基于数字图像处理技术实现的车牌识别系统

刘栉锐 10225101433

鄢祺昊 10225101424

袁城林 10225101422

摘要：我们的程序主要实现了一个车牌识别系统。它读取一张车牌图片，通过图像处理技术提取车牌区域，然后对车牌区域进行字符分割，最终使用 OCR（光学字符识别）技术识别车牌号码。实现了一个完整的车牌识别流程，主要技术包括图像预处理、车牌区域定位、车牌字符分割和最终的字符识别。这些技术在实际应用中可以帮助实现自动化的车牌识别，广泛应用于停车场管理、交通监控等领域。使用技术：opencv，tkinter,ocr ,matplotlib

正文：

代码实现与分析

导入库和字体文件

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from matplotlib import font_manager
import ddddocr
import threadsafe_tkinter as tk
from tkinter import filedialog
```

numpy 用于进行数值计算和矩阵操作。
matplotlib.pyplot 用于绘制图形和显示图像。
matplotlib.font_manager 用于管理和加载字体文件。
ddddocr 是一个 OCR 库，用于识别字符。
Tkinter 实现 GUI 界面

```
font = font_manager.FontProperties(fname=r".\OPPOSans-Heavy.ttf")
```

加载自定义字体文件 OPPOSans，用于在图像显示时使用特定字体。

定义 Get_license 类
图像拉伸函数[1]

```
class Get_license():
    def stretch(self, img):
        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
```

该方法用于拉伸图像的灰度值范围，使图像的对比度更加明显。
maxi 和 mini 是图像的最大和最小灰度值。
循环遍历每个像素，通过线性变换将每个像素值拉伸到 0 到 255 范围内。

二值化处理函数

```
#二值化处理函数
def dobinaryzation(self, img):
    # 使用OpenCV的自动阈值方法
    ret, thresh = cv2.threshold(img, thresh: 0, maxval: 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return thresh
```

该方法用于将灰度图像转换为二值图像。
cv2.threshold 用于图像二值化。

寻找矩形的轮廓

```
def find_rectangle(self, contour):
    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)]
```

该方法用于找到轮廓的边界矩形。
contour 是一系列轮廓点，通过找到最小和最大坐标来确定矩形的边界

定位车牌号

```python
def locate_license(self, img, afterimg):
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    block = []
    for c in contours:
        r = self.find_rectangle(c)
        a = (r[2] - r[0]) * (r[3] - r[1])
        s = (r[2] - r[0]) * (r[3] - r[1])
        block.append([r, a, s])
    block = sorted(block, key=lambda b: b[1])[-3:]

    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]]
        hsv = cv2.cvtColor(b, cv2.COLOR_BGR2HSV)
        lower = np.array([100, 50, 50])
        upper = np.array([140, 255, 255])
        mask = cv2.inRange(hsv, lower, upper)
        w1 = 0
        for m in mask:
            w1 += m / 255
        w2 = 0
        for n in w1:
            w2 += n
        if w2 > maxweight:
            maxindex = i
            maxweight = w2

    return block[maxindex][0]
```

该方法用于定位车牌区域。
使用 cv2.findContours 找到所有的外部轮廓。
计算每个轮廓的面积和长度比，选出面积最大的三个区域。
使用颜色识别判断最可能是车牌的区域，并返回该区域的边界。

预处理函数

```python
def find_license(self, img):
    m = 400 * img.shape[0] / img.shape[1]
    img = cv2.resize(img, (400, int(m)), interpolation=cv2.INTER_CUBIC)
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    stretchedimg = self.stretch(gray_img)
    r = 16
    h = w = r * 2 + 1
    kernel = np.zeros((h, w), np.uint8)
    cv2.circle(kernel, (r, r), r, 1, -1)
    openingimg = cv2.morphologyEx(stretchedimg, cv2.MORPH_OPEN, kernel)
    strtimg = cv2.absdiff(stretchedimg, openingimg)
    binaryimg = self.dobinaryzation(strtimg)
    canny = cv2.Canny(binaryimg, binaryimg.shape[0], binaryimg.shape[1])
    kernel = np.ones((5, 19), np.uint8)
    closingimg = cv2.morphologyEx(canny, cv2.MORPH_CLOSE, kernel)
    openingimg = cv2.morphologyEx(closingimg, cv2.MORPH_OPEN, kernel)
    kernel = np.ones((11, 5), np.uint8)
    openingimg = cv2.morphologyEx(openingimg, cv2.MORPH_OPEN, kernel)
    rect = self.locate_license(openingimg, img)
    return rect, img
```

该方法用于预处理图像并定位车牌。
压缩图像到固定宽度（400 像素），并相应地调整高度。
将 BGR 图像转换为灰度图像。
对灰度图像进行灰度拉伸，以增强对比度。
使用自定义的开运算去除噪声。
边缘检测与形态学操作：
通过 Canny 边缘检测器找到边缘。
使用闭运算和开运算结合来消除小的区域，并保留车牌区域。
车牌定位：
使用 locate_license 函数来定位车牌区域。

图像分割函数

