上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

视频多媒体检索实践课程大作业

基于 OpenCV 和 CNN 的车牌识别系统

学院　电子信息与电气工程学院

组员　杜卓轩　郭佳毅　吴瀚

组员　徐恺阳　严楠

指导老师　何大治

2025 年 1 月 15 日

目录

基于 OpenCV 和 LPRNet 的车牌识别系统

任务分工

- 杜卓轩 523030910083：设计 GUI 界面、生成 exe 文件
- 郭佳毅 523030910130：报告撰写、讲解视频录制
- 吴瀚 523030910020：神经网络搭建
- 徐恺阳 523030910085：车牌检测与裁剪、LOGO 设计、报告撰写
- 严楠 523030910114：车牌字符分割

1 前言

1.1 任务简述

本次视频多媒体检索实践课程大作业中，我们需要搭建一个车牌识别系统。基于课程所学内容方法，如 Canny 边缘检测、SIFT 算法等等，完成对车牌的检测、定位，字符的分割、识别以及最终的输出等。

具体功能需求如下：

1. 正确检测到图像中是否存在车牌；
2. 正确定位车牌在图像中的位置，输出裁切后的车牌图像；
3. 车牌字符分割后，与字符库匹配、检索并输出结果；
4. 正确识别车牌的颜色、字符数量、字符识别结果等；
5. 设计车牌识别系统的 GUI 界面；
6. 生成 exe 文件，供用户运行，并且支持从本地文件中上传或者由摄像头拍摄图像。

1.2 总体流程

- **Step1:** 利用传统的 OpenCV 方法检测车牌并裁剪出来；
- **Step2:** 将裁减好的车牌二值化，根据横向波峰分布分割字符；
- **Step3:** 搭建卷积神经网络，

- **Step4:** 根据用户需求以及内部函数的接口设计 GUI 界面；
- **Step5:** 设计 LOGO，将所有代码封装，统一打包成 exe 文件并发行。

2 车牌检测与裁剪

车牌检测与裁剪、车牌字符分割这两个核心步骤的实现代码封装在 PlatesLocator 类中，主要提供以下方法：

accurate_place(self, card_img_hsv, limit1, limit2, color): 根据车牌颜色进一步精确裁剪车牌边界。

locate_plates(self, car_pic, resize_rate=1): 从原始图片中裁剪出车牌区域的完整操作。

separate_characters(plate_img, color): 从裁剪好的车牌中分离字符。

2.1 基本思路

- 找出图片中所有的矩形轮廓，即为可能的车牌区域；
- 获取轮廓的最小外接矩形，并根据车牌的理论长宽比进行筛选；
- 对倾斜的矩形区域进行仿射变换修正；
- 根据矩形中像素点的主体颜色判断车牌颜色，并筛去非蓝色、绿色的区域；
- 根据车牌颜色裁剪去非蓝色、绿色区域，进一步精确裁剪车牌边界。

2.2 具体实现

Step1: 图像预处理

首先，我们通过 cv2.resize() 和 cv2.GaussianBlur() 这两个函数统一所有图像大小，并进行一遍高斯模糊，目的是降低图像噪声，减少干扰。同时，通过 cv2.cvtColor() 将图像转化灰度图像，为后续边缘检测和二值化做准备。原始图像和经高斯模糊后的灰度图像如图 1和图 2所示。

然后，我们通过 cv2.morphologyEx() 函数对灰度图进行一遍开运算，以去除图片中不相关的小物体。cv2.addWeighted() 将经过开运算后的灰度图与原始灰度图进行加权叠加，将两幅图像合成为一幅图像，增强车牌区域的对比度。效果如图 3所示。

图 1: 原始图像

图 2: 经高斯模糊后的灰度图像

图 3: 开运算并加权叠加

图 4: Canny 边缘检测

Step2: 边缘检测

首先，我们通过 cv2.threshold() 二值化，对二值化化的图像通过 cv2.Canny() 进行 Canny 边缘检测，效果如图 4所示。然后，通过 cv2.morphologyEx() 进行闭运算和开运算让图像边缘成为一个整体，连接边缘中的小缺口，同时消除边缘区域中的小噪声。效果如图 5所示，可以看出车牌所在处已连接成一整块区域。

