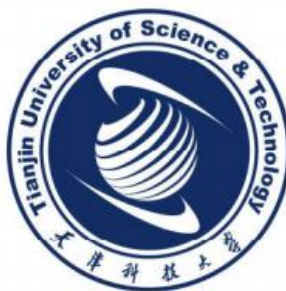

天津科技大学

毕业设计

(2023 届 · 本科)



毕 设 题 目: 基于深度学习的车辆追踪系统的设计与实现

学 名: 19201111

学 生 姓 名: 鲍其瑞

学 院 名 称: 人工智能学院

专 业 名 称: 计算机科学与技术 (信息处理)

指 导 教 师: 李伟

2023 年 5 月

基于深度学习的车辆追踪系统的设计与实现

Design and Implementation of Vehicle Tracking System Based on Deep Learning

专 业: 计算机科学与技术(信息处理)

姓 名: 鲍其瑞

指 导 教 师: 李伟

申请学位级别: 学 士

论文提交日期: 2023 年 5 月 26 日

学位授予单位: 天津科技大学

天津科技大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：鲍世瑞

日期：2023 年 5 月 27 日

天津科技大学

学位论文使用授权书

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于：

☒ 公开论文

☐ 内部论文，保密 ☐ 1 年 / ☐ 2 年 / ☐ 3 年，过保密期后适用本授权书。

☐ 秘密论文，保密 ____ 年（不超过 10 年），过保密期后适用本授权书。

☐ 机密论文，保密 ____ 年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“√”）

学位论文作者签名：鲍世瑞

指导教师签名：李伟

日期：2023 年 5 月 27 日

日期：2023 年 5 月 27 日

日

天津科技大学本科毕业设计（论文）任务书

学 院：人工智能学院 专 业：计算机科学与技术（信息处理）

学生学号：19201111 学生姓名：鲍其瑞 指导教师姓名：李伟

完成期限：2023 年 01 月 10 日至 2023 年 06 月 04 日

一、题目名称：基于深度学习的车辆追踪系统的设计与实现

二、设计(论文)内容及要求：

毕业设计内容：

设计与实现基于深度学习的车辆追踪系统，主要实现检测某摄像头拍摄区域内是否有指定车辆经过，在摄像条件良好的情况下，利用 opencv 确定车牌所在的区域，分割各文字区域，识别各自的汉字，英文数字并得出结果。

该毕业设计主要要求如下：

将车牌照识别和车型识别结合起来，在条件不满足的情况下，可以结合车型识别快速缩短甄别范围。

1. 利用基于 opencv 的图像处理和车牌区域定位，基于深度网络和 tensorflow 的文字识别，基于 opencv 的图像处理，基于 python 的算法实现等技术完成毕设。

2. 开学第 1-3 周，完成与专业有关的毕业实习或调研，撰写不少于 2500 字的实习报告或调研报告。

3. 查阅相关文献，完成一篇不少于 5000 汉字的外文文献翻译。

4. 调研项目需求，撰写开题报告。

5. 根据毕业设计内容撰写毕业论文，要求结构清晰，语言描述到位，展现完整的开发流程，不对他人论文进行抄袭剽窃，字数为 2 万字以上。

参考文献：

[1] 路明玉. 通过 MLP 网络结构进行手写数字识别的系统设计[J]. 科技资讯, 2019, 17(14):2.

[2] Zagoruyko, Sergey, and Nikos Komodakis. "Learning to compare image patches via convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[3] 罗志勇, 郭晓惠, 宦红伦,等. 一种基于 Sobel 算子的图像边缘检测方法:, CN108242060A[P]. 2018.

[4] Bradski, Gary, and Adrian Kaehler. "OpenCV." Dr. Dobb's journal of software tools 3.2 (2000).

[5] 范大昭, 董杨, and 张永生. "卫星影像匹配的深度卷积神经网络方法." (2018).

指导教师：李伟

填写日期： 2023 年 1 月 10

日

注：本任务书发给学生，毕业设计（论文）完成后装入毕业设计（论文）档案袋。

摘 要

车辆追踪系统对于当今社会安全等现实问题具有重大意义。在未来，车辆追踪系统将对犯罪车辆，特殊车辆的定位和追踪，智能驾驶等现实问题的发展产生巨大的推进作用。本毕设拟解决车辆追踪系统中单个摄像头拍摄范围内是否存在指定车辆通过的问题。该系统主要实现，在汽车图像中能清晰拍摄车牌的情况下，使用 `opencv` 技术确定车牌所在区域，划分各文本区域，分别利用汉字和英文数字模型识别车牌照中的字符并输出结果。其中模型训练所需要的数字和英文，汉字数据集分别自行搜集，各部分数据集大小约为几百张图片。模型设计和训练方面，本毕设采用 `mlp` 技术设计英文数字，采用 `cnn` 模型设计汉字识别模型，模型的准确率均能够达到 96% 以上。最后本毕设分析了该系统在实际情况中的缺点，并提出了进一步优化的畅想。本系统主要使用基于 `opencv` 的图像处理，基于深度网络和 `tensorflow` 的字符识别，基于 `Flask` 的接口，基于 `HTML` 的网页等技术完成。

关键词：深度学习； 车牌照识别； 图像处理；

ABSTRACT

Vehicle tracking system is of great significance to social security and other practical problems. In the future, vehicle tracking system will greatly promote the development of criminal vehicles, special vehicle positioning and tracking, intelligent driving and other practical problems. This design aims to solve the problem of whether a specified vehicle passes within the shooting range of a single camera in the vehicle tracking system. The main implementation of this system is to use opencv technology to determine the area where the license plate is located, divide each text area, identify the characters in the license plate with Chinese characters and English digital models respectively and output the results when the license plate can be clearly photographed in the car image. The data sets of digital, English and Chinese characters required for model training were collected by themselves, and the size of each part of the data set was about several hundred pictures. In terms of model design and training, mlp technology is used to design English numbers, and cnn model is used to design Chinese character recognition model. The accuracy of the model can reach more than 96 percent. Finally, this paper analyzes the shortcomings of the system in the actual situation, and puts forward further optimization ideas. This design mainly uses image processing based on opencv, character recognition based on deep network and tensorflow, interface based on FLask, and web page implementation based on HTML.

Keywords: deep learning; License plate recognition; opencv;

目 录

第一章 绪论	1
第一节 课题背景及意义	1
第二节 行业研究现状	3
第二章 课题可行性分析	5
第三章 车牌照识别算法	7
第一节 数据集的选择	7
第二节 模型的选择和设计	8
第三节 车牌字符的分割和处理	16
第四节 字符的分离和处理	25
第五节 预测和识别	27
第四章 追踪系统的具体设计与实现	28
第五章 展望	31
第一节 系统未来发展分析	31
第二节 优化系统方法畅想	31
第三节 优化算法的详细分析	31
第四节 算法优化方式总结	38
结 论	39
参考文献	40
致 谢	42

第一章 绪论

第一节 课题背景及意义

随着信息化和人工智能的不断进步，智能驾驶和车辆追踪技术越来越备受关注。目前中国现有交通工具中汽车数量已超过 3.19 亿辆，其中新能源汽车数量超过 1300 万辆，并且预计今年能够超过 1400 万辆。预计到 2023 年的中后期，中国内陆的主要汽车销售市场中 4s 店和汽车销售公司的总销量有可能突破 2700 万辆，特别是最近发展起来的新能源汽车的销售规模和销售量有可能将达到 900 万辆。^[1]这些数据表明，未来汽车管理和规划将对社会发展和城市管理愈发重要。车辆与车辆、汽车和环境、汽车和人之间的互相识别、互相定位和信息共享将随着汽车智能化的发展越来越密切。为了进一步促进这一目标，对车辆追踪系统进行进一步研究是非常必要的。

市面上的追踪系统主要使用 GPS、4G 等通信技术、北斗实时定位和空间定位技术以及 LBS 定位技术^[4]等。其中，GPS、北斗定位技术是最常用的一种技术，条件简单，效果稳定^{[2][3]}。4G 等通信技术可以实现指定车辆的位置信息和方向信息的实时传输、远程监管和间断性定位。可以在没有卫星信号的情况下及时了解到车辆的具体位置信息。LBS 定位技术是一种新型的定位技术，可以通过不同地方的基站的信号来实现车辆的实时定位和及时追踪。这些定位技术的应用可以帮助车主或车队管理人员实时了解车辆的位置和状态，从而更好地管理车辆和提高运输效率。

基于 GPS 的车辆追踪系统的技术方法主要包括使用 GPS 卫星定位系统获取车辆的位置信息，并通过无线通信技术传输到远程服务器，以实现实时监控和追踪。该系统主要涉及实时数据存储技术和处理技术，地图显示技术，报警技术等。数据存储和处理技术可以将车辆位置信息存储到数据库中，并进行数据处理和分析，以便实现车辆轨迹回放、报警等功能。地图展示技术可将地图上实时展示车辆位置信息，以方便用户或者工作人员进行实时的监控和追踪。这种地图展示技术可以通过监控特定的行为，例如超速、急刹车、急加速等，在当车辆发生异常行为时，系统会自动检测并发出相应的报警信息，提醒用户及时处理。该功能可以帮助用户及时发现车辆发生的问题，例如闯入非法区域，失速等问题。以上是基于 GPS 的车辆追踪系统的主要技术方法，具体实现方式可能会因应用场景和需求的不同而有所差异。

