Harbin Institute of Technology (Shenzhen)

Image Processing Project Report

Experiment Name: project2

Student Name: 李攀

Student ID: 20SK51183

Report Date: 2024/5/19

目录

1 实验内容.....1

2 实现方案.....1

2.1 预处理车牌的图像.....1

2.2 分割车牌位置图像.....4

2.3 车牌字符图像提取.....6

2.4 车牌字符识别模型训练.....7

2.5 根据模型识别出车牌字符.....9

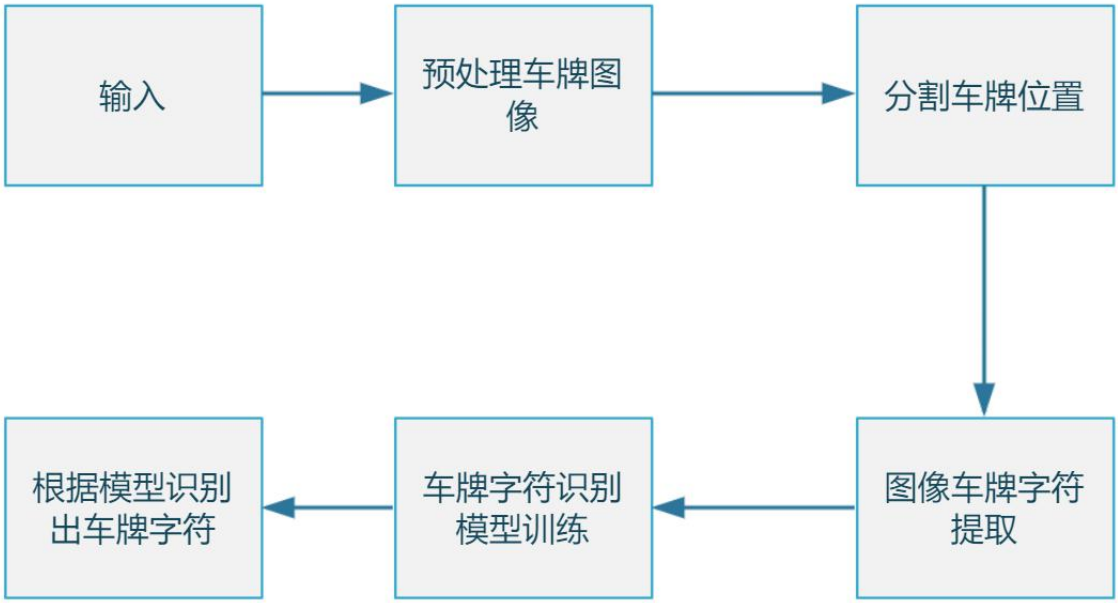
3 结果分析和总结.....9

4 小组分工.....10

1 实验内容

本实验旨在实现车牌识别系统的开发和验证。首先，输入包含车牌的图像，对车牌进行定位，提取出车牌区域的图像。然后，对提取的车牌图像进行文本分割，将车牌上的字符单独分离。最后，对每个分割后的字符图像进行识别，输出车牌上的字符信息。通过这一系列步骤，实现从图像输入到字符识别的完整流程，旨在提高车牌识别的准确率和效率，

2 实现方案



2.1 预处理车牌的图像

车牌的特征是蓝色背景和白色字符，通过颜色分割可以有效地提取车牌区域。首先将图像转换到 HSV 颜色空间，因为在 HSV 空间中，颜色信息（色调、饱和度）与亮度信息分离，这样更容易进行颜色分割。

根据蓝色特征提取出的代码如下

```python
def filter_blue(image):
    """颜色过滤，提取蓝色区域"""
    hsv = convert_to_hsv(image)
    # 蓝色的HSV范围
    lower_blue = np.array([100, 80, 80])
    upper_blue = np.array([140, 255, 255])
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    return mask
```

