



# 车牌识别系统大作业报告

陈玺徽 523030910049

侯湘贻 523030910034

李青雅 523030910004

孙抚养 523030910102

王乐凡 523030910036

# 目录

<b>1 实验内容</b>	<b>2</b>
<b>2 实验分工</b>	<b>2</b>
<b>3 实验环境</b>	<b>2</b>
<b>4 实验原理</b>	<b>2</b>
4.1 YOLOv8 . . . . .	2
4.2 基于卷积神经网络 (CNN) 的特征提取 . . . . .	3
4.3 多任务学习 . . . . .	3
4.4 GUI 界面的设计及架构 . . . . .	4
<b>5 实验流程</b>	<b>4</b>
5.1 输入图像的切割 . . . . .	4
5.2 图像输入模型 . . . . .	5
5.2.1 车牌识别的深度学习训练过程 (train_continuable.py) . . . . .	6
5.2.2 车牌检测与字符识别 (detect_aug.py) . . . . .	6
5.2.3 识别结果展示 . . . . .	6
5.3 GUI 界面制作 . . . . .	7
5.3.1 基本功能及实现 . . . . .	7
5.3.2 美化需求 . . . . .	7
5.4 衔接 . . . . .	8
5.5 exe 生成 . . . . .	8
<b>6 实验结果展示</b>	<b>9</b>
<b>7 实验心得体会</b>	<b>9</b>
<b>8 实验代码附录</b>	<b>10</b>
8.1 cut . . . . .	10
8.2 train . . . . .	11
8.3 detect . . . . .	17
8.4 changeSuffix . . . . .	19
8.5 GUI . . . . .	20

# 1 实验内容

以小组为单位制作一个车牌识别系统，能够完成对车牌的检测、定位、字符识别及输出等。  
具体要求：

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. 将车牌字符分割后，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 GUI 界面；
6. 生成 exe 文件，供其他用户运行、从其本地文件中选取图像

# 2 实验分工

分工	成员
字符识别模型的架构与训练	陈玺徽
车牌标定与模型训练	李青雅，孙抚瑶
代码整合	李青雅，王乐凡
GUI 框架搭建	孙抚瑶，侯湘贻
报告撰写	侯湘贻，王乐凡

整体分工比例均衡。

# 3 实验环境

编程语言：python

编译器：Visual Studio Code

使用到的库：cv2,Pytorch,PyQt5,Numpy,PyInstaller

运用到的现有模型：YOLOv8, CNN

# 4 实验原理

## 4.1 YOLOv8

YOLO(You Only Look Once) 是一种 one-stage 目标检测算法，可以用于图像分类、物体检测和实例分割等任务。可以进行更快的训练和推理，达到更高的精度，具有更好的泛化能力和统一的架构设计。

YOLOv8 提供了一个全新的 SOTA 模型，包括 P5 640 和 P6 1280 分辨率的目标检测网络和基于 YOLACT 的实例分割模型。和 YOLOv5 一样，基于缩放系数也提供了 N/S/M/L/X 尺度的不同大小模型，用于满足不同场景需求。

Loss 计算方面采用了 TaskAlignedAssigner 正样本分配策略，并引入了 Distribution Focal Loss。训练的数据增强部分引入了 YOLOX 中的最后 10 epoch 关闭 Mosiac 增强的操作，可以有效地提升精度。

网络架构：

Backbone: CSPDarknet

Neck: FPN+PAN 结构

Head: 密集预测层

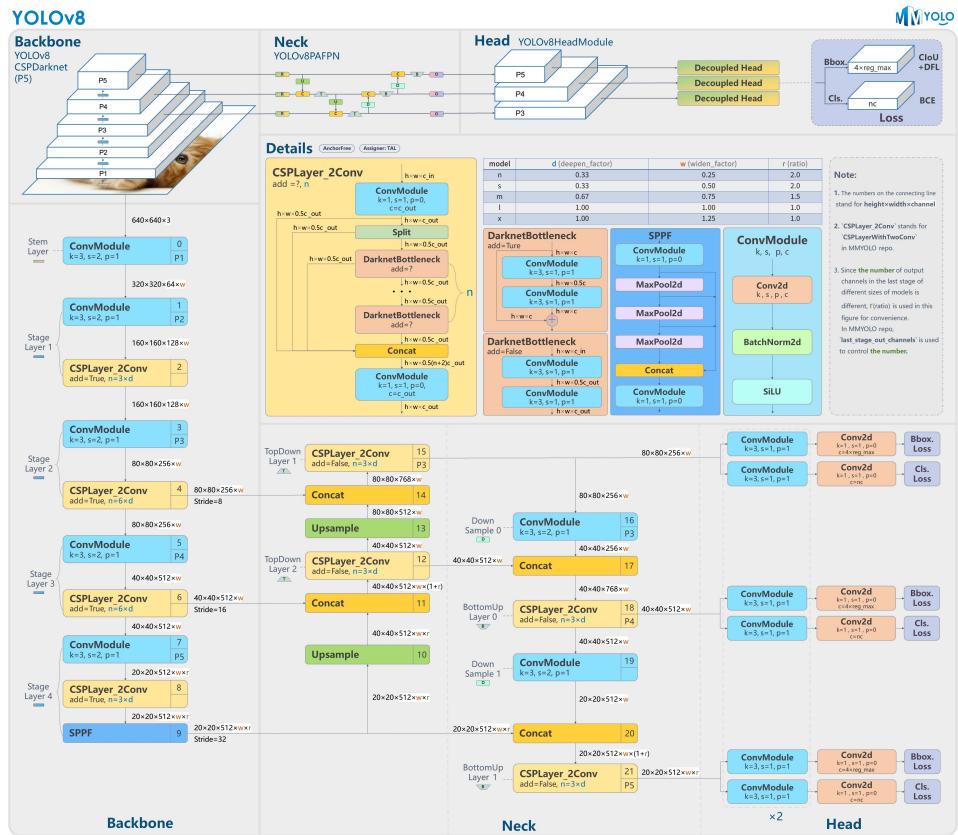


图 1: 网络结构

## 4.2 基于卷积神经网络 (CNN) 的特征提取

卷积神经网络 (CNN) 作为模型的核心架构，从输入图像中提取特征，通过多层卷积操作提取局部特征，逐步捕获图像的高阶表示。同时激活函数 (ReLU) 增强非线性表达能力，使用池化层 (MaxPooling) 减少特征图的维度，从而降低计算复杂度并保留主要特征。在此基础上还特别加入 Dropout 层以防止过拟合。