利用 4G 技术的车辆追踪系统的技术方法主要包括车载终端设备、服务器端软件、数据传输协议和数据处理算法等方面。车载终端设备是车辆追踪系统的核心部件，；利用 5G/4G/3G 等向服务器上传传输车辆方位数据。在服务器端，接收从终端设备传输上来的车辆位置编码信息，并进行处理和存储。数据传输协议已由专门的标准制定单位制定，车辆终端设备和远程服务器之间的数据传输方式和通信方式可直接使用确定好的数据传输协议。数据处理算法负责对车辆所在的位置信息进行实时解码和处理，从中分析出需要的的位置信息。基于 GPRS/3G/4G 通信技术的车辆追踪系统可以实时获取车辆的视频信息，但需要大量的存储空间和带宽。

数据挖掘技术：可以预测车辆的行驶轨迹，但需要在车辆上安装 RFID 设备，需要权衡利弊选择最适合的技术方案。未来可能会出现更先进的技术方

案，实现更好的车辆追踪和管理。

计算机视觉技术：利用图像处理，分析，识别等技术对车辆进行实时追踪，通常由摄像头、图像处理算法和追踪算法组成。首先通过摄像头等工具获取车辆的图像信息，然后利用算法提取出里面含有的车辆特征信息。追踪算法利用特征信息对车辆进行定位追逐。该系统可以应用于交通管理、安防监控等领域^[5]。

通过对上述前沿研究进行进一步的分析，笔者发现对于车辆追踪系统来说，目前大多数技术主要的方向仍然是车辆与道路协同交通，车辆终端与总控制中心通信，利用卫星对车辆进行定位等等，这些技术方案都需要在车辆上部署终端，并且需要进行通信。在大多数现实场景中目前很难满足这些条件，追踪车辆可能由于被人故意破坏、电池消耗和设备使用期限等问题很难长期具备稳定的终端部署条件。由于车路协同的发展成本高，控制系统不够成熟等等因素，依赖于传感器，控制等设备的车辆追踪系统很难实现，因此目前大多数研究都停留在理论和实验阶段，并没有全面快速部署和使用在中国大大小小的高速公路上。例如，GPS 定位的实用性场景不够全面，并且极其依赖车辆上是否部署终端设备，对于犯罪车辆追踪，未知车辆定位等等问题，并不具备解决能力。为了加快车辆追踪系统的发展和实现，利用现有设备对车辆进行追踪，并不依赖车辆上的终端设备的车辆定位系统的进一步开发是很有必要的。

根据市场上的调查公司对中国道路的调查情况分析，发现目前中国最多的道路监管设备是摄像头，中国道路，学校，街头等等各个地方目前已安装了 1.76 亿个摄像头，这个数字在美国仅仅大约 5000 万个。利用摄像头对车辆进行实时追踪，不仅可以覆盖中国大多数的道路交通环境，而且不依赖追踪车辆的

终端设备部署，对于目前中国道路和汽车的改动较小，且具备很好的覆盖效果。因此对于如何充分利用公路上的大量摄像头，对犯罪车辆等进行大致定位和实时追踪，对于车辆追踪技术的进一步发展具有非常重要的意义。

第二节 行业研究现状

车辆追踪作为未来车辆定位，车辆管理过程中的重要技术之一，这些年来获得了广泛的关注，中外学者在车辆识别领域中做了非常多的研究，尤其是在利用计算机视觉进行车辆追踪领域做出了很多卓越贡献。很多研究已经颇具效果。在此简单介绍行业上重要的车辆追踪系统的研究成果，并根据这些研究成果确定本毕设的研究方法。

M. Enzweiler 和 D. M. Gavrila 在 2006 年在国际计算机视觉会议（ICCV）发表论文“Real-time multiple vehicle detection and tracking from a moving vehicle”。论文介绍了一种可在移动车辆上实现的实时多车辆检测和跟踪的方法。该方法基于一种新的车辆检测算法，该算法使用了一种基于 Haar 特征的级联分类器，并结合了一些先进的技术，如背景建模和运动分析。该方法使用基于卡尔曼滤波器的跟踪算法，实现对车辆的准确跟踪。方法在实验中取得了很好的效果，可以实现高效、准确的车辆检测和跟踪^[6]。

Junliang Xing, Xinchu Shi, and Jian Yang 于 2012 年在 IEEE Transactions on Image Processing 上发表论文“Visual tracking with online multiple instance learning”。该论文主要使用多个实例来表示目标，每个实例都是从目标周围的图像块中提取的。该算法会不断更新实例以适应目标变化，可应用于视频监控、自动驾驶和机器人导航等领域。该方法利用多个实例来表示车辆的不同状态，通过在线学习来更新模型，从而实现对车辆的跟踪。该方法具有较高的准确性和实时性，适用于车辆跟踪等实时应用场景。论文中详细介绍了该方法的技术路线和实验结果，对该方法的优缺点进行了分析和讨论^[7]。

Jianbo Shi 和 Carlo Tomasi 在 1994 年于 IEEE Conference on Computer Vision and Pattern Recognition 上发表论文《Combining multiple motion estimates for vehicle tracking》。该论文提出了一种将多个运动估计结合起来进行车辆跟踪的方法。该论文提出了一种结合多个运动估计的新方法，提高了车辆跟踪的准确性和鲁棒性。作者介绍了车辆跟踪的背景和现有方法，然后提出了基于光流、特征点和深度学习的估计方法。实验结果表明，该方法可以显著提高车辆跟踪的准确性和鲁棒性，特别是在复杂场景下。该论文对实际应用中的车辆跟踪具有重要意义^[8]。

Zhu, Li 等人于 2018 年在 IEEE Transactions on Intelligent Transportation Systems 上发表论文 “Big data analytics in intelligent transportation systems: survey.”对智能交通系统中的大数据分析技术进行了分类和介绍，包括数据挖掘、机器学习、深度学习等。最后，文章总结了当前智能交通系统中大数据分析的研究现状和未来发展方向。智能交通系统中涉及的数据种类繁多，包括交通流量、车辆位置、路况信息等，这些数据的规模庞大、速度快、种类多样，同时，智能交通系统中的数据分析需要考虑到实时性、准确性和可靠性等因素，这也增加了数据分析的难度。为了解决挑战，文章介绍了多种大数据分析技术在智能交通系统中的应用。数据挖掘技术可用于交通流量预测、异常检测；机器学习技术可用于交通信号控制、路径规划；深度学习技术可用于交通图像识别、行为分析。这些技术的应用可以帮助智能交通系统更好地处理和分析数据，提高交通系统的效率 and 安全性。总的来说，这篇论文详细介绍了智能交通系统中大数据分析的应用和挑战，对于研究智能交通系统的学者和工程师具有一定的参考价值^[9]。

论文 “Vehicle tracking using a hybrid approach of particle filter and Kalman filter”，介绍了一种基于粒子滤波器和卡尔曼滤波器的混合方法来进行车辆跟踪的技术。该方法结合了两种滤波器的优点，能够更准确地估计车辆的位置和速度，并且能够在车辆出现遮挡或者其他干扰情况下保持跟踪。

“Vehicle Tracking System Based on License Plate Recognition” 是一篇发表于 2017 年 IEEE 国际会议上的论文，作者为 Jianhua Zhang。该论文提出了一种基于车牌识别的车辆跟踪系统，该系统可以通过识别车辆的车牌号码来跟踪车辆的位置和行驶轨迹。该系统使用基于深度学习的车牌识别算法，在复杂环境下准确识别车牌号码。同时，该系统还采用了一种基于 GPS 和 GSM 的定位技术，可以实时跟踪车辆的位置和行驶轨迹。该系统可以广泛应用于交通管理、车辆租赁、物流配送等领域^{[10][11]}。

通过对中外的研究者在车辆追踪方面的算法和工程的研究，发现大多数的研究都偏向于特定条件的算法，需要在特定条件下才可以实现车辆定位。其普遍需要大量的算力进行模型的训练，部署的成本高，代价大。另有部分研究着力于车辆与道路协同，需要对现有的交通监管系统进行大幅度修改，成本极其高昂。低成本在现有的设备上部署，并实现车辆追踪对于车辆追踪系统的推广具有非常重大的意义。

第二章 课题可行性分析

本毕设拟开发车辆追踪系统相关算法，其主要是为了帮助在高速，公路，等各种摄像头拍摄的视频和图片中识别出特定的车辆。利用摄像头对车辆进行识别对于车辆追踪和车辆定位具有非常重要的作用。其中首冲其中的技术就是车牌照识别。该技术目前在解决车辆进出管理，识别车辆身份标志进而了解车辆信息，车辆定位和车辆追踪系统方面发挥愈发重要。在未来随着汽车向大众进一步普及化，车辆号码将作为车辆的身份识别的重要途径，甚至可以说是唯一的身份识别证明途径，汽车的身份证。如何能够将公路上行驶的汽车在不干扰正常驾驶的前提下识别出车辆号码对于未来社会车辆定位，识别管理对于交通工具的科学管理化具有非常大的意义。目前主流的车牌照识别可以大致分为两个方向，在深度学习、人工智能兴起之前，主要通过手工特征提取完成特征的提取，然后根据提取的特征识别汉字数字文字种类。在深度学习兴起之后，通过深度学习提取图片特征信息的方式流行起来，通过合理设计深度神经网络模型，将数据输入到模型中，由模型完成特征提取和车辆特征识别。目前模型的效果越来越好，已有盖过传统手工特征提取的趋势^[12]。其研究方向和发展路径主要四大方向：