```python
def cut_license(self, afterimg, rect):
    rect[2] = rect[2] - rect[0]
    rect[3] = rect[3] - rect[1]
    rect_copy = tuple(rect.copy())
    mask = np.zeros(afterimg.shape[:2], np.uint8)
    bgdModel = np.zeros((1, 65), np.float64)
    fgdModel = np.zeros((1, 65), np.float64)
    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
```

该方法用于图像分割，从图像中提取出车牌区域。
调整 rect 中的宽度和高度，以计算车牌区域的右下角坐标。
创建一个与输入图像大小相同的全零掩膜。
初始化背景模型和前景模型
使用 cv2.grabCut 进行图像分割，其中 rect_copy 是车牌的初始矩形区域。
根据掩膜中的值，创建一个新的掩膜 mask2，其中 2 和 0 被替换为 0，而 1 和 3 被保留为 1。
使用 mask2 从原始图像 afterimg 中提取车牌区域。

定义 Segmentation 类[2]

初始化

```
class Segmentation():
    def __init__(self, cutimg):
        img_gray = cv2.cvtColor(cutimg, cv2.COLOR_BGR2GRAY)
        self.img_thre = img_gray
        cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV, self.img_thre)
        cv2.imwrite('thre_res.png', self.img_thre)
        self.white = []
        self.black = []
        self.height = self.img_thre.shape[0]
        self.width = self.img_thre.shape[1]
        self.white_max = 0
        self.black_max = 0
        for i in range(self.width):
            s = 0
            t = 0
            for j in range(self.height):
                if self.img_thre[j][i] == 255:
                    s += 1
                if self.img_thre[j][i] == 0:
                    t += 1
            self.white_max = max(self.white_max, s)
            self.black_max = max(self.black_max, t)
            self.white.append(s)
            self.black.append(t)
        self.arg = False
        if self.black_max > self.white_max:
            self.arg = True
```

初始化方法，用于对切割后的车牌图像进行二值化处理。
计算每一列的黑白像素总和，确定车牌图像的黑白反转状态。

判断图像是黑底白字还是白底黑字

```
def heibai(self):
    return self.img_thre
```

返回二值化后的车牌图像。

找到字符的结束位置

```
def find_end(self, start_):
    end_ = start_ + 1
    for m in range(start_ + 1, self.width - 1):
        if (self.black[m] if self.arg else self.white[m]) > (
        0.85 * self.black_max if self.arg else 0.85 * self.white_max):
            end_ = m
            break
    return end_
```

找到每个字符的结束位置，用于字符分割。

显示分割后的字符

```
def display(self):
    n = 1
    plt.figure()
    img_num = 0
    while n < self.width - 2:
        n += 1
        if (self.white[n] if self.arg else self.black[n]) > (
        0.15 * self.white_max if self.arg else 0.15 * self.black_max):
            start = n
            end = self.find_end(start)
            n = end
            if end - start > 5:
                cj = self.img_thre[1:self.height, start:end]
                img_num += 1
                cj = cv2.cvtColor(cj, cv2.COLOR_RGB2BGR)
                plt.figure(2)
                plt.subplot(2, 4, img_num)
                plt.title('{}'.format(img_num))
                plt.imshow(cj)
    plt.show()
    return self.img_thre
```

通过 while 循环遍历图片的宽度，跳过前两个像素（可能是为了避免边界效应）。使用条件语句检查当前像素的亮度是否超过某个阈值（这个阈值是基于 self.white_max 或 self.black_max 的15%来计算的，具体取决于 self.arg

的值）。
如果当前像素满足条件，就找到这个高亮区域的开始（start）和结束（end）位置。
处理并显示符合条件的区域：
如果找到的区域足够大（end - start > 5），则从 self.img_thre 中裁剪出这个区域。
将裁剪出的区域从 RGB 转换为 BGR 格式（因为 cv2 默认使用 BGR 格式），尽管如果 self.img_thre 已经是灰度图或二值图，这一步可能是不必要的。
使用 plt.figure(2) 尝试在一个新图形上显示这个区域，但实际上这会覆盖之前的图形，因为 plt.figure(2) 在循环中被重复调用而没有适当的清除或关闭之前的图形。
使用 plt.subplot(2, 4, img_num) 将图像放置在 2 行 4 列的子图网格中的适当位置。

主程序

