杭州電子科技大學

HANGZHOU DIANZI UNIVERSITY

数字图像小作业

——车牌识别的数据处理

学　院：计算机学院

专　业：计算机科学与技术

学　号：22050102

姓　名：赵妍

目　录

一. 程序简介

车牌识别系统是一种应用计算机视觉技术来自动识别车辆车牌号码的系统。这类系统广泛应用于交通监控、停车场管理、高速公路收费以及城市交通管理等领域。本程序结合数字图像处理的相关知识，基于python与OpenCV实现了对图片中车牌的数据的简单处理。

二. 程序设计思路

2.1 图像预处理

本程序首先根据路径读取输入图像，接着对输入的图像进行基本的预处理，预处理是指对采集的车牌图像进行大小规范化、降噪、彩色图像灰度化、腐蚀膨胀等，使图片牌照区域质量得到改善，保留车牌区域信息，去除噪声。在本程序中，限定图像的宽度不能超过1000像素，若图像大小超出了限制的大小，则等比地对图像进行缩放处理。获得符合条件大小的图像后，再使用高斯模糊函数*cv2.GaussianBlur*对图像进行降噪处理，减少噪声对后续操作的干扰。最后，将得到的图像进行灰度化，并保存到固定路径，方便后续操作的进行。预处理的效果如下图显示：

原始图像

预处理后的图像

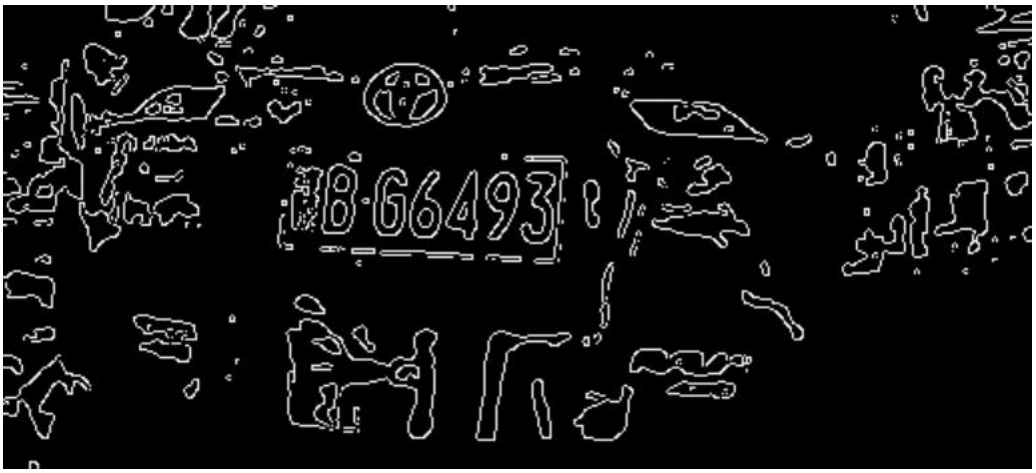
2.2 图像轮廓提取

在预处理完图像后，下一步是对图像轮廓的提取。为使车牌轮廓 特征更加明显，先对图像进行形态学处理，即先对图像进行一次开运 算，去除噪声，除去孤立的小点，毛刺和小桥， 再将所得到的图像与原先的灰度图像进行加权融合，强调原图像中的某些特征，最终得到如下图所示的结果：

经过开运算后的图像

融合后的图像

接下来我们要提取图像的边缘。首先使用Otsu方法对图像进行阈值处理，将图像转换为二值图像（img_thresh），像素值是0或255。接下来，对阈值处理后的图像应用 Canny 边缘检测算法。边缘检测结果如下图所示。



边缘检测后的图像

经过边缘检测后的图像，我们再次对其进行形态学处理，先进行一次闭运算填补小孔和裂缝，再进行一次开运算除去小点及毛刺，最终得到较为明显的区域块。形态学处理实现效果如下：、

形态学处理后图像

2.3车牌的提取

接下来任务是在所得到的轮廓中获得车牌的轮廓。首先定义蓝色、黄色和绿色在 HSV 颜色空间中的阈值范围。这是因为车牌通常是这些颜色。然后将图像从 BGR 颜色空间转换到 HSV。利用cv2.inRange函数和设定的颜色阈值过滤出特定颜色。用于识别图像中对应颜色的区域。接下来进行图像融合和处理，通过cv2.bitwise_and操作结合掩膜，提取出这些颜色的区域，并将提取结果转换为灰度图像。结果如下图：

颜色过滤车牌

灰度图像

然后使用闭运算和开运算进行形态学变换，轮廓检测，使用img_findContours函数在处理后的图像上查找轮廓。找出可能包含车牌的区域。对于每个检测到的轮廓，获取其最小外接矩形，并在原始图像上绘制轮廓。结果如下图：