（一）使用 AI 来进一步实际应用进而来增加对于特定交通工具的识别的准确率和速度。

（二）提高在不同摄像头角度的场景下，对车牌照和交通工具的识别速度和准确度，并针对预测过程中的车牌照识别算法所遇到的问题提出新的解决方案，进一步优化算法。

（三）将车牌照识别技术和其他技术进行结合，多角度，多方面来提高车牌照识别的准确度和速度。另外通过将车牌照识别技术和其他技术进行结合，拓展其他的应用场景。

（四）进一步优化车牌识别技术，提高其质量。车辆识别技术将逐步向更智能、更精准的方向发展。如何提高车牌照识别的质量和速度就成为了车牌照定位系统中非常重要一环。目前车牌照识别还持续遇到很多问题，例如不同的角度拍摄到的车牌照对于模型来说难以训练，不同的时间，光照角度对车牌照造成非常大的干扰，影响车牌照的清晰度。不同的模型训练车牌照的效果还存在一定的差异，对于识别准确率和识别质量还有待提高。复杂环境下的正确识别度不高，不同的角度拍摄到的车牌照对于模型来说难以训练，不同的时间，光照角度对车牌照造成非常大的干扰，影响车牌照的清晰度。

不同的模型训练车牌照的效果还存在一定的差异，在本次毕设中，作者针对一些主流的问题提出解决方案，进一步提高了车牌照识别效果。

本毕设主要由两部分组成，车牌照识别的相关算法以及系统的实现，算法的实际部署。其中车牌照识别算法方面主要目标是实现接收汽车公路图片，车牌照算法识别出这张图片中车牌照数字，之后返回。具体步骤为首先读取照片，对车牌照进行一定的预处理，将 BGR 图片转化为 gray 图片，减轻模型的训练量，在进行一定的车牌照预处理之后，需要对车牌照进行进一步的减轻训练量，更有效的保存有效信息，因此对图片进行了二值化，将像素值转换为只有两种可能值，这样可以更容易地识别和分离出图像中的目标物体。之后利用高斯模糊对车牌照的区域和字符区域进行平滑处理，消除噪点，并且使用闭操作减少断点。接着利用 sobel 边缘检测找到车牌照区域^[13]，利用 hsv 颜色空间识别出字符区域^[14]，接着消去车牌照周围的一圈白色边框，通过计算车牌照各字符之间的宽度，利用分割技术得到各个字符的区域，完成字符的分离和简单处理。之后判断得到的字符串的大小，宽度，长度，来确定车牌照号码部分是否是有效区域，利用模型对车牌照的各个字符进行识别，得到各个字符的识别结果。算法部署以及系统设计方面，将车牌照识别算法部署在各个摄像头终端，检测各个摄像头终端中的视频/照片是否存在指定车辆，通过摄像头拍摄的时间和地点，能够快速追踪车辆位置。

接下来，逐步具体分析各部分的细节。

第三章 车牌照识别算法

车牌照识别技术是一种基于计算机视觉和图像处理技术的应用技术，其主要目的是为了实现在智能交通系统中自动识别车辆的车牌照信息的功能。目前在智能交通系统中发挥着越来越重要的技术，利用车牌照识别追踪技术，交通管理部门可以实现车辆追踪、违法监控等功能。车牌照识别的主要流程包括图像采集、预处理、特征提取和字符识别等步骤。其中图像采集部分，利用高速公路摄像头得到中国大多数公路的道路车辆信息。预处理方面，利用 opencv 技术对车牌照进行简单的预处理，以减轻模型的训练量并能更好的从图像中得到车牌照和车牌照字符信息。特征提取方面，主要目的是从预处理后的图像中提取出车牌照的特征信息，如颜色、形状、字符间距等；方法主要分为手工特征提取和基于深度学习的特征提取，本次试验中设计了 svm 模型，mlp 模型，cnn 模型三种模型来进行字符识别。对比和分析了各个算法的优缺点并设计模型，取得了更好的识别效果。之后开始识别车牌照号码结果，利用刚刚训练好的模型对字符区域图像进行识别，分析比对车牌照上的信息得到最终结果。在这个过程中，需要使用一系列的算法和模型，如图像滤波、边缘检测、形态学处理、模板匹配、神经网络等，来实现车牌照的准确识别和特征提取。

第一节 数据集的选择

本次模型训练主要是识别出车牌照中的各个字符，中国车牌照主要由省份汉字（特殊用途的车辆有特殊字符）后跟随一个英文字母一个隔断点，5 个英文数字混合的字符串组成。其中第一个代表省份的省份汉字为各个省区，自治区，直辖市区简称。所以只需要识别 31 个汉字，A-Z 的英文字符和 0-9 的数字字符。其中汉字字符一共 31 个省份简称字符。系统分别收集各个数字的照片以完成数据集，整个训练数据集共有 13072 张照片。照片的详细信息为 20*32 96dpi 8bit。

以下是部分数据集展示：

部分汉字数据集，如图 3-1 所示：



图 3-1 汉字数据集

部分英文数字数据集，如图 3-2 所示：



图 3-2 英文数字数据集

第二节 模型的选择和设计

近些年来车牌照识别取得了非常大的成就，车牌照识别大致可以根据提取特征的种类分成两类，一类是手工提取特征是指通过人工设计算法，从车牌照片中提取出一些特征，如颜色、形状、纹理等，车牌照识别技术可以通过手工提取特征或利用深度学习提取特征进行分类。手工提取特征的优点是算法简单易懂，计算速度快，对于一些简单的场景效果较好。利用深度学习提取特征则是通过神经网络自动学习特征，不需要人工设计算法，可以取得更好的识别效果。但这种方法需要更多的计算资源和数据量支持。但是对于复杂的场景，手工提取特征的效果会受到限制，需要不断调整算法以适应不同的场景。深度学习提取特征的优点是可以自动学习特征，对于一些无法总结的特征能够很好的识别，泛化和适应能力强，对于复杂的场景效果较好。缺点是模型所需数据量大，需要大量计算资源进行训练，且模型的可解释性较差，难以理解模型的决策过程。为了选择最合适的模型，首先分别设计各个模型，通过对模型效果的比较，分析不同的模型对车牌照识别的结果的影响，进而分析出最终结论。

一、传统的手工提取特征

传统的手工提取特征方法是通过人工设计算法，对图像进行处理，提取出图像中的特征，然后利用这些特征进行分类、识别等任务。在车牌识别中，传统的手工提取特征方法可以对车牌进行精确的定位和分割。这种方法需要人工设计特征，这个过程比较繁琐，而且需要专业知识。除此之外，手工提取特征方法对于光照、噪声等环境因素比较敏感，因此在复杂环境下容易出现误识别。利用深度学习提取特征的方法可以一定程度上解决这个问题，但仍需要更多的数据和计算资源支持。另外，传统的手工提取特征方法对于车牌变形、遮挡等情况也比较敏感。

传统的手工提取特征方法包括逻辑回归、朴素贝叶斯、决策树、随机森林、梯度提升树和支持向量机等。但这些方法对于光照、噪声等环境因素比较敏感，在复杂环境下容易出现误识别。逻辑回归假设数据具有线性关系，而很多分类问题并不是线性可分的，不适用于非线性问题。朴素贝叶斯算法需要的训练数据较少，对缺失数据不太敏感，算法简单。决策树算法可以处理多分类

问题，连续和离散型变量，输出结果易于理解，对中间值的缺失不敏感，可以处理不相关特征数据。但是决策树容易过拟合。随机森林是一种集成学习方法，通过随机选择特征和样本来构建多个决策树，并将这些决策树组合起来进行分类或回归。梯度提升树是一种集成学习方法，通过迭代地训练多个决策树来提高模型的准确率。梯度提升树算法的优点是可以处理高维度数据集，并且在处理缺失数据时表现良好。但是梯度提升树算法需要大量时间来训练模型。SVM（Support Vector Machine）是一种监督学习算法，通过拟合数据进而实现分类和回归分析。SVM 可以将数据集分成两个或多个类别，并将新的数据点分配给这些类别之一。SVM 的目标是找到一个最优的超平面，将不同类别的数据点分开，使得两个类别之间的间隔最大化。其本质方法是通过拟合数据集，通过计算损失函数的大小，找到损失函数最小的拟合线，进而 SVM 可以处理线性和非线性分类问题，并且在处理高维数据时表现良好。

以上这些算法都有各自的特点和优缺点，综合分析发现 svm 处理本问题更具有优势。接下来使用 python 具体实现 svm。如图 3-3 所示。

```

MODEL_PATH = "model/svm_enu.m"
TRAIN_DIR = "data/enu_train"
TEST_DIR = "data/enu_test"
IMAGE_WIDTH = 20
IMAGE_HEIGHT = 20
CLASSIFICATION_COUNT = 34          # 数字+英文字符共有34种类别
LABEL_DICT = {
    '0':0, '1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9,
    'A':10, 'B':11, 'C':12, 'D':13, 'E':14, 'F':15, 'G':16, 'H':17, 'J':18, 'K':19,
    'L':20, 'M':21, 'N':22, 'P':23, 'Q':24, 'R':25, 'S':26, 'T':27, 'U':28, 'V':29,
    'W':30, 'X':31, 'Y':32, 'Z':33
}

def load_data(dir_path):
    data = []
    labels = []

    for item in os.listdir(dir_path):
        item_path = os.path.join(dir_path, item)
        if os.path.isdir(item_path):
            for subitem in os.listdir(item_path):
                subitem_path = os.path.join(item_path, subitem)
                gray_image = cv.imread(subitem_path, cv.IMREAD_GRAYSCALE)
                resized_image = cv.resize(gray_image, (IMAGE_WIDTH, IMAGE_HEIGHT))
                data.append(resized_image.ravel())
                labels.append(LABEL_DICT[item])

    return np.array(data), np.array(labels)

def normalize_data(data):
    return (data - data.mean()) / data.max()

def train():
    print("装载训练数据...")
    train_data, train_labels = load_data(TRAIN_DIR)
    normalized_data = normalize_data(train_data)
    print("装载%d条数据, 每条数据%d个特征" % (normalized_data.shape))

    print("训练中...")
    model = svm.SVC(decision_function_shape='ovo')      # ovo代表采用1 vs 1的方式进行多类别分类处理
    model.fit(normalized_data, train_labels)

    print("训练完成, 保存模型...")
    joblib.dump(model, MODEL_PATH)
    print("模型保存到:", MODEL_PATH)