## 4.3 多任务学习

模型同时解决了两个子任务，包括车牌类型分类（预测车牌的类别：普通蓝牌或新能源车牌）和字符识别（根据车牌类型输出 7 位或 8 位字符的识别结果）。并通过在模型中设计类型分类器和字符分类器，将任务分解并联合训练，提高整体性能。

## 4.4 GUI 界面的设计及架构

通过 PyQt5 的布局管理、控件库、样式表，以及信号与槽机制，实现了一个清晰直观、功能完善的 GUI 界面，适用于智能车牌识别系统的前端界面设计。

## 5 实验流程

### 5.1 输入图像的切割

首先，需要对输入的图像进行预处理和切割，以提取出感兴趣的区域（如车牌区域）。这一过程包括图像的读取、灰度转换、二值化处理以及区域裁剪等步骤，确保提取出的图像区域符合后续处理的要求。



图 2: 车牌检测效果图

上图红框部分即检测到车牌位置。

虽然目前的识别效果对某些图片已经可以做到对车牌的较精准定位（如上述两图），但是在某些场景下仍然很不理想。技术层面上的主要原因有两个，一个是车牌检测算法并没有检测到车牌，传统的 opencv 可能不能很好地检测到车牌的边缘从而进行裁剪，故我们尝试使用 Faster R-CNN 和 YOLO 等经典的目标检测算法，然后做矫正或进一步的区域筛选；另一个原因是在识别算法上，本次我们仅是基于少量的训练数据训练了 SVM，可以尝试增加训练集并把模型替换成一些更复杂的机器学习模型或使用 CNN 训练一个多分类的深度学习模型，亦或是直接考虑一些基于 Attention 的 CNN-RNN 架构的 OCR 识别模型。这里我们使用的是卷积神经网络。

最后我们选择了 YOLOv8。首先选取了 600 张图片，其中  $\frac{2}{3}$  图片中含单张车牌， $\frac{1}{3}$  图片中含多张车牌。我们使用 LabelImg 对这 600 张图片一一进行标注，保存标注好的图片和 txt 文件，运行 YOLOv8 进行训练，得到一个.pt 文件，作为我们训练好的模型参与后面的步骤。

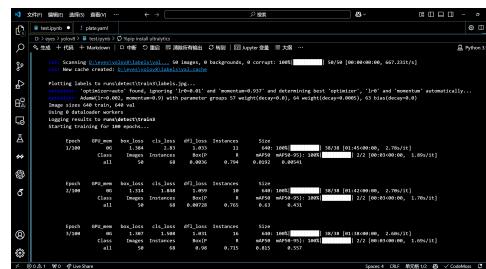


图 3: 训练中

下图为我们训练好的模型的相关参数。可以看到，随着训练轮数的增加，模型在高置信度的

F1-score 较低，损失函数的值也越来越小，说明模型的预测值和实际值越来越接近，准确率也在逐步提升。

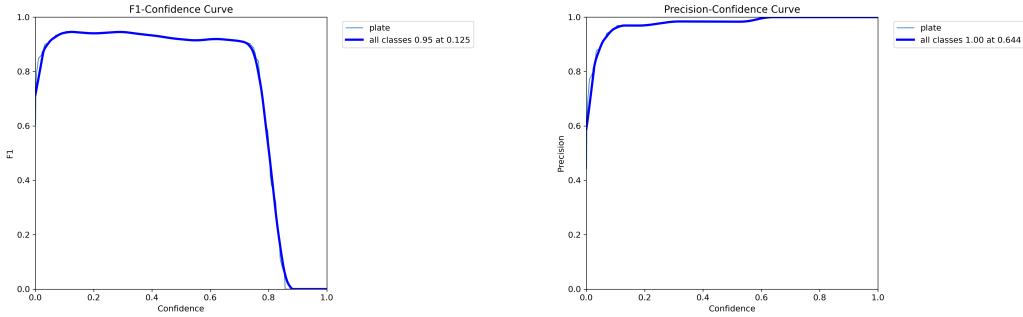


图 4: 置信曲线

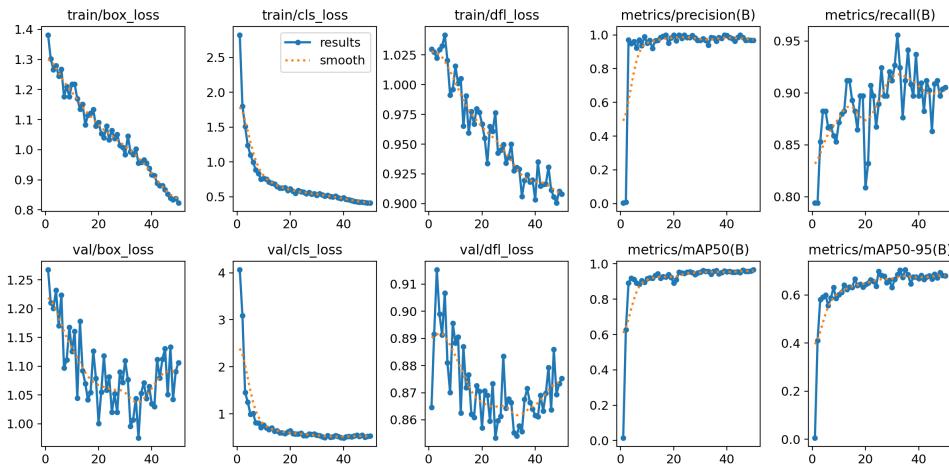


图 5: 每轮训练结果

## 5.2 图像输入模型

训练过后，我们的模型基本上可以做到对输入的图片检测到图片中所有的车牌，对其进行标号，并剪切成单独的图片保存到相应的临时文件夹中。其中为了和后面的文字检测进行更好的衔接，我们再剪切之后为图片描了一层灰边。