提取出的图像如下

图 1

观察到车牌时蓝底白字,但是车框也是白色,为了去除掉白色边框,需要使用白色来提取出 mask,提取的白色 mask 图像如下

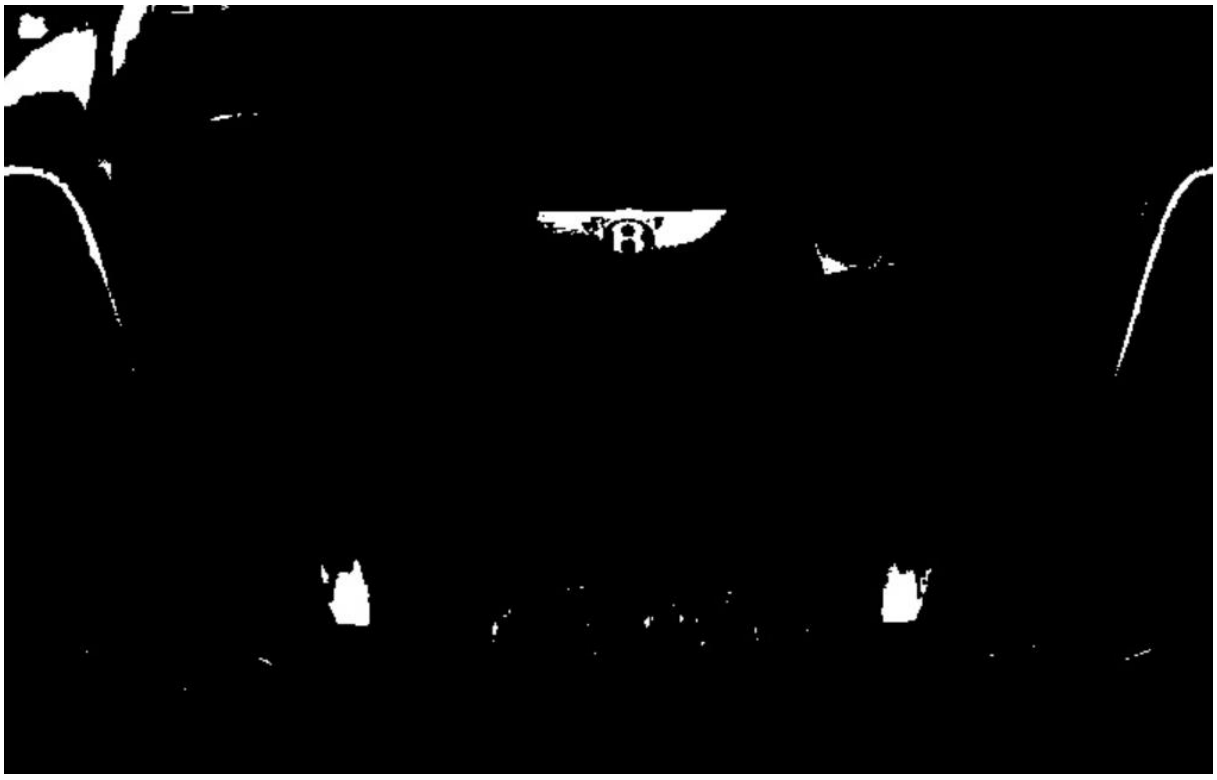


图 2

将蓝色的 mask 与白色 mask 重新生成一个组合的 mask，步骤如下

a. 按位取反操作：假设 mask_white 表示车牌中白色字符部分的掩码，那么 cv2.bitwise_not(mask_white)将得到一个与 mask_white 相反的掩码，即白色字符部分变为黑色，其他部分变为白色。

b. 按位与操作：将上述取反结果与 mask（通常是针对车牌蓝色背景的掩码）进行按位与操作。这样，只有同时属于车牌蓝色背景且不属于白色字符部分的区域才会保留为非零值。

代码实现如下：

```
# 结合蓝色和白色的掩码，去除白色边框但保留白色字符
mask_combined = cv2.bitwise_and(mask, cv2.bitwise_not(mask_white))
```