```

图 3-3 svm 模型英文识别部分代码块

首先，将数字和英文字符分别用 0 到 34 的字符代表，以便于拟合数据。

其次完成数据处理部分。通过 python 的 os 模块，首先遍历数据集文件夹，找到数据集图片的位置并读取图片，并将其转化为灰度图像。这便于降低数据处理量，减轻模型负担，通过使用 resize 函数使得图片能够符合模型大小，之后将数据一维化，这便于后续的进行。当完成了基础的数据集处理，得到了模型训练所需要的数据集 data，和数据集标签 labels。

接着对数据进行正则化。将照片的数值从 0-255 变化到 0-1，正则化通过添加正则化项来限制模型的大小，以达到复杂度和性能之间的平衡，防止过拟合。常用的正则化方法包括 L1 和 L2 正则化，均可以被视为损失函数的惩罚项。L1 正则化可以实现稀疏化并节省存储空间，L2 正则化的控制过拟合的效果更好。

之后进行模型训练，利用 SVM 算法中的 SVC（即支持向量分类器）函数来构建模型，在 1 对 1 策略和 one vs rest 策略中选择 1 对 1 多分类问题策略。然后用模型进行训练。最后将训练好的模型保存起来。

在测试数据集上也同样做这样的步骤。

完成训练集和测试机的训练之后，使用 svm 完成汉字部分的识别。首先将汉字字符同样用 0 到 31 代替，这样便于数据集的拟合。之后和上面一样，首先对数据集进行简单的处理，然后利用正则化处理数据，得到数据集数据和数据集标签，然后利用 svm 模型中的 svc 函数来构建模型，利用模型拟合数据，并将训练好的模型保存下来。

模型训练结果：

英文数字模型训练结果：训练集使用 1904 条数据，测试集 816 条数据。预测结果和标签结果相同，预测正确的数目是 801，预测结果不相同，错误是 15 条，综合计算正确率是 0.981618。

汉字模型训练结果：训练集使用 13072 条数据，测试集使用 570 条数据。预测正确是 491 条，预测错误是 79 条，正确率是 0.861404。

预测：利用上述已经训练好的模型进行预测，首先将标签从数字还原到原始的字符。然后对进行预测的图片进行简单的预处理，将图片转化成黑白图片，裁剪图片大小，将裁减好的图片通过正则化，一维化制作成符合要求的照片，加载已经提前训练好的模型，将图片通过加载好的模型进行训练，输出汉字，英文，数字。

二、利用深度学习的手工提取特征

深度学习算法可以通过对车牌照字符图像进行特征提取，识别出车牌照上的字符和数字。主要技术有 cnn，mlp 等。对 mlp，cnn 技术进行进一步的分析和比较。

（一）利用多层感知器（mlp）进行车牌照字符识别，mlp 是一种基于深度学习的图像识别方法。该技术的基本思路是，将车牌照字符图像作为输入，通过多层神经网络进行特征提取和分类，最终输出车牌照的字符信息^[15]。本毕设使用 python 实现 mlp 部分。具体实现代码如图 3-4 所示。接下来简要介绍实现过程。

首先使用 mlp 设计字符识别模型，之后使用 mlp 完成车牌照上汉字的识别，最后完成预测部分。如图 3-4 所示。

```

MODEL_PATH = "model/mlp_enu.m"
TRAIN_DIR = "data/enu_train"
TEST_DIR = "data/enu_test"
IMAGE_WIDTH = 20
IMAGE_HEIGHT = 20
CLASSIFICATION_COUNT = 34
LABEL_DICT = {
    '0':0, '1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9,
    'A':10, 'B':11, 'C':12, 'D':13, 'E':14, 'F':15, 'G':16, 'H':17, 'J':18, 'K':19,
    'L':20, 'M':21, 'N':22, 'P':23, 'Q':24, 'R':25, 'S':26, 'T':27, 'U':28, 'V':29,
    'W':30, 'X':31, 'Y':32, 'Z':33
}

def load_data(dir_path):
    data = []
    labels = []

    for item in os.listdir(dir_path):
        item_path = os.path.join(dir_path, item)
        if os.path.isdir(item_path):
            for subitem in os.listdir(item_path):
                subitem_path = os.path.join(item_path, subitem)
                gray_image = cv.imread(subitem_path, cv.IMREAD_GRAYSCALE)
                resized_image = cv.resize(gray_image, (IMAGE_WIDTH, IMAGE_HEIGHT))
                data.append(resized_image.ravel())#把数据变成一维数组
                labels.append(LABEL_DICT[item])

    return np.array(data), np.array(labels)

def normalize_data(data):
    return (data - data.mean()) / data.max()

def train():
    print("装载训练数据...")
    train_data, train_labels = load_data(TRAIN_DIR)
    normalized_data = normalize_data(train_data)
    print("装载%d条数据, 每条数据%d个特征" % (normalized_data.shape))

    print("训练中...")
    model = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(48, 24), random_state=1)#lbfgs相当于sgd另外一种调参方法
    model.fit(normalized_data, train_labels)

    print("训练完成, 保存模型...")
    joblib.dump(model, MODEL_PATH)#dump 保存模型
    print("模型保存到:", MODEL_PATH)

def test():
    print("装载测试数据...")
    test_data, test_labels = load_data(TEST_DIR)
    normalized_data = normalize_data(test_data)
    print("装载%d条数据, 每条数据%d个特征" % (normalized_data.shape))

    print("装载模型...")
    model = joblib.load(MODEL_PATH)
    print("模型装载完毕, 开始测试...")
    predicts = model.predict(normalized_data)
    errors = np.count_nonzero(predicts - test_labels)#比如我们预测5 那么5-5应该等于0, 当5-X不等于0时, 说明预测值不对, 这样就能计算准确度。
    print("测试完毕, 预测正确: %d 条, 预测错误: %d 条, 正确率: %f" %
          (len(predicts) - errors, errors, (len(predicts)-errors) / len(predicts)))

```

图 3-4 mlp 模型英文识别部分代码块

英文数字识别：同样，这便于后面模型的拟合，首先要完成加载数据和正则化数据这两部分，在加载数据这一块，同样首先利用 python 中文件模块，读取文件夹中的照片的地址，并打开文件夹中所有照片，修改图片尺寸为长 20，宽 20，黑白图片。训练和测试过程中，首先加载图片，然后对图片进行进行正则化，在处理好数据之后，调用 MLPClassifier 函数，设置优化器种类为 lbfgs，学习率的值为 $1e-5$ 次方，神经网络的隐藏层为两层，一层数量为 48 和一层数量为 24，随机数生成器生成的种子是 1。对模型进行训练，并保存。在

测试集上进行和训练集类似的操作。至此就完成了 `mlp` 英文数字部分的训练集和测试集的模型训练。

汉字识别：首先也是要对汉字进行编码，将 31 个汉字按照 0-31 进行编码，以便于进行数据拟合，然后如英文数字识别一样，首先完成数据加载和正则化，然后构建同英文数字识别一样的 `mlp` 模型。接下来，可以使用该模型对数据集进行训练，并使用 `joblib.dump` 函数保存汉字在 `mlp` 上的训练结果。测试集上同理，这样同样完成了 `mlp` 部分的测试结果。

模型的训练结果：

英文数字模型训练结果：训练集数据和上述 `svm` 一样，其中在训练集用了 1904 条数据，测试集部分共有 816 条数据。其中预测的结果正确的为 790，预测结果为错误的数目是 26，正确率是 0.968137。

汉字模型训练结果：训练集使用 13072 条数据，测试集 570 条数据。预测正确是 502 条，预测错误是 68 条，正确率是 0.880702。

预测部分：在完成利用 `mlp` 模型进行英文数字和汉字的模型训练之后，利用模型对照片进行预测，首先将数据集标签由编码转化为原始字符。将需要预测的图片进行简单的处理，首先装载图片，然后将图片转化为适当的尺寸，并做归一化处理。然后加载模型，并利用模型对图片进行预测。

（二）在深度学习中，卷积神经网络（CNN）是一种常用的模型，可以自动学习图像特征。在车牌照识别中，可以使用 CNN 来提取车牌照图像的特征，然后使用分类器来识别车牌照中的字符。具体来说，车牌照识别的流程如下使用 CNN 等深度学习模型对预处理后的图像进行特征提取，得到车牌照图像的特征向量^[16]。本毕设使用 `tensorflow` 实现了该过程。如图 3-5 所示。接下来利用 `cnn` 来完成模型的构建，并探讨 `cnn` 模型的准确度。

```

def normalize_data(data):
    return (data - data.mean()) / data.max()

def onehot_labels(labels):
    onehots = np.zeros((len(labels), CLASSIFICATION_COUNT))
    for i in np.arange(len(labels)):
        onehots[i, labels[i]] = 1
    return onehots

def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):
    # padding='SAME',使卷积输出的尺寸=ceil(输入尺寸/stride),必要时自动padding
    # padding='VALID',不会自动padding,对于输入图像右边和下边多余的元素,直接丢弃
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

x = tf.placeholder(tf.float32, shape=[None, IMAGE_HEIGHT * IMAGE_WIDTH])
y_ = tf.placeholder(tf.float32, shape=[None, CLASSIFICATION_COUNT])
x_image = tf.reshape(x, [-1, IMAGE_HEIGHT, IMAGE_WIDTH, 1])

W_conv1 = weight_variable([5, 5, 1, 32]) # color channel == 1; 32 filters
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # 24x48
h_pool1 = max_pool_2x2(h_conv1) # 24x48 => 12x24

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2) # 12x24
h_pool2 = max_pool_2x2(h_conv2) # 12x24 => 6x12

# 全连接神经网络的第一个隐藏层
# 池化层输出的元素总数为: 12(H)*24(W)*64(filters)
W_fc1 = weight_variable([6 * 12 * 64, 1024]) # 全连接第一个隐藏层神经元1024个
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 6 * 12 * 64]) # 转成1列
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1) # Affine+ReLU

keep_prob = tf.placeholder(tf.float32) # 定义Dropout的比例
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob) # 执行dropout

# 全连接神经网络输出层
W_fc2 = weight_variable([1024, CLASSIFICATION_COUNT]) # 全连接输出为10个
b_fc2 = bias_variable([CLASSIFICATION_COUNT])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

learning_rate = 1e-4
max_epochs = 20
batch_size = 50
check_step = 50

# 使用softmax成本函数
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