图 6: 剪切效果

将预处理和切割后的图像输入到预先训练好的模型中进行处理。模型将对输入的图像进行特征提取和分类，输出相应的识别结果。

### 5.2.1 车牌识别的深度学习训练过程 (train\_continuable.py)

**train\_continuable** 部分代码定义了一个完整的车牌识别系统，包括数据集类、CNN 模型结构、训练流程以及模型保存等步骤。通过训练一个深度卷积神经网络，模型能够识别车牌的类型（普通蓝牌或新能源小型车）以及车牌号的字符。在训练过程中，模型通过不断优化类型和字符的预测，最终生成车牌识别模型，并记录训练过程中的损失值。

**数据集定义 (LicensePlateApp 类)：** class 继承自 torch.utils.data.Dataset，用于加载车牌的图像数据集，通过读取指定的 txt\_file 文件，加载每一行包含图像路径、标签和车牌类型的条目（如“普通蓝牌车”“新能源小车”）并保存信息。并对标签进行处理，实现标签到字符索引的转换，并对车牌类型进行编码映射（如“普通蓝牌映射为 1”）

**卷积神经网络模型定义 (CNN 类)：** 本部分首先使用多个卷积层和池化层提取图像特征，并在卷积层后加入了 ReLU 激活函数和 Dropout 层以增强模型的非线性表达能力并防止过拟合。其次使用了一个全连接层进行车牌类型分类（输出大小为 2，因为只有两个类型）。接着对两种小车定义了两个分类器列表，每个分类器由一层全连接网络组成，输出为每个字符可能的类别（共 65 类包括数字、字母和省份缩写）。最后图像通过特征提取部分，得到特征图后，分别传入车牌类型分类器和字符分类器，如果车牌类型为普通蓝牌，则使用 outputs\_7 进行字符预测；如果是新能源小型车，则使用另一种。

**模型训练及保存记录：** 训练过程中，使用 DataLoader 迭代数据集，每个批次包含图像数据、标签和车牌类型标签。同时使用交叉熵损失函数来计算类型分类和字符分类的损失：对车牌类型进行分类，乘以一个权重 (type\_loss\_weight=2.0) 来提高类型分类的影响；分别计算两种车型的字符分类损失并根据标签的有效性（有效范围为 0-64）计算损失并更；两者相加就能得到总损失并进行反向传播和梯度更新。在每一轮训练结束后保存当前模型的权重并将当前轮次的损失值记录到日志文件中。

### 5.2.2 车牌检测与字符识别 (detect\_aug.py)

**detect\_aug** 部分代码实现了车牌识别系统的车牌检测与字符识别的功能，首先通过车牌检测切割出车牌区域，然后使用预训练的深度学习模型识别车牌的类型和字符。通过将字符索引映射为实际字符，最终识别出车牌号码。

**加载训练模型：** 使用已经训练好的卷积神经网络来加载预训练的模型权重，从而实现车牌类型和字符的识别。

**图像预处理：** 将输入图像调整为 128\*48 的大小，使其符合模型的输入要求，并对图像进行转换，转变为张量 (tensor) 并进行归一化处理。

**车牌检测：** 将图像传入模型并进行前向传播，模型的输出包括了车牌类型、字符的预测（普通蓝牌 7 个，新能源小型车 8 个）并获取每个字符的最大概率类别。

**字符识别：** 根据车牌类型，选择对应的字符输出数组，将每个字符的输出索引转换为对应的字符并输出最终的车牌号码。

**车牌类型判定：** 通过映射字符索引得到最终识别出的车牌字符：如果模型预测车牌类型为“普通蓝牌”，则选择输出 7 个字符；如果为“新能源小型车”，则选择输出 8 个字符。

### 5.2.3 识别结果展示



(a) 测试图片 1

车牌类型: 普通蓝牌  
识别出的字符: 湘VWUJ3N

(b) 测试结果 1



(a) 测试图片 2

车牌类型: 新能源小型车  
识别出的字符: 苏RDC5740

(b) 测试结果 2

验证集车牌类型分类准确率: 100.00%  
验证集字符分类准确率: 95.30%

图 9: 测试结果展示

### 5.3 GUI 界面制作

将模型的输出结果在前端界面上展示给用户。通过友好的用户界面，用户可以直观地查看识别结果。此步骤包括结果的格式化处理、界面设计以及交互功能的实现，确保用户能够方便地获取和理解识别结果。

#### 5.3.1 基本功能及实现

标题，原始图像呈现，处理结果展示，文字提示，上传图片选项，开始处理选项，清空内容选项。

上网搜索知 tkinter 和 PyQt5 两个库均可以实现 GUI 界面可视化，对比之后选择使用 PyQt5，因为 PyQt5 的功能相对更加丰富。确定好所使用的库和所需实现的功能以后就开始着手进行编程。GUI 的主界面使用 QWidget 创建主窗口类，并设置好窗口标题和大小以及样式。顶部放置标题；中间水平布局，放置两个图片区域，左部分显示原始图片，右部分放置对车牌进行截取之后的对比图片；其下一行放置文字输出框，显示识别结果；再下一行放置三个按钮，分别实现上传图片，图片处理，清空页面功能；最下行作为状态栏，显示当前加载状态。