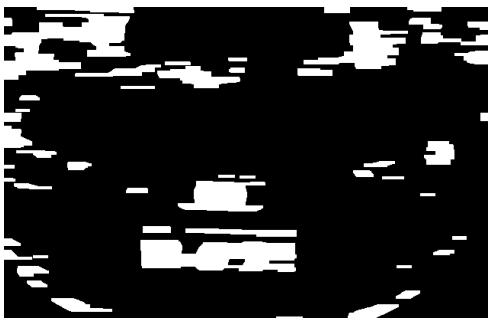

图 5: 闭开运算

图 6: 所有矩形区域

Step3: 轮廓检测并筛选矩形

我们通过 cv2.findContours() 查找图像边缘整体形成的轮廓，并且过滤掉面积过小或者是过大的轮廓。经筛选后，画面中仅剩如图 6所示的零星的几个矩形轮廓。

Step4: 矫正车牌

由于某些矩形区域可能是倾斜的，所以我们通过如下代码（以正角度倾斜为例）对该区域进行仿射变换，从而将其调整至水平。

```
new_right_point = [right_point[0], heigth_point[1]]
```

```
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(old_img, M, (pic_width, pic_height))
card_img = dst[int(left_point[1]):int(heigth_point[1]),
int(left_point[0]):int(new_right_point[0])]
ard_imgs.append(card_img)
```

Step5: 确定车牌颜色

首先，我们通过 cv2.cvtColor() 将车牌图像转换至 HSV 色彩空间，并统计其中在蓝色或者绿色范围内的像素点的个数，根据最多的像素颜色决定该区域是否为车牌，以及是什么颜色的车牌。

- 绿色 HSV 范围：$35 < H <= 99$ and $S > 43$ and $V > 46$
- 蓝色 HSV 范围：$99 < H <= 124$ and $S > 43$ and $V > 46$

Step6: 进一步裁剪边界

确定好车牌的颜色后，我们据此进一步精细地裁剪车牌边界。统计车牌图像每一行（列）的像素点在车牌颜色范围的个数，如果没有达到阈值，则将该行（列）裁剪出去。该部分由函数 accurate_place(self, card_img_hsv, limit1, limit2, color) 实现。

裁剪后的车牌如图 7所示。

图 7: 裁剪后的车牌

3 车牌字符分割

该部分目的是将切割好的车牌图像转换成黑白图像，并将每一个字符切割出来，用于后续神经网络的识别，由函数 find_waves(threshold, histogram) 和 separate_characters(plate_img, color) 实现。

3.1 基本思路

- 以灰度图的方式读取车牌图像，并二值化为黑白图像；
- 根据黑白图像每一列白色像素个数判断该列是否属于字符；
- 根据字符位置分割车牌，得到字符。

3.2 具体实现

Step0:

Step1: 图像预处理

首先，我们通过 cv2.cvtColor() 读入车牌的灰度图像。然后，对于蓝色车牌，直接通过 cv2.threshold() 二值化得到黑底白字的黑白图像；对于绿色车牌，先通过 cv2.bitwise_not() 取反，再二值化得到黑白图像。效果如图 8所示。

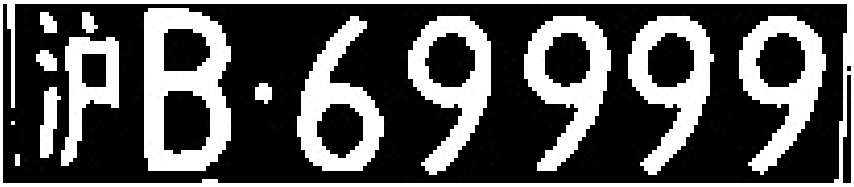

图 8: 二值化后的车牌

Step2: 寻找字符的位置

由于经预处理，字符呈白色，所以我们在每个 x 位置遍历该列的所有像素点，计算白色像素的比例，一旦这个比例超过设定的阈值，即认为这一列上有白色字符的信息。从左到右遍历，如果 x_1 列开始出现较多的白色像素，一直持续到 x_2 列，则认为水平坐标位于 (x_1, x_2) 的这部分图像是一个字符（或汉字的一部分）。