生成的图像如下图：

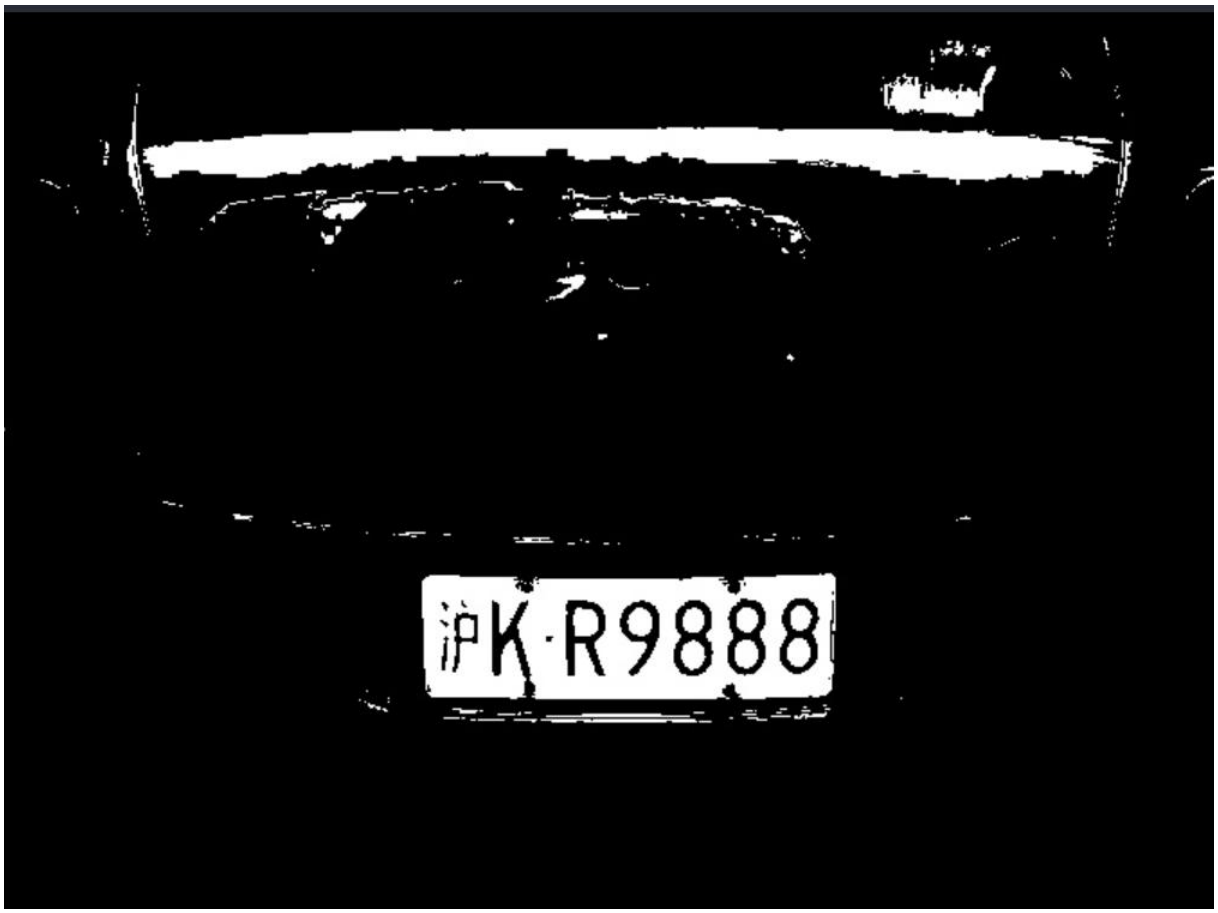


图 3

相比图 1，图 3 白色区域块明显变少。仍然存在部分椒噪声，因此使用形态学中的开闭运算去除图像中的噪声。

形态学中的开运算和闭运算是通过结构元素对图像进行操作，以去除噪声和保持目标形状。开运算由腐蚀和膨胀顺序组成，腐蚀使图像中的亮区域缩小，去除小的噪声点，而膨胀则恢复被腐蚀的区域，但不会恢复之前被去除的小噪声点。其效果是消除小于结构元素的噪声点，并保持整体形状特征。闭运算由膨胀和腐蚀顺序组成，膨胀使亮区域扩展，填补前景区域中的小孔和细缝，腐蚀则恢复膨胀后的前景形状，但不包括填补的小孔和细缝。闭运算的效果是填补小孔和细缝，并保持整体形状特征。开运算用于去除图像中的小噪声点，例如去除二值化图像中的盐噪声；闭运算用于填补图像中的小孔和细缝，例如填补二值化图像中的椒噪声。通过这些形态学操作，可以有效处理图像噪声，使后续图像分析和处理更加准确可靠。代码实现如下

```python
def morphological_operations(mask, k=5):
    """形态学操作，去除噪声"""
    kernel = np.ones((k, k), np.uint8)
    morphed = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    morphed = cv2.morphologyEx(morphed, cv2.MORPH_CLOSE, kernel)
    return morphed
```

