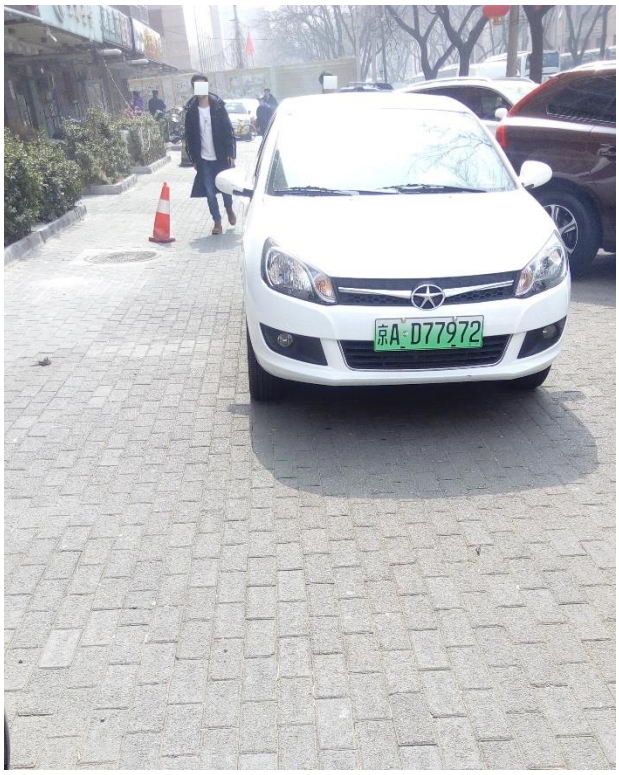
识别结果：

{'UseTime': 0.55, 'InputTime': '2021-07-14 15:58:53', 'Type': '黄色牌照',

'List': ['京', 'A', 'C', '9', '3', '3', '1'], 'Number': '京A·C9331',

实际得分：10，完全正确

3、测试样例三（特性：右倾斜，绿色车牌）



识别结果：

{'UseTime': 0.91, 'InputTime': '2021-07-14 16:02:36', 'Type': '绿色牌照',

'List': ['京', 'A', 'D', '7', '7', '9', '7', '2'], 'Number': '京A·D77972',

实际得分：10，完全正确

4、测试样例四（特性：右倾斜，蓝色车牌，模糊）

识别结果：

{'UseTime': 0.31, 'InputTime': '2021-07-14 16:10:12', 'Type': '蓝色牌照',

'List': ['皖', 'A', '8', '7', '2', '7', '1', 'H'], 'Number': '皖A·87271H',

实际得分：9，多识别出了一个 H 字母，也许是因为本测试用例较为模糊。

5、测试用例五（特性：左倾斜、蓝色车牌）

识别结果：

{'UseTime': 0.62, 'InputTime': '2021-07-14 16:57:49', 'Type': '蓝色牌照',

'List': ['蒙', 'A', 'G', 'X', '4', '6', '8'], 'Number': '蒙A·GX468',

实际得分：10，完全正确

测试用例 6：（特性：右倾斜、黄色车牌）

识别结果：

{'UseTime': 0.72, 'InputTime': '2021-07-14 17:11:24', 'Type': '黄色牌照',

'List': ['京', 'E', '5', '1', '6', '1', '9'], 'Number': '京E·51619',

实际得分：10，完全正确

测试用例 7：（特性：正面，蓝色车牌）

识别结果：

{'UseTime': 0.31, 'InputTime': '2021-07-14 17:16:10', 'Type': '蓝色牌照',

'List': ['赣', 'E', '2', 'R', '6', '1'], 'Number': '赣E·2R61',

实际得分：9，将一个 1 错误的当成车牌的右边缘而导致没有识别。

第五章 总结与展望

本次的实验中大量用到了 python 和 opencv 对于图像处理的能力。虽然有过计算机视觉课程的经验，但是这次用到的 SVM 算法和上次计算机视觉课使用的 PCA 算法不同。导致对于图片的预处理部分要求有些不同。因为缺乏对图片的抗色偏（deskew）和提前计算 hog 特征的预处理，导致识别效果一度很不理想，后来去 opencv 的 github 官网查看了 python SVM 的 sample，从里面学习到了这两个预处理步骤。效果就好了很多。