首先，我们将车牌上下 2% 的位置裁去，即去除上下白边。然后，考察图像的水平投影直方图，倘若每一列的白色像素高于设定的阈值就是存在字符的部分。接着，我们定义 find_waves(threshold, histogram) 函数，根据设定的阈值和图像的水平投影直方图，找出所有的波峰，用于分隔字符。

Step3: 裁去多余的竖直白线

分割字符过程中，我们发现裁剪的图片很多时候最左边会有一条白边，如图 8所示。因此，这里我们判断第一个波峰的宽度是否小于最后一个波峰宽度的 1%，如果比这个小，则认为是白线，将其去除。代码如下：

```
if wave_peaks[0][0] / wave_peaks[-1][1] < 0.01:
    wave_peaks.remove(wave_peaks[0])
```

Step4: 根据字符位置分割车牌

首先，我们判断 find_waves() 函数得到的波峰个数，如果过少，即认为车牌裁剪不正确。然后，由于一个汉字的左右部分可能会被识别成两个峰，会导致一个汉字被分成两部分，所以需要将汉字的左右部分合在一起分割。

由于我们已经去掉了车牌的左侧边缘白线，所以第一个峰就是汉字的一部分，对后面的峰进行遍历，如果后面峰的结束位置减去第一个峰的宽度大于最后一个峰（即字母或者数字）的宽度的 70%，则认为这几个峰合起来是一个完整的汉字，然后将这个合成的新的峰取代遍历过的几个峰。

Step5: 去除车牌第 2、3 字符之间的白点

这里，我们对所有的峰进行遍历，找到峰最大的宽度，将所有峰中小于这个最大宽度的 30% 的峰都删除，这样剩下的峰的宽度就和最宽的相近，即认为都是字母、数字和汉字字符，除去了车牌第 2、3 字符之间的白点。

Step6: 分割字符

根据几个峰的位置信息切割图片，代码如下：

```
part_cards = [gray_img[:, wave[0]:wave[1]] for wave in wave_peaks]
```

最终效果如图 9所示。

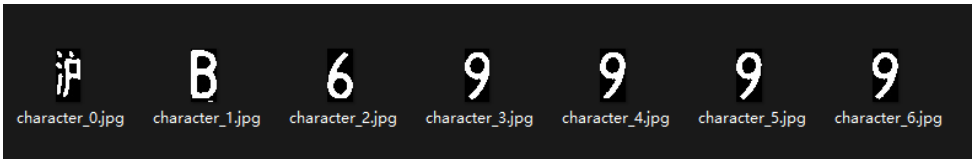

图 9: 车牌字符分割结果

4 字符匹配

该部分在车牌裁剪、字符分割模块的基础上，采用**端到端 (End-to-End)** 模型，将车牌裁剪、字符分割、字符匹配这三个步骤合并为一个整体，即模型直接接受输入图像并直接输出识别结果，无需进行显式的字符分割过程。

换言之，模型在训练过程中已经学会了如何从整张车牌图像中提取并识别字符，大大简化处理流程，提高效率。

4.1 基本思路

- 使用轻量化的深度学习模型 **LPRNet** 提取字符特征；
- 利用训练好的模型对字符图像进行分类，得到字符的预测结果；
- 使用 **Greedy Decode** 对模型输出的概率分布进行后处理，生成最终的车牌号码。

4.2 具体实现

Step0: 模型的训练与测试

对于本次模型的训练，我们选取的数据集来源于 CBLPRD330k[1] ，我们选取了 50000 张图片进行训练，10000 张图片进行测试。为了提升模型的泛化能力，训练数据经过多种数据增强技术处理。主要包括：

- **高斯模糊：** 以一定概率（15%）对图像应用模糊处理，模拟不同的拍摄环境和噪声干扰；

[1]下载链接：https://github.com/SunlifeV/CBLPRD-330k

- **随机旋转：**以一定概率（10%）对图像进行随机旋转（最大角度限制在 15°），增强模型对不同角度车牌的识别能力；
- **归一化：**将图像像素值标准化到特定范围，确保数据输入的一致性。