```

图 3-5 cnn 模型英文识别部分代码块

和上述一样，将标签进行编码，将上述数字英文和汉字的长度为 34 和 31 的字典进行解码之后，利用 onehot 来对标签进行编码，之后加载数据，制作数据集，对数据进行正则化。接下来来对模型进行构建：

首先来构建 cnn 中的卷积网络的模型组成部分，

权重 W 变量：利用 tensorflow 中的 `tf.truncated_normal` 函数生成一个指定形状的张量，其中的值是从截断正态分布中随机抽取的。`stddev` 参数指定了这个正态分布的标准差，这样可以控制随机化产生的变量在某个范围内。通过这种方式可以使得产生的变量在某个范围内。

偏置量 B ：使用 `tf` 中的 `constant` 变量，定义一个常量张量，其中 `0.1` 是该张量的初始值，`shape` 参数指定了张量的形状。这个张量的值将在整个计算图中保持不变，因此此张量是一个固定值。

`conv2d` 层：使用卷积函数 `tf.nn.conv2d` 对输入张量 x 进行二维卷积操作。这样可以输出一个新的张量，并最终得到输出张量。填充方式 ‘SAME’，在卷积操作中会在输入张量的边缘进行填充，以保证输出张量的大小与输入张量相同。

`pools` 层（池化层）：可以使用 TensorFlow 中的 `tf.nn.max_pool` 函数对输入的张量 x 进行最大池化操作，从而减小张量的尺寸。池化窗口的大小为 `[1,2,2,1]`，其中第一个和最后一个维度为 1，第二个和第三个维度分别为池化窗口的高度和宽度。`strides` 参数指定了池化窗口在输入张量上的滑动步长，定义为 `[1,2,2,1]`，这一个四维的张量，第一个和最后一个维度必须为 1，第二个和第三个维度分别表示在输入张量上的垂直和水平方向上的步长。`padding` 参数指定了边界填充的方式，可以取值为 ‘SAME’ 或 ‘VALID’，分别表示使用零填充或不使用填充。最终的输出张量的尺寸是输入张量尺寸除以池化窗口大小，向下取整。

在构建好基本的权重和卷积神经网络层之后，可以开始构建模型。

首先利用 `tf` 中的 `placeholder` 函数来定义一个占位符，这样便于在执行时赋具体的值。具体来说，这个占位符是一个二维的张量，第一维的大小是 `None`，表示可以接受任意数量的输入数据，每个输入数据的大小为 `IMAGE_HEIGHT * IMAGE_WIDTH`。在实际执行时，可以通过 `feed_dict` 参数将具体的输入数据传递给这个占位符。再来给标签也设置一个占位符，用于在模型训练和推理时接收输入数据，数据类型为 `tf.float32`，形状为 `shape=[None, CLASSIFICATION_COUNT]`，其中 `None` 表示可以接受任意数量的数据，`CLASSIFICATION_COUNT` 表示分类的数量。

构建卷积层并初始化权重和偏置量，指定激活函数为 `relu`，并使用 `conv2d` 函数构建卷积层，之后增加池化层、卷积层和池化层，并设置全连接隐藏层。在此之后，利用 `flat` 层将数据转为一维数据，使用 `dropout` 以防止过拟合，提高模型的泛化能力。最后，可以使用全连接网络输出层，完成模型各层的构建。

这样，就可以得到一个完整的卷积神经网络模型，其中每个输入数据的大小为 $\text{IMAGE_HEIGHT} * \text{IMAGE_WIDTH}$ 。tf.equal()函数用来比较两个张量的元素是否相等，返回一个布尔型的张量。correct_prediction 是一个布尔型的张量，表示模型预测的结果和真实标签是否匹配。最后使用 TensorFlow 框架计算模型的准确率。其中，tf.cast(correct_prediction, tf.float32) 将 correct_prediction 张量转换为 float32 类型的张量，以便进行后续的计算。使用 tf.reduce_mean()函数可以计算张量中所有元素的平均值，即计算正确预测的比例，从而得到模型的准确率。

接下来创建会话，来完成模型的加载和执行，首先初始化所有的变量，然后对数据进行加载，正则化，对标签进行 onehot 编码，得到数据集和数据标签，之后再利用模型对数据集进行训练，并将训练结束的模型保存，便于下一步的预测。至此完成了利用 cnn 模型对英文数字部分的模型的构建和训练。中文部分与此相似，在此不做赘述。

运行结果：英文模型准确度为 0.975，汉字模型准确度为 0.949

当训练好 cnn 模型之后，对其进行预测，并查看结果，首先同样对数据集进行一定的简单的处理，将图片转化为灰白图片，裁剪图片尺寸，正则化数据集，然后将数据集一维化，便于后续训练。构建好数据集和数据标签之后，开始对模型进行定义将图片传入已经构建好的模型当中，并进行预测。模型预测的结果具有不错的表现。

至此完成了所有的 svm, mlp, cnn 模型的构建，经过分析三个模型的效果和准确度。决定英文采用 mlp 英文识别模型，汉字采用 cnn 汉字识别模型

第三节 车牌字符的分割和处理

车牌照字符的分割处理流程大致由以下几个步骤组成：

一、图像预处理。对车牌图像进行预处理，以便后续处理^[17]。本次毕设对车牌照进行预处理的代码如图 3-6 所示。

```
### 步骤1：读取文件，预处理并获取等值线框
# 在包含中文路径时，可采用下列方法读取图像文件
origin_image = cv.imread(np.fromfile(plate_file_path, dtype=np.uint8), -1)
# 对原始图片进行高斯模糊
blured_image = cv.GaussianBlur(origin_image, (5, 5), 0)
# 灰度化和二值化处理
gray_image = cv.cvtColor(blured_image, cv.COLOR_BGR2GRAY)
ret, binary_image = cv.threshold(gray_image, 0, 255, cv.THRESH_OTSU)
```

图 3-6 车牌照图像预处理相关代码块

在本次毕设中，同样对含有车牌照的图像中，对图片进行预处理，首先利用 `np.fromfile` 将读取文件夹中的图片数据，并以 `numpy` 的形式返回，然后利用 `imdecode` 函数将 `numpy` 数组解码成 `opencv` 的 `mat` 格式，在这个过程中自动检测图像类型。之后对读取出来的图片进行高斯模糊，其中参数 `(5, 5)` 表示高斯核大小，`0` 表示标准差。通过高斯模糊，将数据中的微小噪点进行模糊处理，使得图像更加的平滑，减少噪点和细节，从而使得图像更加清晰，同时高斯模糊也可以用来模糊不必要的信息。接着通过 `opencv` 库中的 `cvtColor` 函数，将一个经过模糊处理的彩色图像转换为灰度图像。具体来说，该函数将 `BGR` 格式的彩色图像转换为灰度图像，其中 `BGR` 是指蓝色、绿色和红色通道。这个函数的第一个参数是输入图像，第二个参数是转换的颜色空间，这里是从 `BGR` 到灰度。转换后的灰度图像将被存储在 `gray_image` 变量中。将彩色图像转换为灰度图像可以减少图像的复杂度，使得图像处理更加简单和高效。此外，灰度图像还可以更好地突出图像的纹理和形状特征，方便后续的图像分析和处理。接着使用 `OpenCV` 的库中的 `threshold` 函数来对本次毕设中的每个灰度图像，对其进行二值化处理。其中 `cv` 函数中的 `THRESH_OTSU` 参数表示使用 `OTSU` 算法自动确定二值化阈值。利用二值化处理可以简化对图像处理和分析。二值化后的图像只有黑白两种颜色，可以更加明显地突出图像中的轮廓和特征，便于进行图像识别、分割、检测等操作。同时，二值化还可以减少图像数据量，提高图像处理的效率。