#### 5.3.2 美化需求

设置背景图片，美化按钮样式，统一文字大小。

上网寻找合适的背景图片，将该图片调整大小后使用 PyQt5 的调色盘对象将该图片设置为背景。使用 setStyleSheet 属性的 QPushButton 统一设置按钮的整体背景颜色，按钮中文字颜色，按钮边框，按钮圆角半径，按钮长宽高等性质。最后根据窗口和文本框的大小，以及图片的色彩分布，调整其他文字的大小和颜色。

进一步优化：发现随着窗口大小的拉动，背景图片大小不太合适，加入了背景大小的自适应和自调整功能。



## 5.4 衔接

将训好的模型和对应的调用函数放进 GUI 的空壳里，即将 cut 和 detect 放进 GUI 的框架里。

即在 GUI 中调用 cut 函数，然后对 cut 的结果进行 detect，在处理结果窗口中显示 cut 的结果。

在输出部分调用 detect.py 源文件中的 detect\_license\_plate 函数，将处理好的车牌号和车牌类型放进 GUI 的文字输出里。

## 5.5 exe 生成

在本项目中，为了便于用户使用，我们利用 pyinstaller 库将 Python 脚本转换为可执行文件。通过在命令行中使用以下命令：

```
pyinstaller --onefile --windowed --icon=icon.ico --add-data "background.jpg;." --
           add-data "cnn_epoch_35.pth;." --add-data "best.pt;." gui.py
```

生成了一个不带控制台窗口的 exe 文件。

此外，为了使生成的 exe 文件更具标识性，我们从网上选择了一张与车牌识别相关的摄像头图片，并通过图像工具将其转换为.ico 格式，作为 exe 文件的图标。最终，用户运行该文件时，不仅可以直接启动车牌识别系统，还可以通过图标直观识别其功能。



图 11: 图标图片

## 6 实验结果展示

本实验针对车牌识别的应用场景，完成了从车牌检测到字符识别的完整流程。通过系统处理，输入的车辆图片经过车牌区域定位、图像预处理、字符分割与识别等步骤，最终输出了车牌的具体字符信息。以下实验结果展示了系统的实际运行效果，包括车牌检测、识别区域的提取，以及对应的字符识别结果。



图 12: 实验结果

## 7 实验心得体会

在本次大作业中，小组同学们分工合作共同完成了车牌识别系统的开发。结合课程所学内容，课外知识的搜索查询，以及自学创造，成功实现了车牌检测、定位、字符识别等功能，同时设计开发了 GUI (图形用户界面) 生成可执行的 exe 文件，使得用户便于在本地进行使用。

本次作业的重难点是确保能准确检测到图像中是否存在车牌、车牌的具体位置、将车牌精细的裁剪出来便于后续的识别检测。为此我们采用了传统方法与神经网络结合的方式，使用卷积神经网络 (CNN) 进行车牌的分类和字符的识别。最为困难的部分是实现车牌的精细裁剪。首先我们采用了 Canny 边缘检测的方法对车牌进行处理，发现会有较多背景部分和车牌一起被裁剪出来。然后我们尝试使用 Harr 级联分类器和 Harris 角点定位来进行更精细、贴边的裁剪，但随即发现如此做法会导致程序直接无法检测到图片中的车牌。于是我们最后尝试了 Labelimg 和 Yolov8 结合的方法，先人工手动将车牌从图像中标定出来，然后把图像和车牌信息放在一起训练，得到的模型文件运行后裁剪出的车牌和实际情况吻合得很好。

但是当我们把 Yolo 检测并裁剪出来的照片放入字符识别的模型后，得到的识别结果并不理想。经过组员们的讨论，我们发现是因为用于识别的模型训练集与 Yolo 裁剪后的图像有出入。训练集中，车牌边缘还有一小圈背景，而我们用 Yolo 裁剪出来的车牌则是贴着车牌的边缘。所以我们在裁剪完毕后在得到的车牌图像周围加了一圈用于模拟背景的灰绿色条带，增加后，字符的识别率大幅提高。

本次作业的另一个核心部分是字符识别。我们利用字符库进行匹配，通过对车牌进行字符的分割，将每个字符与字符库中的字符进行比对，从而实现高准确度的车牌字符识别。在这个过程中我们不仅要处理字符识别方面的内容还要确保编写的代码能够正确判断车牌的类别（根据不同的颜色

进行划分) 和字符的数量。

同时为了实现良好的用户体验，我们还设计了车牌识别系统的 GUI 界面，并将所用文件打包成便于使用的 exe 文件。用户可以方便地从本地选择图像并进行识别。同时避免用户输入异常的情况，即只能传入图片形式的文件，不符合规定的文件类型无法输入，确保用户能够进行正确操作。

在生成 exe 文件的时候，我们也遇到了小小的困难。安装的 pyinstaller 在执行 exe 生成命令的过程中被 windows 安全防护认为是病毒被拦截，上网查询知，可能是因为下载的 pyinstaller 的版本过高导致的，经卸载后重新安装低版本 pyinstaller，输入相同指令，成功生成 exe。

通过此次的大作业，我们不仅巩固了以前各项 lab 中学习到的计算机视觉和深度学习的各种相关知识，同时在遇到困难的时候，通过网络的查询和自学相关的知识模型，不断探索解决的途径并优化测试结果，进而提升了我们通过实践解决问题的能力。从无到有通过团队成员之间的紧密合作和有效分工搭建一个车牌识别系统，在获得成就感的同时也提升了团队协作能力和解决问题的能力，最终保证了项目的顺利完成。