去除噪声后的图像如下

图 4

2.2 分割车牌位置图像

首先找到所有轮廓，使用如下代码

```python
contours, _ = cv2.findContours(
    mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)的作用是在二值化的掩码图像中查找所有的轮廓，并返回一个包含这些轮廓的列表。首先，函数 cv2.findContours 接收三个参数：掩码图像 mask、轮廓检索模式 cv2.RETR_TREE 和轮廓近似方法 cv2.CHAIN_APPROX_SIMPLE。掩码图像 mask 通常是经过预处理的二值图像，其中前景对象为白色（255），背景为黑色（0）。轮廓检索模式 cv2.RETR_TREE 指示函数要检索轮廓的完整层次结构，这意味着即使是嵌套在其他轮廓内部的轮廓也会被检测到。轮廓近似方法 cv2.CHAIN_APPROX_SIMPLE 用于压缩水平、垂直和对角线方向的轮廓点，只保留轮廓的端点，以减少内存消耗和提高处理效率。绘制出的所有轮廓如下图。

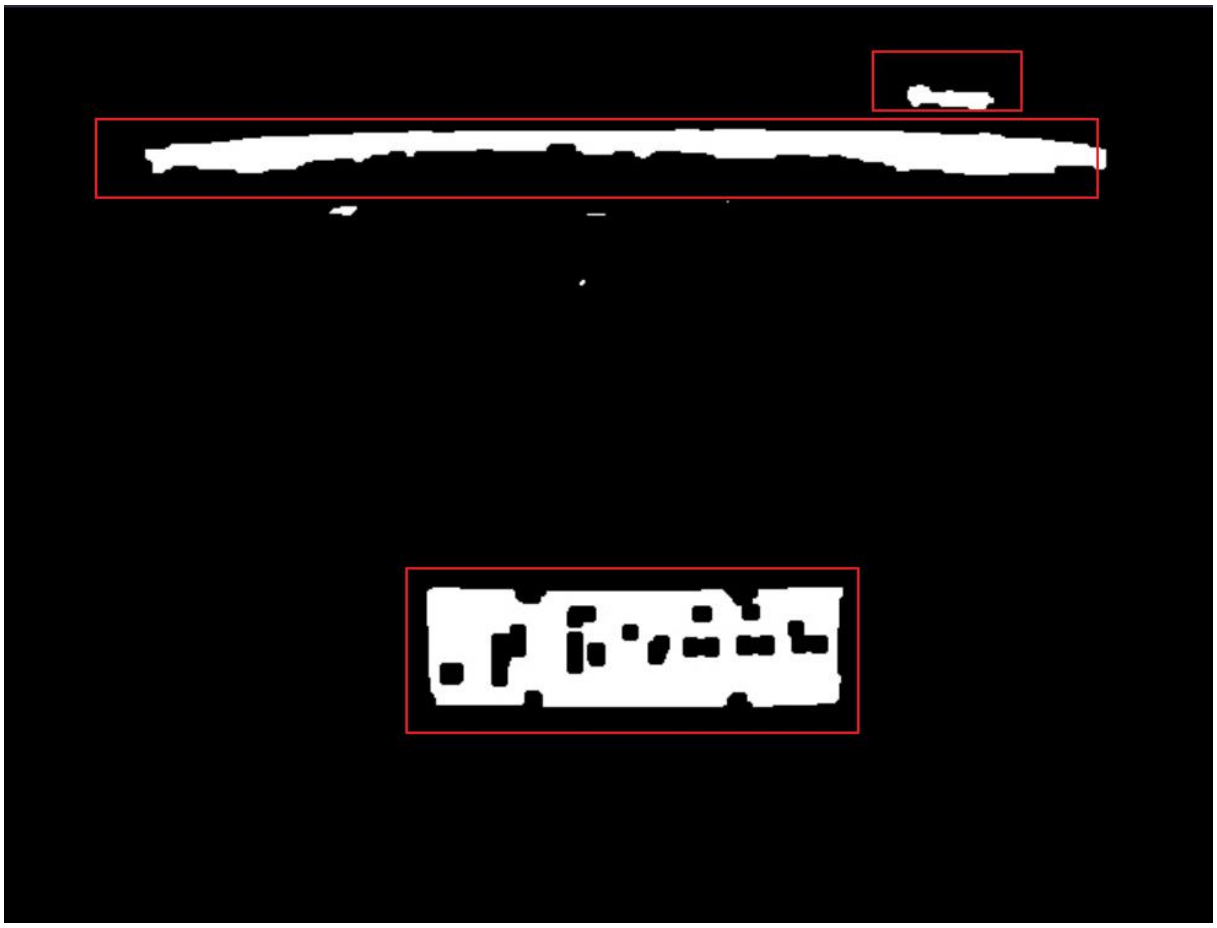


图 5

车牌的特征时宽度和高度的比值在 2~5，宽度大于 50，高度大于 20，因此可以根据这些特征过滤出符合条件的轮廓。