除此之外，对于图片中车牌的识别和分割才是本次实验的重点和难点。单纯的利用机器学习和数据挖掘的手段去识别数字或者是汉字的话有很多的参考和教程。但是车牌识别这整一个工作中，首先需要把车牌识别出来，然后再将车牌中的字符分割开，才能最终进入字符识别的过程。识别车牌这个过程运用了识别矩形的比例加上颜色的判断，方法算式比较巧妙，这也是在一篇论文中得到的灵感。在分割字符这个步骤中遇到了很多的困难。我们使用的是波峰法，一开始常见的问题是因为车牌上下左右边缘的白框造成波峰数量过多。之后由于阈值的设置过大，导致 0 和 U 这样的字符被分作两个波峰进而被分割为两个图片。再然后遇到的问题为识别数字时，将车牌上的铆钉识别进去，以及将车牌右边缘的白框识别为数字 1。后来通过改进排除了这些错误的判断。

诚实而言，本次实验还有不足的地方。为了避免将车牌右边缘的白框识别为数字 1 的改动，在某些车牌距离镜头较近的情况下，也许会将正常的数字 1 也排除（比如测试用例 7）。同时在某些特殊的角度下，也许会将数字 9 识别为字母 G。但是总体而言，我们对这次实验的效果还是满意的。也学习到了非常多的 opencv 和 SVM 的知识。

本次实验学习了大量 opencv 的函数用法和图像处理相关的内容。同时学习使用了 SVM 算法的模型训练和使用过程。并且学习提升了寻找数据集和训练集的能力。一并了解了中文论文的搜索，从前搜索的论文多为国外理论论文居多。类似于这类的小型实验的实践类论文，中文论文的搜索也许会更占优势。

希望以后还能用更多的模型和算法，通过机器学习和数据挖掘的方法，解决一个个实践的问题，做出实质性的成果。

参考文献：

[1]朱克佳, 郝庆华, 李世勇,等. 车牌识别综述[J]. 现代信息科技, 2018, 002(005):4-6.

[2]张志佳, 石佳. 车辆牌照识别系统综述[J]. 中国科技博览, 2012, 000(013):103-103.

[3]王兰, 吴谨. 一种改进的 Canny 边缘检测算法[J]. 微计算机信息, 2010, 26(002):198-199.

[4]黄文杰. 基于投影的车牌字符分割方法[J]. 现代计算机, 2009, 000(008):57-60.

[5]吕文强. 基于 Adaboost 和 SVM 的车牌识别方法研究[D]. 南京理工大学, 2013.

[6]李红林. 基于改进的 HSV 颜色模型及颜色均值对的车牌检测与定位[J]. 云南民族大学学报:自然科学版, 2009(03):268-272.

鲁沛 18342066：

- 研究课题的确立和任务的分配
- 图像预处理（3.2.2）的代码和报告
- 车牌定位（3.2.3）的代码和报告
- 报告第一章概述/引言的编写
- 报告第二章相关工作综述的主要编写

马靖成 18342071：

- 图像分割部分（3.2.4）的代码和报告
- 图像识别部分（3.2.5）的代码和报告
- 图像识别的训练部分以及主程序（3.2.6）的代码和报告
- 报告第五章总结与展望的编写
- 报告第二章相关工作中与图像识别有关的部分（SVM 等）

陆世炜 18342068：

- 数字识别和字母、汉字识别所需的训练集的寻找收集
- 车牌照片的寻找收集
- 实验的运行与测试
- 报告第四章实验结果与分析的编写