## 8 实验代码附录

### 8.1 cut

```
from ultralytics import YOLO
import cv2
import os

def real_cut(image_file,model_path,crop_dir_name):
    # 加载模型，根据实际使用的模型进行替换
    model = YOLO(model_path)

    # image_file = r"multi.jpg"
    # if not os.path.exists(image_file):
    #     print(f"文件不存在: {image_file}")

    # 读取图像
    im0 = cv2.imread(image_file)
    if im0 is None:
        print(f"无法读取图像文件: {image_file}")

    # 使用模型进行预测
    results = model(im0)

    # 裁剪并保存结果
    border_size = 5 # 设置边框宽度
    border_color = [179, 186, 185] # 灰色 (BGR 格式)

    for i, result in enumerate(results):
        for box in result.boxes:
```

```

x1, y1, x2, y2 = map(int, box.xyxy[0])
cropped_img = im0[y1:y2, x1:x2]

# 添加灰色边框到裁剪图像
cropped_img_with_border = cv2.copyMakeBorder(
    cropped_img,
    top=border_size,
    bottom=border_size,
    left=border_size,
    right=border_size,
    borderType=cv2.BORDER_CONSTANT,
    value=border_color
)

# 保存带边框的裁剪图像
crop_file_name = os.path.join(crop_dir_name, f"{i}.jpg")
cv2.imwrite(crop_file_name, cropped_img_with_border)
print(f"保存裁剪图像: {crop_file_name}")
i += 1

return i

```

## 8.2 train

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import numpy as np
import cv2
import os

class LicensePlateDataset(Dataset):
    def __init__(self, dataset_path, txt_file, transform=None):
        self.dataset_path = dataset_path
        self.transform = transform
        self.image_paths = []
        self.labels = []
        self.types = [] # 新增类型记录
        self.max_length = 8 # 最大标签长度为 8

```

```

# Load data from filtered txt file
with open(txt_file, 'r', encoding='utf-8') as file:
    for line in file:
        line = line.strip()
        if not line:
            continue
        parts = line.split(' ')
        if len(parts) == 3:
            img_path, label, plate_type = parts
            if all(char in "京沪津渝冀晋蒙辽吉黑苏浙皖闽赣鲁豫鄂湘粤桂琼川贵云
                            藏陕甘青宁新0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" for char in
                            label):
                self.image_paths.append(os.path.join(self.dataset_path,
                                                    img_path))
                self.labels.append(label)
                self.types.append(plate_type) # 记录车牌类型

    self.char_dict = {char: idx for idx, char in enumerate(
        list("京沪津渝冀晋蒙辽吉黑苏浙皖闽赣鲁豫鄂湘粤桂琼川贵云藏陕甘青宁新
            0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"))}
    self.type_dict = {"普通蓝牌": 0, "新能源小型车": 1} # 新增类型字典

def __len__(self):
    return len(self.image_paths) * 3 # 每张图片生成三种样本: 原图和两种增强图

def __getitem__(self, idx):
    original_idx = idx // 3
    augmentation_type = idx % 3

    img_path = self.image_paths[original_idx]
    label = self.labels[original_idx]
    plate_type = self.types[original_idx]

    img = cv2.imdecode(np.fromfile(img_path, dtype=np.uint8), -1)
    if augmentation_type == 1: # 横向压缩
        img = self.augment_horizontal_compress(img)
    elif augmentation_type == 2: # 纵向压缩
        img = self.augment_vertical_compress(img)

    img = cv2.resize(img, (128, 48)) # 调整图像大小到 128x48
    label_indices = [self.char_dict[char] for char in label] # 假定所有字符已合

```

```

    法
label_indices += [-1] * (self.max_length - len(label_indices)) # 使用 -1 作
    为填充值
label_indices = torch.tensor(label_indices, dtype=torch.long)

type_label = torch.tensor(self.type_dict[plate_type], dtype=torch.long) #

    类型标签

if self.transform:
    img = self.transform(img)

return img, label_indices, type_label # 返回填充后的标签和类型标签

def augment_horizontal_compress(self, img):
    h, w = img.shape[:2]
    new_w = int(w * 0.8)
    compressed = cv2.resize(img, (new_w, h))
    result = np.full((h, w, 3), 128, dtype=np.uint8) # 灰色填充
    result[:, (w - new_w) // 2:(w - new_w) // 2 + new_w] = compressed
    return result

def augment_vertical_compress(self, img):
    h, w = img.shape[:2]
    new_h = int(h * 0.8)
    compressed = cv2.resize(img, (w, new_h))
    result = np.full((h, w, 3), 128, dtype=np.uint8) # 灰色填充
    result[(h - new_h) // 2:(h - new_h) // 2 + new_h, :] = compressed
    return result

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

```

```

        nn.Dropout(0.5),

        nn.Conv2d(32, 64, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.Conv2d(64, 64, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2, 2),
        nn.Dropout(0.5)
    )

    self.type_classifier = nn.Sequential(
        nn.Flatten(),
        nn.Linear(64 * 6 * 16, 128), # 调整全连接层输入大小为实际大小
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(128, 2) # 两种类型：普通蓝牌、新能源小型车
    )

    self.classifiers_7 = nn.ModuleList([nn.Sequential(
        nn.Flatten(),
        nn.Linear(64 * 6 * 16, 256),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(256, 65)
    ) for _ in range(7)])

    self.classifiers_8 = nn.ModuleList([nn.Sequential(
        nn.Flatten(),
        nn.Linear(64 * 6 * 16, 256),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(256, 65)
    ) for _ in range(8)])
}

def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), -1) # 展平特征图
    plate_type_output = self.type_classifier(x) # 车牌类型预测
    outputs_7 = [classifier(x) for classifier in self.classifiers_7] # 普通蓝牌
    outputs_8 = [classifier(x) for classifier in self.classifiers_8] # 新能源小
        型车
    return plate_type_output, outputs_7, outputs_8

if __name__ == "__main__":

```

```

dataset_path = "./DataSet"
txt_file = "./DataSet/filtered_train.txt"