随后通过自定义的构建函数，搭建 **LPRNet 模型**结构，并根据需求选择加载预训练权重或随机初始化网络参数。权重初始化采用 **Kaiming 正态分布**，适应 ReLU 激活函数，确保模型在训练初期具有良好的收敛性。

训练过程中我们采用 **RMSprop 优化器**，结合动量和权重衰减等参数，优化模型参数。损失函数选择 **CTC 损失（Connectionist Temporal Classification）**，适用于处理序列预测任务，如车牌字符的识别与对齐。

随后，我们记录下我们的训练曲线，如图 10所示。

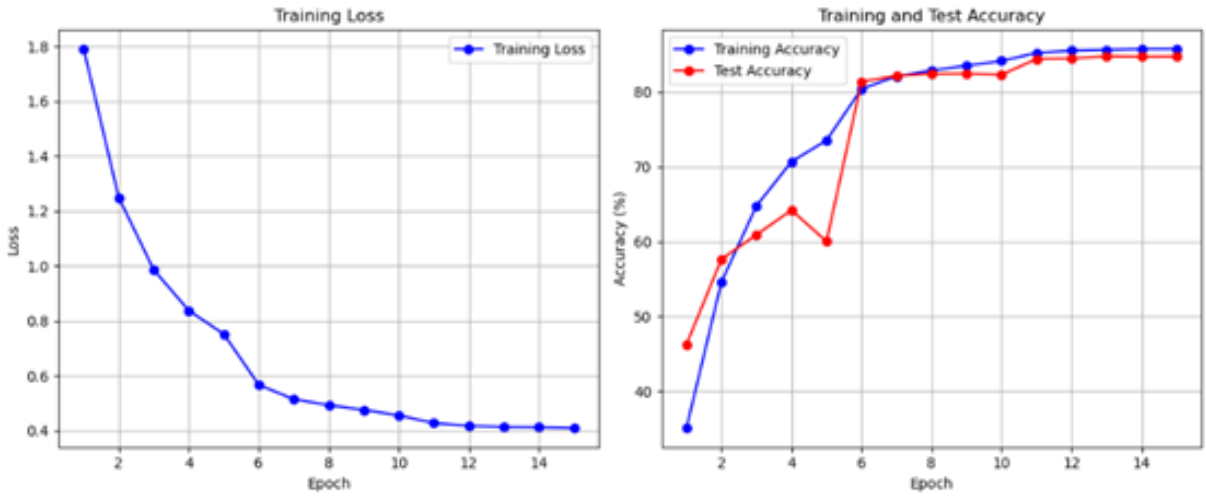

图 10: 模型的训练曲线

可见，在 15 个 epcho 内，我们的训练损失持续下降，训练正确率验证正确率均显著上升，并且没有明显的过拟合现象。

随后我们采用了 2000 张图片（来源均为 CBLPRD330k）进行预测，预测结果为：

- 对于任意的单张图片，预测完全正确的概率约为 54%；
- 对于任意单个字符，预测完全正确的概率约为 86%。

虽然目前的结果不能说尽如人意，但是对于 LPRNet 这样的轻量化模型来说已经非常不错了。

Step1: 定义轻量化特征提取模块

为了降低模型的计算复杂度和参数量，实现更高效的车牌字符识别，我们定义了轻量化特征提取的模块 small_basic_block，通过多层卷积操作提取图像的局部和全局特征。其中，1×1 卷积可以减少计算量，提高效率；3×1 与 1×3 卷积分别提取水平和垂直方向的特征。部分代码如下：

```
nn.Conv2d(ch_in, ch_out // 4, kernel_size=1),
nn.ReLU(),
nn.Conv2d(ch_out // 4, ch_out // 4, kernel_size=(3, 1), padding=(1, 0)),
nn.ReLU(),
nn.Conv2d(ch_out // 4, ch_out // 4, kernel_size=(1, 3), padding=(0, 1)),
nn.ReLU(),
nn.Conv2d(ch_out // 4, ch_out, kernel_size=1),
```