二、车牌定位

通过图像处理技术，将车牌从图像中定位，选择，通常采用的方法有基于颜色的分割、基于形状的分割等。其主要目的是为了定位车牌照位置^[18]。

车牌照识别中的边缘检测算法有很多种，常见的有 `Canny` 算法、`Sobel` 算法、`Laplacian` 算法，等值线算法等^{[19] [20]}。

等值线算法：利用 `python` 和 `opencv` 实现该算法，具体实现过程如图 3-7 所示。

```

import numpy as np
import cv2 as cv

WINDOW_TITLE = "Plage Locate"
plate_file_path = "images/京A82806.jpg"

### 步骤1: 读取文件, 预处理并获取等值线框
# 在包含中文路径时, 可采用下列方法读取图像文件
origin_image = cv.imdecode(np.fromfile(plate_file_path, dtype=np.uint8), -1)
# 对原始图片进行高斯模糊
blurred_image = cv.GaussianBlur(origin_image, (5, 5), 0)
# 灰度化和二值化处理
gray_image = cv.cvtColor(blurred_image, cv.COLOR_BGR2GRAY)
ret, binary_image = cv.threshold(gray_image, 0, 255, cv.THRESH_OTSU)
# 计算等值线矩形
contours, _ = cv.findContours(binary_image, cv.RETR_CCOMP, cv.CHAIN_APPROX_NONE)
# 在原始图片上绘制等值线框
cv.drawContours(origin_image, contours, -1, (0,0,255))
cv.imshow(WINDOW_TITLE, origin_image)
cv.waitKey()

### 步骤2: 对等值线框进行筛选, 得到候选车牌区域
# 检查每个矩形大小以筛选可能的车牌区域
candidate_regions = []
for i in np.arange(len(contours)):
    x, y, w, h = cv.boundingRect(contours[i])
    ratio = w * 1.0 / h          # 计算区域的宽高比
    if ratio < 1:                 # 支持竖排的情形
        ratio = 1.0 / ratio
    area = w * h
    if area > 136 * 36 and area < 136 * 36 * 10 and ratio > 2.0 and ratio < 5.0:
        candidate_regions.append(origin_image[y:y+h, x:x+w])

if len(candidate_regions) == 0:
    print("没有找到候选的车牌区域!")

# 显示候选车牌区域的图像
for i in np.arange(len(candidate_regions)):
    cv.imshow(WINDOW_TITLE, candidate_regions[i])
    cv.waitKey()

```

图 3-7 等值线算法部分代码块

等值线算法是一种常用的图像处理算法, 可以用于查找车牌照区域。该算法基于图像的灰度值, 将图像分成若干个等值线区域, 从而实现对图像的分割和识别。具体步骤如下: 将图像转换为灰度图像。对灰度图像进行平滑处理, 以去除噪声。根据灰度值的分布, 确定等值线的数量和间隔。根据等值线的数量和间隔, 将图像分成若干个等值线区域。对每个等值线区域进行二值化处理, 和形态学处理, 以去除噪声和连接断裂的区域, 以便于后续处理。根据车牌照的特征, 确定车牌照区域。在本次毕设中, 查找二进制图像中的轮廓可以用于对象检测、形状分析、图像识别等任务。通过提取图像中的轮廓, 可以得到物体的边界信息, 从而进行形状匹配、分类、跟踪等操作。此外, 轮廓还可

以用于图像的压缩和编码，以及在图像处理中进行边缘检测和图像分割等操作。轮廓列表被赋值给了 `contours` 变量，而层次结构被赋值给了下划线变量。接下来在原始图像上绘制轮廓线。其中利用 `opencv` 中的 `cv.drawContours()` 函数。在这里，轮廓线的颜色是红色，即 `(0,0,255)`。至此就完成了读取文件，预处理并获取等值线框的部分，接下来需要进行对等值线框的筛选，得到候选车牌的区域。需要检查每个矩形的大小，以筛选可能的车牌区域。

Canny 算法能够显著地检测出车牌照的边缘，具有高准确率和低误检率。能够检测出较细的边缘并抑制噪声，但计算量大且速度慢。该算法适用于需要高精度边缘检测的场景。

能够检测出图像中的弱边缘，并且对于噪声有很好的抵抗能力。因此，**Canny** 算法适合于需要高精度边缘检测的场景，比如图像处理、计算机视觉、机器人视觉等领域。同时，**Canny** 算法也适用于需要对图像进行前期处理的情况，比如图像分割、目标检测等。

Sobel 算法是一种基于梯度的边缘检测算法，能够快速检测出车牌照的边缘，但是对于噪声比较敏感，容易产生误检。**Sobel** 算法是一种常用的边缘检测算法，适合于对图像中边缘信息较为明显的情况进行检测。可以检测出图像中的水平和垂直边缘，并且可以通过组合这些边缘信息来得到更加准确的边缘检测结果。**Sobel** 在图像中存在一定程度的噪声时，也可以得到较为准确的边缘检测结果。但是，对于图像中边缘信息不明显的情况，**Sobel** 算法可能会产生较多的误检测结果。

Laplacian 算法能够检测出车牌照的边缘，但是容易产生较多的噪声和误检。**Laplacian** 算法适合于检测边缘比较明显、变化比较剧烈的图像。可以检测出图像中的高频部分，即边缘部分，但对于噪声比较多的图像效果不佳。此外，**Laplacian** 算法还可以用于图像增强和特征提取等方面。

由于本毕设的识别场景中车牌照是属于在车牌照边缘是比较明显，颜色差异大的情景。**sobel** 算法和定值线算法的特性更符合实际情境，通过和等值线算法的效果比较，最终决定可以采用 **sobel** 边缘检测算法。本毕设同样实现了利用 **sobel** 边缘检测算法实现车牌照定位功能。具体代码如图 3-8 所示。

```

import numpy as np
import cv2 as cv
import util

plate_file_path = "images/plate4.jpg"
plate_image = cv.imread(plate_file_path)

### 将输入图片进行预处理，以供后续的等值线生成
# (1)高斯模糊
blured_image = cv.GaussianBlur(plate_image, (5, 5), 0)
# (2)转成灰度图
gray_image = cv.cvtColor(blured_image, cv.COLOR_BGR2GRAY)
# (3)使用Sobel算子，求出水平方向一阶导数，此时应该可以看到较为明显的车牌区域
grad_x = cv.Sobel(gray_image, cv.CV_16S, 1, 0, ksize=3)
abs_grad_x = cv.convertScaleAbs(grad_x) # 转成CV_8U
# 叠加水平和垂直(本例中没有用到垂直方向)两个方向的梯度结果，形成最终的输出
grad_image = cv.addWeighted(abs_grad_x, 1, 0, 0, 0)
# (4)二值化操作
ret, threshold_image = cv.threshold(grad_image, 0, 255, cv.THRESH_OTSU)
# (5)执行闭操作，使车牌区域连成一个矩形填充的区域
kernel = cv.getStructuringElement(cv.MORPH_RECT, (10, 3))
morphology_image = cv.morphologyEx(threshold_image, cv.MORPH_CLOSE, kernel)

### 执行筛选、矫正和尺寸调整
contours, _ = cv.findContours(morphology_image, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
verified_plates = []
for i in np.arange(len(contours)):
    if util.verify_plate_sizes(contours[i]):
        output_image = util.rotate_plate_image(contours[i], plate_image)
        output_image = util.unify_plate_image(output_image)
        verified_plates.append(output_image)

for i in np.arange(len(verified_plates)):
    cv.imshow("", verified_plates[i])
    cv.waitKey()

cv.destroyAllWindows()

```

图 3-8 sobel 算法部分代码块

sobel 算法边缘检测识别车牌照区域：接着对定位车牌照并且把车牌照提取出来。在将照片进行高斯模糊和灰度图转化后，使用 OpenCV 库中的 Sobel 函数，对灰度图像进行了一阶水平方向的 Sobel 边缘检测。其中，输入的灰度图像是 gray_image，图像深度为 cv.CV_16S 即 16 位有符号整数，算子的大小为 3x3。输出的水平方向的 Sobel 边缘检测结果是 grad_x。之后将梯度图像 grad_x 转换为 CV_8U 格式，并将结果存储在 abs_grad_x 中。CV_8U 是 OpenCV 中的一种 8 位无符号整数格式，用于表示图像像素的灰度值。这个转换可以使梯度图像更容易处理和显示。之后在叠加水平和垂直(本例中没有用到垂直方向)两个方向的梯度结果，形成最终的输出。之后在使用了 OpenCV 库中的 threshold 函数，将梯度图像 grad_image 进行二值化处理^[21]，生成二值化图像

`threshold_image`。接着使用 OTSU 算法自动确定二值化阈值。OTSU 算法计算得到的阈值用函数的返回值用 `ret` 表示。

接着本毕设使用 OpenCV 库进行形态学操作。首先定义了一个结构元素 `kernel`，其结构是一个矩形，大小为(10,3)。接着使用 `cv.morphologyEx()`函数对 `threshold_image` 进行形态学闭运算，使用的结构元素是 `kernel`。闭运算可以用来填补图像中的小孔洞或连接图像中的断裂部分。在进行闭运算时，首先确定合理的结构元素，结构元素的质量会影响闭运算的效果。闭运算可以应用于图像分割、形态学滤波、形态学重构等领域，可以提高图像处理的精度和效果。接着使用形态学操作来找到车牌照图片的轮廓，在将找到的轮廓绘制在车牌图像轮廓上，用红色表示，得到等值线。然后查看一下等值线的效果，挑出处于畸变位置的等值线，并将其框出来。首先将等值线的轮廓赋值给变量 `contour`，本次毕设假设只有一个等值线框。使用 `cv.minAreaRect` 函数找到该等值线的最小外接矩形，返回一个结构体 `rect`，包含矩形中心点坐标、宽度、高度和旋转角度等信息^[22]。

在得到车牌照矩形之后，由于车牌照识别过程中，除了倾斜和扭曲之外，还可能存在模糊、遮挡、光照不足等问题。这些问题都会影响车牌照的识别效果。为了提高识别准确率，可以采用图像增强、车牌矫正、多角度拍摄等技术手段。