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

dataset = LicensePlateDataset(dataset_path, txt_file, transform=transform)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=-1)

type_loss_weight = 2.0 # 提高类型损失的权重

# 尝试加载已保存的 checkpoint
checkpoint_path = "train_model_aug/cnn_epoch_10.pth" # 替换为实际路径
start_epoch = 0 # 默认从头开始
if os.path.exists(checkpoint_path):
    print(f"加载 checkpoint: {checkpoint_path}")
    checkpoint = torch.load(checkpoint_path)
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    start_epoch = checkpoint['epoch'] # 恢复起始 epoch
    print(f"从 epoch {start_epoch} 恢复训练")

epochs = 35
os.makedirs("train_model_aug", exist_ok=True)
for epoch in range(start_epoch, epochs): # 从恢复点继续训练
    model.train()
    total_loss = 0
    for images, labels, type_labels in dataloader:
        images = images.to(device)
        labels = labels.to(device)
        type_labels = type_labels.to(device)

        optimizer.zero_grad()
        plate_type_output, outputs_7, outputs_8 = model(images)

```

```

# 类型分类损失
type_loss = criterion(plate_type_output, type_labels) *
    type_loss_weight

# 车牌类型的索引
blue_indices = (type_labels == 0).nonzero(as_tuple=True)[0]
green_indices = (type_labels == 1).nonzero(as_tuple=True)[0]

# 普通蓝牌字符损失
blue_char_loss = 0
if len(blue_indices) > 0:
    valid_mask_blue = (labels[blue_indices, :] >= 0) & (labels[
        blue_indices, :] < 65)
    for j in range(7):
        valid_indices = valid_mask_blue[:, j]
        if valid_indices.sum().item() > 0:
            blue_char_loss += criterion(outputs_7[j][blue_indices][
                valid_indices],
                labels[blue_indices, j][valid_indices])
            labels[blue_indices, j][valid_indices])

# 新能源小型车字符损失
green_char_loss = 0
if len(green_indices) > 0:
    valid_mask_green = (labels[green_indices, :] >= 0) & (labels[
        green_indices, :] < 65)
    for j in range(8):
        valid_indices = valid_mask_green[:, j]
        if valid_indices.sum().item() > 0:
            green_char_loss += criterion(outputs_8[j][green_indices][
                valid_indices],
                labels[green_indices, j][
                    valid_indices])

# 总字符损失
char_loss = (blue_char_loss + green_char_loss) / len(images)

# 总损失
loss = type_loss + char_loss
loss.backward()
optimizer.step()

```

```

    total_loss += loss.item()

    average_loss = total_loss / len(dataloader)
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {average_loss:.4f}")

    # Save model and loss after each epoch
    model_save_path = os.path.join("train_model_aug", f"cnn_epoch_{epoch+1}.pth")
    checkpoint = {
        'epoch': epoch + 1,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': average_loss
    }
    torch.save(checkpoint, model_save_path)

    with open(os.path.join("train_model_aug", "loss_log.txt"), "a") as log_file
        :
        log_file.write(f"Epoch [{epoch+1}/{epochs}], Loss: {average_loss:.4f}\n")

print("Training complete. Models and logs are saved in 'train_model' directory")
.
```

### 8.3 detect

```

import torch
import cv2
import numpy as np
from torchvision import transforms
from train_continuable import CNN # 确保模型类与训练代码中一致


def detect_license_plate(image_path, model_path):
    """
    检测图像中的车牌，并返回车牌类型和识别出的字符。
    :param image_path: 图像文件路径
    :param model_path: 训练好的模型权重路径
    :return: 车牌类型和识别出的字符
    """
    # 加载模型

```

```

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)

# 加载模型权重
checkpoint = torch.load(model_path, map_location=device, weights_only=True)
model.load_state_dict(checkpoint['model_state_dict']) # 仅加载模型权重
model.eval()

# 预处理图像
img = cv2.imdecode(np.fromfile(image_path, dtype=np.uint8), -1)
if img.shape[-1] == 4:
    img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
img_resized = cv2.resize(img, (128, 48)) # 确保图像大小为 128x48
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])
img_tensor = transform(img_resized).unsqueeze(0).to(device) # 添加 batch 维度

# 前向传播
with torch.no_grad():
    plate_type_output, outputs_7, outputs_8 = model(img_tensor)

# 车牌类型
plate_type = "普通蓝牌" if plate_type_output.argmax(dim=1).item() == 0 else "新能源小型车"

# 字符预测
if plate_type == "普通蓝牌":
    outputs = outputs_7
    char_count = 7
else:
    outputs = outputs_8
    char_count = 8

char_indices = [output.argmax(dim=1).item() for output in outputs[:char_count]]

# 映射字符索引到实际字符
char_map = "京沪津渝冀晋蒙辽吉黑苏浙皖闽赣鲁豫鄂湘粤桂琼川贵云藏陕甘青宁新
0123456789ABCDEFGHIJKLMNPQRSTUVWXYZ"
recognized_chars = ''.join([char_map[idx] for idx in char_indices])

```

```

    return plate_type, recognized_chars

if __name__ == "__main__":
    # 示例使用
    image_path = r"cut_imgs\19_0.jpg" # 替换为实际测试图像路径
    model_path = r"train_model_aug\cnn_epoch_35.pth" # 替换为实际模型权重路径

    plate_type, recognized_chars = detect_license_plate(image_path, model_path)
    print(f"车牌类型: {plate_type}")
    print(f"识别出的字符: {recognized_chars}")

```

## 8.4 changeSuffix

```

import os
from PIL import Image

def convert_to_jpg(input_path, output_path=None):
    """
    将图像转换为 JPG 格式
    :param input_path: 输入图像文件路径
    :param output_path: 输出图像文件路径 (如果为 None, 则自动生成)
    """
    # 打开图像
    img = Image.open(input_path)