Step2: 搭建 LPRNet 模型的主干网络

该步骤负责从输入图像中提取特征，包含多个卷积层、池化层和激活层。我们通过多层卷积和池化操作提取特征，使用 small_basic_block 提高计算效率。网络中的主要层次如下：

卷积层 + ReLU 激活函数

- 第一层是一个 3x3 卷积层，输出 64 个特征图；
- 然后使用 BatchNorm 和 ReLU 激活函数进一步处理。

最大池化层：用于减少空间维度，增强特征

- 使用 MaxPool3d，池化核为 (1, 3, 3)，步长为 (1, 1, 1)；
- small_basic_block：这是一个自定义的小型卷积块，由一系列卷积层组成，目的是进一步提取车牌图像的特征。每个 small_basic_block 先进行 1x1 卷积来调整通道数，再进行不同尺寸的卷积（如 (3, 1) 和 (1, 3)）来处理空间特征，最后再使用 1x1 卷积来恢复通道数；
- small_basic_block 层重复：在模型中，我们可以看到有多个这样的卷积块（如 ch_in=64, ch_out=128，ch_in=64, ch_out=256 等），这些块帮助网络更好地捕捉车牌图像中的各种空间模式。

池化层

- 在某些层后，模型使用 MaxPool3d 或 AvgPool2d 来减少特征图的空间维度，以降低计算量，并增强对车牌字符的全局信息理解。

Dropout 层

- 为了防止过拟合，网络使用了 Dropout 层，这有助于在训练过程中随机丢弃部分神经元，从而提高网络的泛化能力。

Step3: 全局上下文建模

该部分是 LPRNet 的特色之一，目的是通过对不同层特征的操作（如平均池化、平方和均值计算等）来建模车牌的全局上下文信息。这种信息有助于增强车牌字符的识别效果。主要步骤如下：

1. **平方和均值化:** 该步骤对每个特征图的平方进行平均，然后标准化，这样可以增强网络对全局特征的关注；

2. **特征拼接:** 将不同层的全局特征进行拼接，形成一个新的特征向量。

Step4: 图像预处理

preprocess_image(img_path, img_size) 将输入图像调整为模型期望的尺寸与格式，具体步骤如下：

- 调整大小以使图像满足模型输入要求；
- 归一化，减去均值后缩放到 [-1, 1] 范围；
- 转换通道顺序，从 HWC 格式变为 CHW 格式，适配 PyTorch。

部分代码如下：

```
def preprocess_image(img_path, img_size):
    image = cv2.imdecode(np.fromfile(img_path, dtype=np.uint8), cv2.IMREAD_COLOR)
    if image is None:
        raise ValueError(f"Unable to load image: {img_path}")

    image = cv2.resize(image, tuple(img_size))
    image = image.astype('float32')
    image -= 127.5
    image *= 0.0078125
    image = np.transpose(image, (2, 0, 1))
    return torch.from_numpy(image).unsqueeze(0)
```

Step5: 模型推理与解码

首先，我们通过模型得到字符类别的概率分布，从概率分布中提取类别。然后，消除重复字符与无效的空白字符，生成最终字符序列，并将类别索引映射为实际字符。代码如下：

```
no_repeat_blank_label = []
pre_c = preb_label[0]
if pre_c != len(CHARS) - 1:
    no_repeat_blank_label.append(pre_c)
for c in preb_label:  # Drop repeated and blank labels
    if (pre_c == c) or (c == len(CHARS) - 1):
        pre_c = c
        continue
```

```
        no_repeat_blank_label.append(c)
        pre_c = c

result = ''.join([CHARS[i] for i in no_repeat_blank_label])
```

模型对前文所述的车牌的预测结果如图 11所示。

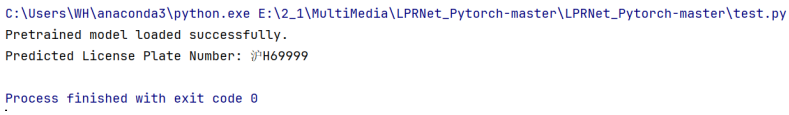

图 11: 车牌字符的预测结果

5 GUI 界面

5.1 基本思路

- 使用 **PyQt5** 创建图形界面，用于图片上传、处理和结果显示；
- 设计图形界面的功能模块，包括图片上传按钮、结果展示区以及菜单栏；
- 调用图片处理算法（如车牌识别），根据图片特性提取并显示相关结果；
- 优化界面交互，通过图像缩放、提示信息和样式设置，提升用户体验。