```
if __name__ == '__main__':
    img = cv2.imread('8.png')
    img1 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    plt.figure(1)
    plt.suptitle('车牌识别', fontproperties=font)
    plt.subplot(2, 3, 1)
    plt.title('原始图像', fontproperties=font)
    plt.imshow(img1)

    license = Get_license()
    rect, afterimg = license.find_license(img)
    afterimg = cv2.cvtColor(afterimg, cv2.COLOR_RGB2BGR)
    plt.subplot(2, 3, 2)
    plt.title('预处理后图像', fontproperties=font)
    plt.imshow(afterimg)

    cv2.rectangle(afterimg, (rect[0], rect[1]), (rect[2], rect[3]), (0, 255, 0), 1)
    x1, y1, x2, y2 = int(rect[0]), int(rect[1]), int(rect[2]), int(rect[3])
    print('车牌框出:', x1, x2, y1, y2)
    plt.subplot(2, 3, 3)
    plt.title('车牌框出', fontproperties=font)
    plt.imshow(afterimg)

    cutimg = license.cut_license(afterimg, rect)
    plt.subplot(2, 3, 4)
    plt.title('车牌背景去除', fontproperties=font)
    plt.imshow(cutimg)
    print('车牌背景去除', x1, y1, x2, y2)
```

```
    # 创建根窗口
    root = tk.Tk()
    root.title('车牌识别程序')
    root.geometry('600x450')
    # 创建一个标签用于显示图片路径
    label = tk.Label(root, text='未选择图片')
    label.pack(pady=20)  # 使用pack布局管理器，并添加一些垂直填充

    # 创建一个按钮，点击时会调用select_image函数
    button = tk.Button(root, text='选择图片', command=select_image)
    button.pack()

    # 启动事件循环
    root.mainloop()
```

```
cutimg = cutimg[y1 + 3:y2 - 3, x1 - 1:x2 - 3]
print('cutimg:', cutimg)
height, width = cutimg.shape[:2]
cutimg1 = cv2.resize(cutimg, (2 * width, 2 * height), interpolation=cv2.INTER_CUBIC)
plt.subplot(2, 3, 5)
plt.title('分割车牌与背景', fontproperties=font)
plt.imshow(cutimg)

seg = Segmentation(cutimg)
plt.subplot(2, 3, 6)
img_hei = seg.heibai()
img_hei = cv2.cvtColor(img_hei, cv2.COLOR_RGB2BGR)
plt.title('车牌二值化处理', fontproperties=font)
plt.imshow(img_hei)
seg.display()
plt.show()

ocr = ddddocr.DddddOcr()
with open('thre_res.png', 'rb') as f:
    image = f.read()
res = ocr.classification(image)
print(res)
```

主程序部分使用 tkinter GUI 让用户[3]从文件中读取车牌图像，进行车牌预处理、定位、分割和字符识别，最终显示车牌识别结果。

详细总结

1. **导入所需库**：加载必要的库用于图像处理和绘图。
2. **加载字体**：加载自定义字体文件以便在图像上绘制文字时使用特定字体。
3. **定义 Get_license 类**：包含了一系列方法用于图像处理和车牌区域的提取。
4. stretch 用于图像灰度值拉伸。
    - dobinaryzation 用于图像二值化。
    - find_rectangle 用于寻找轮廓的矩形边界。
    - locate_license 用于通过颜色和轮廓面积定位车牌区域。
    - find_license 用于对车牌图像进行预处理并定位车牌。
    - cut_license 用于从图像中分割出车牌区域。
5. **定义 Segmentation 类**：包含字符分割相关方法。
    - 初始化方法对切割后的车牌图像进行二值化。
    - find_end 用于找到每个字符的结束位置。
    - display 用于显示分割后的字符。
6. **主程序**：调用各个方法实现车牌图像的预处理、车牌区域定位、背景去除、车牌字符分割和识别，并使用 tkinter 创建 GUI，最终显示结果并打印识别出的车牌号码。

通过这些步骤，代码完成了从图像中提取车牌并识别车牌字符的任务。

i

参考文献

i

[1].https://blog.csdn.net/weixin_43626082/article/details/125189531?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217200037971680022554787 8%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=172000379716800225547878&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-4-125189531-null-null.142^v100^pc_search_result_base9&utm_term=tkinter%20%E6%B8%85%E5%8D%8E%E6%BA%90&spm=1018.2226.3001.4187

[2].https://blog.csdn.net/MZYYZT/article/details/128212029?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217196709021680018586688 8%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=171967090216800185866888&

biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-~all~top_click~default-4-128212029-null-null.142^v100^pc_search_result_base9&utm_term=%E6%95%B0%E5%AD%97%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E8%BD%A6%E7%89%8C%E8%AF%86%E5%88%AB&spm=1018.2226.3001.4187
[3]https://blog.csdn.net/alan1ly/article/details/125423118?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217195441701680018854855 3%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=171954417016800188548553&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-125423118-null-null.142^v100^pc_search_result_base9&utm_term=%E6%95%B0%E5%AD%97%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E8%BD%A6%E7%89%8C%E8%AF%86%E5%88%AB&spm=1018.2226.3001.4187