    # 自动生成输出路径
    if output_path is None:
        # 替换后缀为 .jpg
        base_name = os.path.splitext(input_path)[0]
        output_path = f"{base_name}.jpg"

    # 将图像保存为 JPG 格式
    img.convert("RGB").save(output_path, "JPEG")
    print(f"图像已转换并保存为: {output_path}")

    # 删除原始图像文件
    os.remove(input_path)
    print(f"原始图像已删除: {input_path}")

# 示例使用

```

```

if __name__ == "__main__":
    input_file = r"multi.png" # 替换为实际的输入图像路径
    convert_to_jpg(input_file)

```

## 8.5 GUI

```

import sys
import os
import cv2
import shutil
from PyQt5.QtWidgets import (
    QApplication, QLabel, QPushButton, QVBoxLayout, QHBoxLayout,
    QWidget, QFileDialog, QGroupBox, QTextEdit, QStatusBar, QLayoutItem
)
from PyQt5.QtGui import QPixmap, QFont, QFontMetrics, QPalette, QBrush
from PyQt5.QtCore import Qt
import detect_aug
import change_suffix
import real_cut

# 将资源文件打包到可执行文件中
def get_resource_path(relative_path):
    try:
        # PyInstaller 创建一个临时文件夹并将资源存储在_MEIPASS 中
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)

# def get_resource_path(relative_path):
#     return relative_path

class LicensePlateApp(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    # 初始化界面
    def init_ui(self):
        self.setWindowTitle("智能车牌识别系统")
        self.setGeometry(100, 50, 1200, 1000) # 更大的窗口尺寸

```

```
# 设置背景图片
self.set_background_image()

self.setStyleSheet("""
    QPushButton {
        background-color: #0078d7;
        color: white;
        border: none;
        padding: 10px; /* 保证按钮的内边距相同 */
        font-size: 25px;
        font-weight: bold;
        border-radius: 8px;
        width: 60px; /* 设置固定的宽度 */
        height: 60px; /* 设置固定的高度，确保按钮方形 */
        margin: 10px; /* 设置按钮之间的间距 */
    }
    QPushButton:hover {
        background-color: #005a9e;
    }
    QTextEdit {
        border: 1px solid #ccc;
        border-radius: 5px;
        padding: 8px;
        font-size: 25px;
        font-weight: bold;
    }
    QLabel {
        font-size: 100px;
    }
""")
# 主布局
main_layout = QVBoxLayout()

# 顶部标题
title_label = QLabel("智能车牌识别系统")
title_label.setAlignment(Qt.AlignCenter)
# 设置标题样式
title_label.setStyleSheet("""
    font-size: 30px;
    font-weight: bold;
    color: #003366; /* 设置字体颜色为深蓝色 */
""")
```

```

border:1px solid #003366; /* 为标题添加深蓝色边框 */
padding: 4px; /* 给标题添加内边距, 确保文字和边框之间有空间 */
border-radius: 4px; /* 为边框添加圆角效果 */
""")
main_layout.addWidget(title_label)

# 图片区域
image_layout = QHBoxLayout()
self.original_image = QLabel("原始图像")
self.original_image.setAlignment(Qt.AlignCenter)
self.original_image.setStyleSheet("border: 1px solid #ccc; background: #
e0e0e0; font-size: 30px;")
self.original_image.setFixedSize(500, 400) # 扩大显示区域

self.result_image = QLabel("处理结果")
self.result_image.setAlignment(Qt.AlignCenter)
self.result_image.setStyleSheet("border: 1px solid #ccc; background: #
e0e0e0; font-size: 30px;")
self.result_image.setFixedSize(500, 400) # 扩大显示区域

image_layout.addWidget(self.original_image)
image_layout.addWidget(self.result_image)
main_layout.addLayout(image_layout)

# 结果输出区域
result_box = QGroupBox("识别结果")
result_layout = QVBoxLayout()
self.result_text = QTextEdit()
self.result_text.setReadOnly(True)
result_layout.addWidget(self.result_text)
result_box.setLayout(result_layout)
result_box.setStyleSheet("""
QGroupBox {
    font-size: 22px;
}
QGroupBox::title {
    color: white; /* 设置标题文字颜色为白色 */
}
""")
main_layout.addWidget(result_box)

# 按钮区域

```

```

button_layout = QBoxLayout()
self.upload_button = QPushButton("上传图片")
self.process_button = QPushButton("开始处理")
self.clear_button = QPushButton("清空内容")

self.upload_button.clicked.connect(self.upload_image)
self.process_button.clicked.connect(self.process_image)
self.clear_button.clicked.connect(self.clear_content)

button_layout.addWidget(self.upload_button)
button_layout.addWidget(self.process_button)
button_layout.addWidget(self.clear_button)
main_layout.addLayout(button_layout)

# 状态栏
self.status_bar = QStatusBar()
self.status_bar.showMessage("准备就绪")
self.status_bar.setStyleSheet("font-size: 16px;")
main_layout.addWidget(self.status_bar)

self.setLayout(main_layout)

# 设置背景图像并确保其按窗口大小自适应
def set_background_image(self):
    palette = self.palette()
    pixmap = QPixmap(get_resource_path("background.jpg"))
    pixmap = pixmap.scaled(self.size(), Qt.KeepAspectRatioByExpanding)
    palette.setBrush(QPalette.Background, QBrush(pixmap))
    self.setPalette(palette)

# 窗口大小改变时调整字体大小
def resizeEvent(self, event):
    self.adjust_text_font()
    self.set_background_image()
    super().resizeEvent(event)

# 根据文本框大小调整字体大小
def adjust_text_font(self):
    text_edit_width = self.result_text.width()
    text_edit_height = self.result_text.height()