5.2 具体实现

Step1: 初始化主界面

首先，我们继承 `QMainWindow` 定义主窗口类，并通过 `setCentralWidget()` 方法设置中央窗口组件。主界面布局包括以下部分：

- 左侧：图片上传和摄像头按钮。
- 中间：用于显示上传图片的 `QLabel`。
- 右侧：结果展示区域，包含分组框显示处理后的图像、识别结果和处理描述。

主窗口初始化代码如下：

```
self.central_widget = QWidget()
self.setCentralWidget(self.central_widget)
self.main_layout = QHBoxLayout(self.central_widget)
self.create_menu()
self.create_layouts()
```

Step2: 菜单栏创建

菜单栏包括“文件”和“帮助”两个主菜单，提供退出程序和关于信息的功能。菜单栏的实现代码如下：

```
menu_bar = self.menuBar()
file_menu = menu_bar.addMenu("文件")
exit_action = QAction("退出", self)
exit_action.triggered.connect(self.close)
file_menu.addAction(exit_action)

help_menu = menu_bar.addMenu("帮助")
about_action = QAction("关于", self)
about_action.triggered.connect(self.show_about)
help_menu.addAction(about_action)
```

初始化主界面并创建菜单栏后的 GUI 界面如图 12所示。

图 12: 初始 GUI 界面

Step3: 图片上传功能

我们通过 QFileDialog 实现图片选择，并将图片显示在中央区域。上传图片后调用图片处理函数，处理结果动态更新在右侧展示区域。图片上传的部分代码如下：

```
file_path, _ = QFileDialog.getOpenFileName(
    self, "选择图片", "", "图片文件 (*.png *.jpg *.jpeg)")
if file_path:
    pixmap = QPixmap(file_path)
    self.uploaded_image_label.setPixmap(pixmap)
    self.process_image(file_path)
```

Step4: 识别结果展示

结果展示包括以下功能模块：

- 处理后的图像：通过 QVBoxLayout 动态添加图像组件。
- 识别结果：在分组框中显示车牌号码。
- 处理描述：通过 QLabel 动态添加文字描述。

以下为动态更新结果展示的关键代码：

```
label = QLabel(recognized_chars)
label.setWordWrap(True)
self.recognized_chars_layout.addWidget(label)

image_label = QLabel()
image_label.setPixmap(QPixmap(result_image_path))
self.processed_images_layout.addWidget(image_label)
```

Step5: 保存处理结果

通过 QFileDialog 实现保存功能，允许用户选择保存的路径和格式，部分代码如下：

```
save_path, _ = QFileDialog.getSaveFileName(
    self, "保存图片", "", "图片文件 (*.png *.jpg)")
if save_path:
    pixmap.save(save_path, "PNG")
```