之后会指出 4 个角点的坐标，并且画出 4 个角点的坐标，进而得到所对应的外接正交矩形。接着是将一个图像进行放大处理。首先，将原始图像的宽度和高度分别扩大到原来的 1.5 倍，然后创建一个新的图像，大小为扩大后的宽度和高度，并且与原始图像的通道数相同。计算原始图像在新图像中的位置，并将其复制到新图像中心位置。返回新图像中的感兴趣区域（ROI），即原始图像在新图像中的位置。这种方法可以使图像更清晰，更易于观察和分析。同时，通过将原始图像复制到新图像中心的位置，可以保留原始图像的重要信息，避免图像失真或变形。最终返回的感兴趣区域可以用于后续的图像处理和分析。之后将旋转前的图像拷贝到放大图像的中心位置，为了图像完整性，拷贝 `boundingRect` 中的内容。之后计算旋转中心，这里直接将放大图像的中心点作为旋转中心，然后计算旋转所需的变换矩阵，在这里逆时针旋转，然后执行旋转，把照片的角度调整正确。在车牌识别中，预处理是必不可少的步骤。其中包括裁剪和旋转车牌图像，以及针对图像模糊、遮挡、光照不足等问题的处理。对于遮挡问题，可以使用基于纹理合成的图像修复算法，如局部纹理算法。对于光照不足问题，可以使用基于直方图均衡化来增强图像。这些方法可

以提高车牌识别的准确率和效果^[23]。对于多角度拍摄问题，可以使用基于多视角几何的图像拼接算法。这些技术手段可以相互结合，提高图像处理的效果。本毕设针对车牌照识别的实际过程中遇到的问题进行优化，提高算法的鲁棒性效果，具体代码如图 3-9 所示。

```
import numpy as np
import cv2 as cv

# 通过面积和长宽比来判断车牌是否有效
def verify_plate_sizes(contour):
    (center_x, center_y), (w, h), angle = cv.minAreaRect(contour)
    w = int(w)
    h = int(h)
    MIN_ASPECT_RATIO = 2.0;          # 车牌区域轮廓矩形的最小长宽比
    MAX_ASPECT_RATIO = 4.0;          # 车牌区域轮廓矩形的最大长宽比
    MIN_AREA = 34.0 * 8.0 * 10       # 车牌面积的最小值
    MAX_AREA = 34.0 * 8.0 * 100      # 车牌面积的最大值

    # 检查面积
    area = w * h
    if area > MAX_AREA or area < MIN_AREA:
        return False

    aspect = w / h
    if aspect < 1:
        aspect = 1 / aspect

    # 检查长宽比
    if aspect > MAX_ASPECT_RATIO or aspect < MIN_ASPECT_RATIO:
        return False

    return True

# 通过旋转矫正倾斜的车牌区域
def rotate_plate_image(contour, plate_image):
    # 获取该等值线框对应的外接正交矩形(长和宽分别与水平和垂直方向平行)
    x, y, w, h = cv.boundingRect(contour)
    bounding_image = plate_image[y : y + h, x : x + w]

    rect = cv.minAreaRect(contour)      # rect结构: (center (x,y), (width, height), angle of rotation)
    rect_width, rect_height = np.int0(rect[1]) # 转成整数
    angle = np.abs(rect[2])              # 获得畸变角度

    # 如果宽度比高度小,说明矩形相对于最低角点而言,在第二象限;否则相对于最低角点而言在第一象限
    if rect_width < rect_height:
        temp = rect_height
        rect_height = rect_width
        rect_width = temp
        angle = 90 + angle

    if angle <= 5.0 or angle >= 175.0:    # 对于较小的畸变角度,不予处理
        return bounding_image

    # 创建一个放大的图像,以便存放之前图像旋转后的结果
    enlarged_width = w * 3 // 2
    enlarged_height = h * 3 // 2
    enlarged_image = np.zeros((enlarged_height, enlarged_width, plate_image.shape[2]), dtype=plate_image.dtype)
    x_in_enlarged = (enlarged_width - w) // 2
    y_in_enlarged = (enlarged_height - h) // 2
    roi_image = enlarged_image[y_in_enlarged:y_in_enlarged+h, x_in_enlarged:x_in_enlarged+w]
    # 将旋转前的图像拷贝到放大图像的中心位置,注意,为了图像完整性,应拷贝boundingRect中的内容
    cv.addWeighted(roi_image, 0, bounding_image, 1, 0, roi_image)
    # 计算旋转中心.此处直接使用放大图像的中点作为旋转中心
    new_center = (enlarged_width // 2, enlarged_height // 2)
    # 获取执行旋转所需的变换矩阵
    transform_matrix = cv.getRotationMatrix2D(new_center, -angle, 1.0) # 角度为正,表明逆时针旋转
    # 执行旋转
    transformed_image = cv.warpAffine(enlarged_image, transform_matrix, (enlarged_width, enlarged_height))

    # 截取与最初等值线框长、宽相同的部分
    output_image = cv.getRectSubPix(transformed_image, (rect_width, rect_height), new_center)

    return output_image

# 将车牌图片调整成统一尺寸
def unify_plate_image(plate_image):
    PLATE_STD_HEIGHT = 36;          # 车牌区域标准高度
    PLATE_STD_WIDTH = 136;          # 车牌区域标准宽度
    uniformed_image = cv.resize(plate_image, (PLATE_STD_WIDTH, PLATE_STD_HEIGHT))
    return uniformed_image
```

图 3-9 车牌区域矫正, 裁剪, 检测算法部分代码块

在进行车牌识别时, 首先需要验证车牌区域轮廓矩形的长宽比和面积是否符合要求。可以使用 `cv.minAreaRect` 函数获取车牌区域轮廓矩形的中心点坐标、宽度、高度和旋转角度, 并将宽度和高度转换为整数类型。接着, 可以使用基于深度学习的去模糊算法、基于纹理合成的图像修复算法、基于直方图均衡化的图像增强算法和基于边缘检测和透视变换的车牌矫正算法等方法对车牌图像进行预处理, 以提高车牌识别的准确率和效果。接着, 定义了最小长宽比、最大长宽比、最小面积和最大面积的常量。最后, 通过比较车牌区域轮廓矩形的长宽比和面积是否在规定范围内来判断该车牌区域是否合法。如果车牌区域的面积、长宽比不符合要求, 则返回 `False`, 否则返回 `True`。之后再对车牌图像进行旋转矫正, 使其水平方向与竖直方向平行。首先获取车牌图像中车牌的轮廓(contour), 然后根据轮廓获取外接正交矩形的位置和大小(x, y, w, h), 并将该矩形从车牌图像中截取出来(bounding_image)。接着使用 `cv.minAreaRect` 函数获取该矩形的最小外接矩形(rect), 其中 rect 结构为(center (x,y), (width, height), angle of rotation), 即矩形中心点坐标、宽度、高度和旋转角度。根据矩形的宽度和高度以及旋转角度, 判断矩形相对于最低角点而言在第一象限还是第二象限, 并进行相应的处理。最后, 如果畸变角度较小, 则不进行处理, 直接返回截取出来的矩形图像。接着将车牌图片调整成统一的尺寸。首先定义了车牌区域的标准高度和宽度, 然后使用 OpenCV 库中的 `resize` 函数将输入的车牌图片调整为标准尺寸, 最后返回调整后的图片。

一、车牌照字符定位

主要思想是通过 `hsv` 技术将车牌照字符从车牌照区域中定位分离出来^[24]。具体实现代码如图 3-10 所示。


```

import numpy as np
import cv2 as cv
import util

HSV_MIN_BLUE_H = 100                                # HSV中蓝色分量最小范围值
HSV_MAX_BLUE_H = 140                                # HSV中蓝色分量最大范围值
MAX_SV = 255
MIN_SV = 95

def get_candidate_plates(plate_image):
    hsv_image = cv.cvtColor(plate_image, cv.COLOR_BGR2HSV)
    h_split, s_split, v_split = cv.split(hsv_image)      # 将H,S,V分量分别放到三个数组中
    rows, cols = h_split.shape

    binary_image = np.zeros((rows, cols), dtype=np.uint8)

    # 将满足蓝色背景的区域，对应索引的颜色值置为255，其余置为0，从而实现二值化
    for row in np.arange(rows):
        for col in np.arange(cols):
            H = h_split[row, col]
            S = s_split[row, col]
            V = v_split[row, col]
            # 在蓝色值域区间，且满足S和V的一定条件
            if (H >= HSV_MIN_BLUE_H and H <= HSV_MAX_BLUE_H) \
                and (S >= MIN_SV and S <= MAX_SV) \
                and (V >= MIN_SV and V <= MAX_SV):
                binary_image[row, col] = 255

    # 执行闭操作，使相邻区域连成一片
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (10, 3))
    morphology_image = cv.morphologyEx(binary_image, cv.MORPH_CLOSE, kernel)

    contours, _ = cv.findContours(morphology_image, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
    verified_plates = []
    for i in np.arange(len(contours)):
        if util.verify_plate_sizes(contours[i]):
            output_image = util.rotate_plate_image(contours[i], plate_image)
            output_image = util.unify_plate_image(output_image)
            verified_plates.append(output_image)

    return verified_plates

```

图 3-10 hsv 算法部分代码块

接下来对车牌照上的定位字符区域。在进行车牌识别时，可以将车牌图片转换为 HSV 颜色空间，并将 H、S、V 三个分量分别放到三个数组中。然后，可以通过对满足蓝色背景的区域对应索引的颜色值置为 255，其余置为 0 的方式实现二值化。这样可以提高车牌识别的准确率和效果。然后执行闭操作，使相邻区域连成一片。最后，找到符合车牌大小的轮廓，并对车牌进行旋转和统一化处理，最终显示出识别出的车牌图片。首先根据中国车牌照的特性识别并找出车牌照的蓝色背景色值中 HSV 蓝色分量的最小值和最大值的范围大概是 100 到 140，然后首先将车牌照变换为灰白图片，减轻训练量，之后在使用 cv 中的 split 函数对图片的 HSV 颜色空间中的 H，S，V 进行分离，之后进行二值化处理，创建一个和 h 颜色空间大小一样的全零矩阵。之后进行二值化操作，并对其转化为 hsv 空间，以便于下一步将白色的字符从蓝色的车牌照上提取出来。具体实现是通过遍历图像的每个像素点，判断该像素点的 HSV 值是否在蓝