```
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    if 2 < w / h < 5 and w > 50 and h > 20:
        plate_image = image[y:y+h, x:x+w]
        return plate_image
```

最后得到如下车牌图像。

图 6

2.3 车牌字符图像提取

车牌图像经过二值化处理后，仍然存在许多白点，如图

图 7

可以使用开闭元算来去除这些白点，具体代码如下:

```
def preprocess_plate_image(plate_image):
    """车牌图像预处理"""
    gray_plate = cv2.cvtColor(plate_image, cv2.COLOR_BGR2GRAY)
    m1 = morphological_operations(gray_plate, k=3)
    _, binary_plate = cv2.threshold(
        m1, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return binary_plate
```

处理后的图像如图 8。

图 8

在二值图像中寻找连通组件并分割字符的过程中，使用 cv2.connectedComponentsWithStats 函数是一个常见的方法。该函数接收二值图像作为输入，其中前景像素值通常为 255，背景像素值为 0。函数会扫描图像中的所有像素，并基于像素连通性（通常是 8 连通性）识别出不同的连通组件。cv2.connectedComponentsWithStats 不仅返回每个连通组件的标签图像，还提供了每个组件的统计信息和质心位置。标签图像中，每个连通组件的像素值唯一且相同，表示它们属于同一个组件。统计信息包括每个连通组件的边界框、面积等，这对于字符分割尤为有用。通过遍历这些统计信息，可以根据面积和边界框筛选出符合字符尺寸和形状的组件，从而实现字符的分割。质心信息则可用于进一步的字符排列和对齐。总体来说，cv2.connectedComponentsWithStats 函数提供了一种高效的方法来识别和分割二值图像中的独立字符，广泛应用于光学字符识别（OCR）和图像处理领域。显然白色的点也会被当做一个连通组件，此处可以根据面积大小直接过滤。为了保证切分出的连通分量在 x 轴是有序的，此处按照 x 坐标排序。具体代码如下

```python
def segment_characters(binary_image):
    # 查找连通组件
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(
        binary_image, connectivity=8)

    # 提取字符
    characters = []
    for i in range(1, num_labels):  # 跳过背景
        x, y, w, h, area = stats[i]
        if area < 50:
            continue
        character = binary_image[y:y+h, x:x+w]
        characters.append((x, character))

    characters.sort(key=lambda i: i[0])
    return [x[1] for x in characters]
```

现在，分割得到如下的字符图像.

图 9

2.4 车牌字符识别模型训练

项目中提供了一个 template 字符图像数据集，本实验首先需要将数据集进行清洗，首先对灰度图像应用自适应阈值分割，通过 cv2.threshold 函数将其转换为二值图像 binary_image，其中 cv2.THRESH_BINARY + cv2.THRESH_OTSU 结合了固定阈值二值化和 Otsu's 方法。Otsu's 方法自动计算一个最优阈值，将图像分割为前景和背景部分，从而生成一个只有黑白像素的二值图像。接着，使用 cv2.resize 函数将生成的二值图像调整为尺寸为 40x20 的图像，interpolation=cv2.INTER_NEAREST 参数指定最近邻插值方法，这种方法通过选择最近的像素值进行插值，适合用于二值图像，因为它不会引入新的灰度值。这一过程确保了图像被标准化到指定尺寸，并保留了其二值特性，适用于后续的图像处理和分析步骤。

代码实现如下。