    # 计算字体大小，确保字体适应文本框

```

```

font = self.result_text.font()
font_metrics = QFontMetrics(font)

# 使用文本框宽高计算字体大小
max_font_size = min(text_edit_width // font_metrics.averageCharWidth(),
                     text_edit_height // font_metrics.height())

# 设置字体大小
font.setPointSize(max_font_size)
self.result_text.setFont(font)

# 上传图片
def upload_image(self):
    try:
        file_path, _ = QFileDialog.getOpenFileName(self, "选择图片", "", "Images (*.png *.jpg *.bmp)")
        if file_path:
            # 检查文件格式
            if not file_path.lower().endswith('.jpg'):
                # 如果不是 JPG 格式, 使用 change_suffix 转换
                new_file_path = change_suffix.convert_to_jpg(file_path)
                if new_file_path:
                    file_path = new_file_path
                else:
                    self.result_text.append("转换格式失败!")
                    self.status_bar.showMessage("转换格式失败")
                    return
            pixmap = QPixmap(file_path)
            self.original_image.setPixmap(pixmap.scaled(500, 400, Qt.KeepAspectRatio))
            self.result_text.append(f"图片已加载: {file_path}")
            self.image_path = file_path
            self.status_bar.showMessage("图片加载成功")
        else:
            self.status_bar.showMessage("未选择图片")
    except Exception as e:
        self.result_text.append(f"文件上传出现错误: {str(e)}")
        self.status_bar.showMessage("文件上传失败")

# 清空布局
def clear_layout(self, layout):
    if layout is not None:

```

```

while layout.count():
    child = layout.takeAt(0)
    if child.widget() is not None:
        child.widget().deleteLater()

# 处理图片
def process_image(self):
    try:
        if hasattr(self, 'image_path'):
            self.result_image.setPixmap(QPixmap())
            if self.result_image.layout():
                # 清除所有子部件
                while self.result_image.layout().count():
                    item = self.result_image.layout().takeAt(0)
                    if item.widget():
                        item.widget().deleteLater()
                # 删除布局
                QWidget().setLayout(self.result_image.layout())
            self.result_image.setText("处理结果") # 恢复默认文本
            # 使用 real_cut 识别并裁剪车牌
            folder_name = "cut_temp"
            if not os.path.exists(folder_name):
                os.makedirs(folder_name)
            n = real_cut.real_cut(self.image_path, get_resource_path("best.pt"),
                                  "cut_temp")
            cropped_images = [f"cut_temp/{i}.jpg" for i in range(n)]
            if cropped_images:
                # 清空 result_image 和 result_text
                self.clear_layout(self.result_image.layout())
                self.result_text.clear()

            # 创建一个新的 QWidget 作为容器
            container_widget = QWidget()
            layout = QVBoxLayout(container_widget)

            for i, cropped_img in enumerate(cropped_images):
                # 创建一个新的 QLabel 来显示裁剪后的车牌图片
                label = QLabel()
                pixmap = QPixmap(cropped_img)
                if pixmap.isNull():
                    self.result_text.append(f"无法加载图片: {cropped_img}")
                    continue

```

```

        label.setPixmap(pixmap.scaled(500, 400, Qt.KeepAspectRatio))
        layout.addWidget(label)

        # 使用 detect_license_plate 识别车牌类型和字符
        plate_type, recognized_chars = detect_aug.
            detect_license_plate(cropped_img, get_resource_path("cnn_epoch_35.pth"))
        self.result_text.append(f"处理完成! \n车牌号: {recognized_chars}\n车牌类型: {plate_type}")
        self.status_bar.showMessage("处理完成")

        # 将容器 widget 设置到 result_image
        self.result_image.setLayout(layout)
    else:
        self.result_text.append("未检测到车牌! ")
        self.status_bar.showMessage("处理失败: 未检测到车牌")
    else:
        self.result_text.append("请先上传图片! ")
        self.status_bar.showMessage("处理失败: 未上传图片")
except Exception as e:
    self.result_text.append(f"图像处理出现错误: {str(e)}")
    self.status_bar.showMessage("处理失败")
finally:
    # 删除临时文件夹
    if os.path.exists(folder_name):
        shutil.rmtree(folder_name)

# 清空内容
def clear_content(self):
    try:
        # if self.original_image.pixmap():
        # self.original_image.setPixmap(QPixmap()) # 清空图片而不是清除文本

        # # 清空 result_image 的布局
        # layout = self.result_image.layout()
        # if layout is not None:
        # while layout.count():
        # child = layout.takeAt(0)
        # if child.widget() is not None:
        # child.widget().deleteLater()
        # self.result_image.setLayout(QVBoxLayout()) # 重新设置布局
    
```

```

# self.result_text.clear()
# self.status_bar.showMessage("内容已清空")
# 清空原始图像
if self.original_image.pixmap():
    self.original_image.setPixmap(QPixmap())
    self.original_image.setText("原始图像") # 恢复默认文本

# 清空结果图像
self.result_image.setPixmap(QPixmap())
if self.result_image.layout():
    # 清除所有子部件
    while self.result_image.layout().count():
        item = self.result_image.layout().takeAt(0)
        if item.widget():
            item.widget().deleteLater()
    # 删除布局
    QWidget().setLayout(self.result_image.layout())
self.result_image.setText("处理结果") # 恢复默认文本

# 清空文本结果
self.result_text.clear()

# 更新状态栏
self.status_bar.showMessage("内容已清空")
except Exception as e:
    self.result_text.append(f"内容清除出现错误: {str(e)}")
    self.status_bar.showMessage("内容清除失败")

if __name__ == "__main__":
    # 创建应用程序和窗口对象
    app = QApplication(sys.argv)
    window = LicensePlateApp()
    # 显示窗口
    window.show()
    # 运行应用程序，并监听事件
    sys.exit(app.exec_())

```