定位到的车牌

三. 程序代码

本程序的代码如下：

```
import os
import cv2
from PIL import Image
import numpy as np
import function
import img_recognition
SZ = 20  # 训练图片长宽
MAX_WIDTH = 1000  # 原始图片最大宽度
# Min_Area = 2000  # 车牌区域允许最大面积
PROVINCE_START = 1000


class StatModel(object):
    def load(self, fn):
        self.model = self.model.load(fn)

    def save(self, fn):
        self.model.save(fn)

#车牌识别中的机器学习任务
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)#核类型（RBF核）
        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):
        #预测输入样本的类别
        r = self.model.predict(samples)
        return r[1].ravel()


class CardPredictor:
    def __init__(self):
        pass

    def train_svm(self):
        # 识别英文字母和数字
        self.model = SVM(C=1, gamma=0.5)
        # 识别中文
```

```
48.         self.modelchinese = SVM(C=1, gamma=0.5)
49.         if os.path.exists("svm.dat"):
50.             self.model.load("svm.dat")
51.         if os.path.exists("svmchinese.dat"):
52.             self.modelchinese.load("svmchinese.dat")
53.
54.     def img_first_process(self, car_pic_file):
55.         """
56.         :param car_pic_file: 图像文件
57.         :return:已经处理好的图像文件，二值图像 原图像文件
58.         """
59.         if type(car_pic_file) == type(""):
60.             img = function.img_read(car_pic_file)   #读取文件
61.         else:
62.             img = car_pic_file
63.         # 优化处理速度和效果，因为过大的图像可能不必要地增加计算负担
64.         pic_hight, pic_width = img.shape[:2]  #取彩色图片的高、宽
65.         if pic_width > MAX_WIDTH:
66.             resize_rate = MAX_WIDTH / pic_width
67.             # 缩小图片
68.             img = cv2.resize(img, (MAX_WIDTH, int(pic_hight * resize_rate)), interpolation=cv2.INTER_AR
    EA)
69.         # 关于interpolation 有几个参数可以选择:
70.         # cv2.INTER_AREA - 局部像素重采样，适合缩小图片。
71.         # cv2.INTER_CUBIC和 cv2.INTER_LINEAR 更适合放大图像，其中INTER_LINEAR为默认方法。
72.
73.         img = cv2.GaussianBlur(img, (5, 5), 0)
74.         # 高斯滤波是一种线性平滑滤波，对于除去高斯噪声有很好的效果
75.         # 0 是指根据窗口大小（ 5,5 ）来计算高斯函数标准差
76.
77.
78.         oldimg = img
79.         # 转化成灰度图像
80.         # 转换颜色空间 cv2.cvtColor
81.         # BGR ---> Gray  cv2.COLOR_BGR2GRAY
82.         # BGR ---> HSV  cv2.COLOR_BGR2HSV
83.         img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
84.
85.         cv2.imwrite("./tmp/img_gray.jpg", img)
86.
87.         #ones()返回一个全1的n维数组
88.         Matrix = np.ones((20, 20), np.uint8)
89.
90.         # 开运算:先进行腐蚀再进行膨胀就叫做开运算。它被用来去除噪声。 cv2.MORPH_OPEN
91.         img_opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, Matrix)
92.         cv2.imwrite("./tmp/img_opening.jpg", img_opening)
93.         # 图片叠加与融合
94.         # g (x) = (1 - α)f0 (x) + αf1 (x)   α→（0，1）不同的α值可以实现不同的效果
95.         # 图像融合
96.         img_opening = cv2.addWeighted(img, 1, img_opening, -1, 0)
97.         cv2.imwrite("./tmp/img_ronghe.jpg", img_opening)
98.         # 创建20*20的元素为1的矩阵 开操作，并和img重合
99.
```

```
100.
101.         # Otsu's二值化
102.         ret, img_thresh = cv2.threshold(img_opening, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
103.         # Canny 边缘检测
104.         # 较大的阈值2用于检测图像中明显的边缘  一般情况下检测的效果不会那么完美，边缘检测出来是断断续续的
105.         # 较小的阈值1用于将这些间断的边缘连接起来
106.         img_edge = cv2.Canny(img_thresh, 100, 200)
107.         cv2.imwrite("./tmp/img_edge.jpg", img_edge)
108.
109.         Matrix = np.ones((4, 19), np.uint8)
110.         # 闭运算:先膨胀再腐蚀，闭运算可以填充图像中的小孔，同时也可以连接相邻的物体
111.         img_edge1 = cv2.morphologyEx(img_edge, cv2.MORPH_CLOSE, Matrix)
112.         # 开运算：开运算可以消除图像中的小孔，同时也可以分离相邻的物体。
113.         img_edge2 = cv2.morphologyEx(img_edge1, cv2.MORPH_OPEN, Matrix)
114.         cv2.imwrite("./tmp/img_xingtai.jpg", img_edge2)
115.         return img_edge2, oldimg
116.
117.     def img_only_color(self, filename, oldimg, img_contours):