上传待检测图片并显示识别结果后的 GUI 界面如图 13所示。

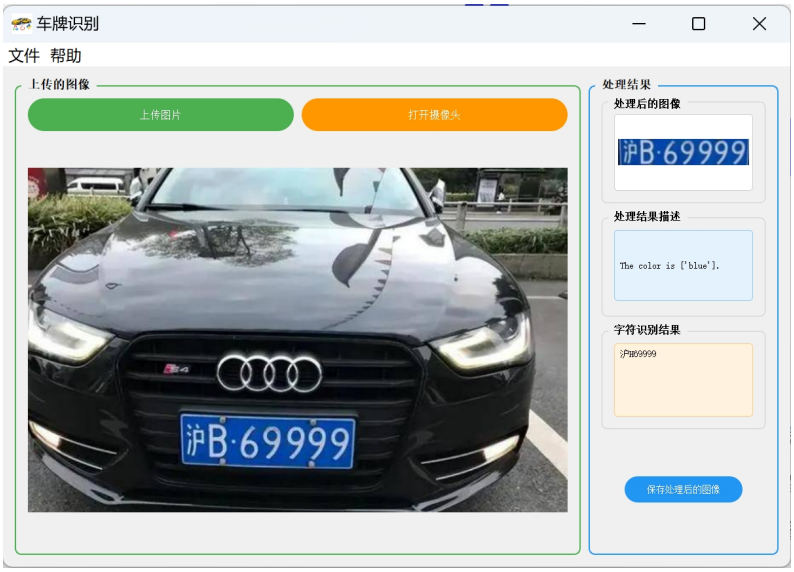

图 13: 上传图片后的 GUI 界面

Step6: 摄像头集成

通过调用图片处理函数支持摄像头的实时图片获取与处理，代码如下：

```
result = recognize_license_plate(input_source="camera")
self.handle_processing_result(result)
```

打开摄像头后的 GUI 界面如图 14所示，用户根据提示按下'q' 键后即可进行拍照，并得出相应的识别结果。

图 14: 打开摄像头后的 GUI 界面

6 打包发行

由于本车牌识别系统基于 Python 实现，并且用到 OpenCV 和 PyTorch 等相关库，故设计如图 15所示的 LOGO，作为 exe 文件的图标。

图 15: LOGO

7 结语

本次实践课程大作业从任务分析到系统设计，再到代码实现与调试，完成了**基于 OpenCV 和 LPRNet 的车牌识别系统**。在此过程中，我们团队成员共同协作，解决了诸多技术难题，并在系统的性能优化和功能实现上取得了一定的突破。这些成果的取得离不开何老师的悉心指导，以及助教们在实践中的细心帮助。在此，向何老师和助教们表达最诚挚的感谢！

本项目的开发让我们深入理解了车牌识别技术的核心流程，包括图像预处理、车牌检测与裁剪、字符分割与识别等。同时，通过设计 GUI 界面与打包发行，我们还进一步掌握了制作软件的流程。未来，我们希望能够进一步优化系统性能，例如提升识别速度、增强复杂场景的鲁棒性，并探索更多智能视觉应用的可能性。我们诚挚欢迎老师和同学们对我们的成果提出宝贵意见！

最后，再次感谢何老师和助教的付出，以及团队成员的共同努力！同时，本团队使用 GitHub 进行多人协作，仓库详见https://github.com/PasserbyZzz/PlateRecognition。欢迎提出 Issues 以及 Pull requests。**Wish for your Star!!**

参考文章

[1] 小布啊啊啊. Yolov5、cnn、svm 实现车牌检测, 2022. https://blog.csdn.net/swust512017/article/details/125637044.

[2] 自大人. python+opencv 只抠出车牌部分, 2020. https://blog.csdn.net/u014431739/article/details/107525601.

[3] wzh191920. 车牌号识别 python + opencv, 2018. https://blog.csdn.net/wzh191920/article/details/79589506.

[4] 荣仔！最靓的仔！. 基于 opencv 和 python 的车牌提取和字符分割, 2020. https://blog.csdn.net/IT_charge/article/details/107427133.

[5] detectRecog. Ccpd (chinese city parking dataset, eccv), 2019. https://github.com/detectRecog/CCPD.

[6] wzh191920. License-plate-recognition, 2018. https://github.com/wzh191920/License-Plate-Recognition.

[7] Dara to win. License-plate-recognition, 2018. https://github.com/Dara-to-win/Plate-Recognition.

[8] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. In *2018 European Conference on Computer Vision (ECCV)*, pages 580–596, Sep 2018.

[9] Xiangyu Zhang, Zhanpeng Li, Chen Change Loy, and Xiaoou Tang. Understanding and improving convolutional neural networks via concatenated rectified linear units. *arXiv preprint arXiv:1806.10447*, 2018.

[10] S L. China-balanced-license-plate-recognition-dataset-330k. https://github.com/SunlifeV/CBLPRD-330k, 2023.

[11] sirius ai. Lprnet_pytorchn, 2019. https://github.com/sirius-ai/LPRNet_Pytorch.