```python
def read_and_convert_to_binary(image_path, output_path):
    # 加载图像并直接转换为灰度图像
    gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # 应用阈值，将灰度图像转换为二值图像
    _, binary_image = cv2.threshold(
        gray_image, 0, 255,  cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    resize = cv2.resize(binary_image, (40, 20),
                        interpolation=cv2.INTER_NEAREST)
    cv2.imwrite(output_path, resize)

    return binary_image
```

将此数据集拆分成训练集和验证集，然后使用卷积升级网络模型训练出字符识别模型。

该模型采用 Keras 的 Sequential API 构建，通过一系列层来处理输入数据。首先，输入形状为 input_shape 的图像经过一个具有 32 个过滤器的卷积层，该层使用 3x3 的卷积核，ReLU 激活函数，并在边界处填充以保持输入尺寸。接着，应用一个 2x2 的最大池化层，同样在边界处填充以保持尺寸。然后，图像通过一个具有 64 个过滤器的卷积层，再次使用 3x3 的卷积核和 ReLU 激活函数，随后是另一个 2x2 的最大池化层。接下来，添加一个具有 128 个过滤器的卷积层，再次使用 3x3 的卷积核和 ReLU 激活函数，以及一个 2x2 的最大池化层。经过这些卷积和池化层后，特征图被展平成一维向量，并通过一个具有 512 个神经元和 ReLU 激活函数的全连接层。最后，输出层是一个全连接层，其神经元数量等于 num_classes，使用 softmax 激活函数进行多分类输出。模型在编译时，指定了 Adam 优化器、分类交叉熵损失函数，以及准确率指标。函数返回构建并编译好的模型。

```python
def build_model(input_shape, num_classes):
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(
            32, (3, 3), activation='relu', input_shape=input_shape, padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
```

最后，将编译好的模型保存到本地，便于后续加载和使用。

2.5 根据模型识别出车牌字符

使用训练好的模型对输入图像进行分类预测。首先接收三个参数：训练好的模型 model、预处理后的图像数组 img_array 以及类别索引的字典 class_indices。索引的字典在训练阶段创建。

函数通过 model.predict(img_array)方法对输入图像数组进行预测，返回一个包含各类别预测概率的数组。随后，使用 np.argmax(predictions[0])找到预测概率最高的类别索引，这个索引对应的是预测结果中概率最大的类别。接着，通过从 class_indices 字典中提取键名，将索引转换为类别名称，即通过 list(class_indices.keys())生成类别名称列表，并根据预测的类别索引 predicted_class_index 找到对应的类别名称 predicted_class。最终，函数返回预测的类别名称和包含各类别预测概率的数组。这个过程实现了从图像输入到类别预测的完整流程，利用模型的输出概率进行解释，并将结果转换为易于理解的类别名称。

实现的代码如下:

```python
def predict_image(model, img_array, class_indices):
    # 使用模型进行预测
    predictions = model.predict(img_array)
    # 找到概率最高的类别
    predicted_class_index = np.argmax(predictions[0])
    # 获取类别名称
    class_names = list(class_indices.keys())
    predicted_class = class_names[predicted_class_index]
    return predicted_class, predictions[0]
```

最终输出结果如图:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
['沪', 'K', 'R', '9', '8', '8', '8']
```

图 10

3 结果分析和总结

本实验首先通过车牌的颜色特征来快速定位到车牌图像处于的空间位置，然后使用轮廓查找加上车牌高宽比例过滤，分割出车牌图像，这部分图像处理效率高，速度较快。

然后将运用形态学操作去除了二值化车牌中的边框和部分白点，便于字符图像分割。通过查找车牌图像中的连通组件，快速识切割出字符图像。切割出的连通组件中，仍然包括部分小白点，这

个对应图像中车牌铆钉，可以通过面积大小直接过滤掉。这些小白点也可以提前去除，但通过均值滤波和形态学操作，效果均不理想。

基于项目提供的车牌字符图像数据集，本实验先对所有数据进行清洗，通过最近邻近插值的方式调整图像的大小到 40x20。然后使用卷积神经网络训练出字符图像识别模型，持久化模型到本地，后面使用此模型完成字符识别。

4 小组分工

此实验的所有部分，均为本人独立完成。