118.         """
119.         :param filename: 图像文件
120.         :param oldimg: 原图像文件
121.         img_contours: 图像轮廓
122.         :return: 识别到的字符、定位的车牌图像、车牌颜色
123.         """
124.         pic_hight, pic_width = img_contours.shape[:2] # #取彩色图片的高、宽
125. #三个颜色范围的阈值
126.         lower_blue = np.array([100, 110, 110])
127.         upper_blue = np.array([130, 255, 255])
128.         lower_yellow = np.array([15, 55, 55])
129.         upper_yellow = np.array([50, 255, 255])
130.         lower_green = np.array([50, 50, 50])
131.         upper_green = np.array([100, 255, 255])
132.
133.         # BGR ---> HSV
134.         hsv = cv2.cvtColor(filename, cv2.COLOR_BGR2HSV)
135.         # 利用cv2.inRange函数设阈值，去除背景部分
136.         # 参数1：原图
137.         # 参数2：图像中低于值，图像值变为0
138.         # 参数3：图像中高于值，图像值变为0
139.         mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
140.         mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
141.         mask_green = cv2.inRange(hsv, lower_yellow, upper_green)
142.
143.         # 图像算术运算  按位运算 按位操作有： AND， OR， NOT， XOR 等
144.         output = cv2.bitwise_and(hsv, hsv, mask=mask_blue + mask_yellow + mask_green)
145.         # 根据阈值找到对应颜色
146.         cv2.imwrite("./tmp/img_corlor.jpg", output)
147.         output = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)
148.         cv2.imwrite("./tmp/img_huidu.jpg", output)
149.         Matrix = np.ones((20, 20), np.uint8)
150.         #使用一个 20x20 的卷积核
151.         img_edge1 = cv2.morphologyEx(output, cv2.MORPH_CLOSE, Matrix)  #闭运算
152.         img_edge2 = cv2.morphologyEx(img_edge1, cv2.MORPH_OPEN, Matrix) #开运算
```

```
153.  #使用function.img_findContours函数找到边缘轮廓
154.          card_contours = function.img_findContours(img_edge2)
155.          # for rect in card_contours :
156.          #     box = cv2.boxPoints(rect)  # 获取轮廓的四个角点
157.          #     box = np.int0(box)  # 转换为整数
158.          #     cv2.drawContours(oldimg, [box], 0, (0, 255, 0), 2)  # 在原始图像上绘制轮廓
159.          # cv2.imwrite("./tmp/img_contour.jpg",oldimg)
160.          # function.img_Transform函数对轮廓进行变换。
161.          card_imgs = function.img_Transform(card_contours, oldimg, pic_width, pic_hight)
162.          i=0
163.          # for item in card_imgs:
164.          #     i=i+1
165.          #     cv2.imwrite("./tmp/img_jiaozheng{}.jpg".format(i), item)
166.          # 使用function.img_color函数获取颜色信息。
167.          colors, car_imgs = function.img_color(card_imgs)
168.          # cv2.imwrite("./tmp/img_yansemay.jpg",car_imgs[0])
169.          # cv2.imwrite("./tmp/img_yansemay1.jpg", car_imgs[1])
170.          predict_result = []
171.          predict_str = ""
172.          roi = None
173.          card_color = None
174.          # 遍历颜色列表，找到蓝色、黄色和绿色的颜色，然后对对应的卡片图像进行灰度转换。如果灰度转换失败，会打印
175.          # "gray转换失败"。
176.          for i, color in enumerate(colors):
177.
178.              if color in ("blue", "yello", "green"):
179.                  card_img = card_imgs[i]
180.                  cv2.imwrite("./tmp/img_chepailast.jpg",card_img)
181.                  try:
182.                      gray_img = cv2.cvtColor(card_img, cv2.COLOR_BGR2GRAY)
183.                  except:
184.                      print("gray转换失败")
185.
186.                  # 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
187.                  if color == "green" or color == "yello":
188.                      gray_img = cv2.bitwise_not(gray_img)#二进制位运算
189.  #执行该代码后，ret表示阈值计算的结果，gray_img为经过二值化处理后的图像。
190.                  ret, gray_img = cv2.threshold(gray_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
191.                  cv2.imwrite("./tmp/img_yansegray.jpg", gray_img)
```

四. 程序后续改进

6.1程序缺陷

本程序只实现了车牌数据处理，但并没有实现识别车牌号。

6.2后续改进

后续考虑从以下方面进行改进：

1. 添加识别车牌号和颜色的功能

2. 做成可视化界面，增强用户友好性