色值域区间内，且满足一定的 S 和 V 条件，如果该像素点的颜色值在某个区间内，则通过形态学的操作改变图像的形态和结构，去除噪声、填补空洞、分离物体等，提高图像处理的准确性和效率。具体步骤为用 OpenCV 库中的函数 `cv.getStructuringElement()` 创建了一个矩形形态学结构元素 `kernel`，大小为 (10,3)。然后使用 `cv.morphologyEx()` 函数对二值化图像 `binary_image` 进行形态学闭运算，使用 `kernel` 作为结构元素。最终得到的 `morphology_image` 是经过形态学闭运算后的图像。这样可以填充图像中的空洞、连接断裂的线条。这样就通过 HSV 得到了各个字符所在的区域。注意：由于中国车牌照外部会有一圈白色的的边框，该边框和中国的车牌照的字符颜色非常接近，通过 `hsv` 不能分离出来，因此在后面还需要对这一部分进行处理。

第四节 字符的分离和处理

由于中国的车牌照的外部边缘存在一个长度为 3，宽度为 2 的边框。在得到车牌照的字符区域之后，需要对车牌照中的字符区域进行分割，首先使用灰度化和二值化来处理该区域，然后去掉车牌照周围的一圈白色字符框。具体实现代码如图 3-11 所示，接下来进行简要分析。

```

import numpy as np
import cv2 as cv

def get_candidate_chars(candidate_plate_image):
    # 灰度化和二值化该区域
    gray_image = cv.cvtColor(candidate_plate_image, cv.COLOR_BGR2GRAY)
    ret, binary_image = cv.threshold(gray_image, 0, 255, cv.THRESH_OTSU)
    # 去掉外部白色边框, 以免查找轮廓时仅找到外框
    offsetX = 3
    offsetY = 5
    offset_region = binary_image[offsetY:-offsetY, offsetX:-offsetX]
    working_region = np.copy(offset_region)

    # 仅将汉字字符所在区域模糊化, 使得左右结构或上下结构的汉字不会被识别成多个不同的轮廓
    chinese_char_max_width = working_region.shape[1] // 8; # 假设汉字最大宽度为整个车牌宽度的1/8
    chinese_char_region = working_region[:, 0:chinese_char_max_width]
    cv.GaussianBlur(chinese_char_region, (9, 9), 0, dst=chinese_char_region) # 采用In-Place平滑处理

    # 在工作区中查找轮廓
    char_contours, _ = cv.findContours(working_region, cv.RETR_CCOMP, cv.CHAIN_APPROX_NONE)
    # cv.drawContours(candidate_plate_image, char_contours, -1, (0, 0, 255))
    # cv.imshow("", candidate_plate_image)
    # cv.waitKey()
    # 以下假设字符的最小宽度是车牌宽度的1/40 (考虑到I这样的字符), 高度是80%
    min_width = working_region.shape[1] // 40
    min_height = working_region.shape[0] * 7 // 10
    valid_char_regions = []
    for i in np.arange(len(char_contours)):
        x, y, w, h = cv.boundingRect(char_contours[i])
        # 字符高度和宽度必须满足条件
        if h >= min_height and w >= min_width:
            # 这里采用offset_region而不是working_region, 是因为:
            # (1)二者区块相同
            # (2)working_region被上面汉字模糊化处理过, 会导致提取的汉字区域模糊
            valid_char_regions.append((x, offset_region[y:y+h, x:x+w]))
    # 按照轮廓的x坐标从左到右排序
    sorted_regions = sorted(valid_char_regions, key=lambda region: region[0])

    # 包装到一个list中返回
    candidate_char_images = []
    for i in np.arange(len(sorted_regions)):
        candidate_char_images.append(sorted_regions[i][1])

    return candidate_char_images

```

图 3-11 字符串分离和选取部分代码块

中国车牌照的边框左右宽度为 3, 上下宽度为 5, 裁剪通过 hsv 得到的字符穿区域, 得到去掉左右上下边框的车牌照区域。之后对汉字字符所在区域进行模糊化, 使得左右上下结构, 中间有断接, 非连体的汉字不会被识别成两个不同的汉字。具体办法为首先计算中为字符的最大宽度, 中国车牌照中一共有 7 个字符和一个中心隔断点, 大多数字符的大小相同, 和中心隔断点一起各占用八分之一的宽度, 将工作区域的宽度除以 8 得到结果。但是也有一些字符例如 I 的比例会小一点, 需要对其进行调整。对于不同的字符, 最小宽度也会大于四十分之一, 高度是百分之 80。取出各个字符区域, 具体办法是从一个二维数组中取出指定列的数据, 取出所有行的第 0 列到第 chinese_char_max_width-1 列的数据, 赋值给变量 chinese_char_region。然后使用 OpenCV 库中的 GaussianBlur 函数对一个中文字符区域进行高斯模糊处理, 其中高斯核的大小为 9x9, 高斯核的标准差为 0, 输出的图像矩阵。然后开始在 opencv 中查找二

进制图片的大致轮廓。类似的短宽度字符，需要特别考虑一下，首先计算有效字符区域的最小宽度和最小高度，并将其存储在 `valid_char_regions` 列表中。将 `working_region` 的宽度除以 40，最小高度的值为区域的高度的 70%，接下来对字符轮廓进行遍历，获取每个字符的位置和大小信息，并筛选出符合要求的字符区域，将其加入到有效字符区域列表中。其中，对字符高度和宽度的最小值进行限制，指定是原始图像中的区域。

然后按照轮廓的 X 坐标，对识别出来的各个字符串数字进行排序，这样就得到了排序好的车牌照字符。

第五节 预测和识别

当完成了车牌照的区域定位，字符的分离和处理，模型的训练，接下来则利用已训练的模型对牌照字符区域图片进行逐个识别，将识别的结果整合作为最终结果。

以下做简单展示（如图 3-12 所示）：

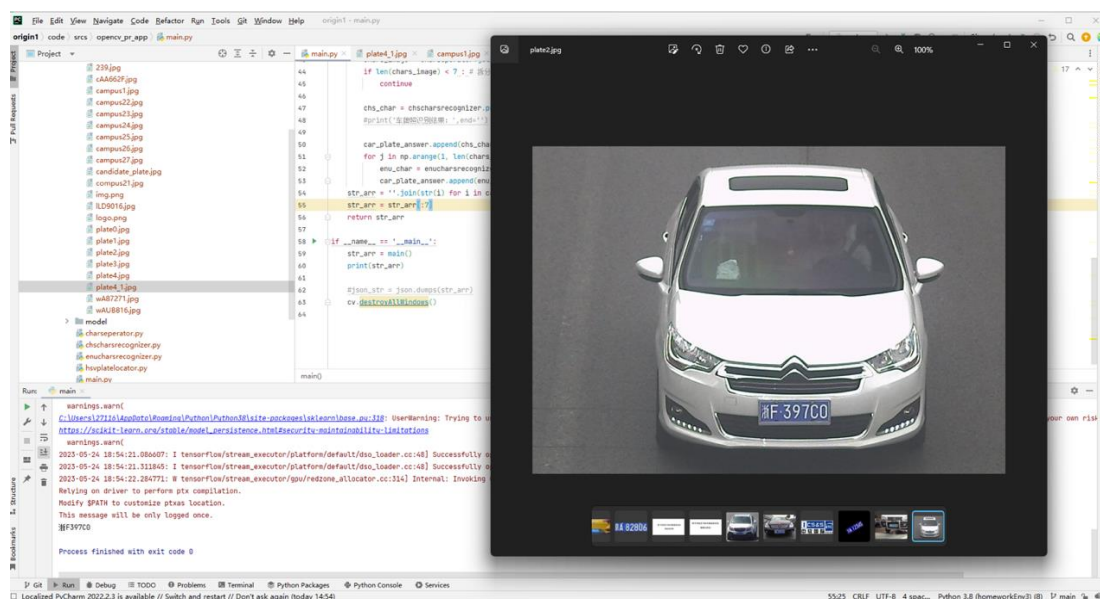


图 3-12 模型识别效果

第四章 追踪系统的具体设计与实现

本毕设毕业设计拟部署在各摄像头的终端中，首先通过收集公路摄像头的视频，算法实时分析各个路段的摄像头采集的视频结果，并与待查询车辆进行比对，如果比对结果成功，则显示摄像头所记录的时间和路段信息。对此，由于中国道路摄像头采用的是公安网，并不对外开放调用接口，且除了部分河南高速路段等地的摄像头，绝大多数摄像头并不对外开放，处于这样的情况，在这里简单模拟展示。

首先，本毕设算法应该部署在各个终端，获取所有的摄像头上的实时画面，之后利用算法，每隔一秒从摄像头中抽取实时画面，分析是否含有指定车牌照的图像。记录出现在摄像头中的时间和摄像头地点实时返回。当需要追踪某一车辆，首先上传待追踪车辆的图片，获取待追踪车辆的车牌照信息，之后通过算法与在摄像头中获取的信息进行比对，分析出车辆轨迹。具体操作流程如下：

网页端界面（如图 4-1 所示）：



图 4-1 网页展示界面

在网页端界面，用户在左侧上传待检测车辆图片，通过车牌照检测算法识别出车牌照号码，以待比对。下面逐步分析各个部分的具体细节

检测车辆部分（如图 4-2 所示）：



图 4-2 待检测车辆上传识别功能展示图

点击选择图片，用户可以上传车辆图片，当用户上传含有待检测车辆的车牌照图片后，算法会自动执行，并在下方显示车牌照信息。

摄像头机位部分（如图 4-3 所示）：




图 4-3 各地点摄像头机位上传视频

基于深度学习的车辆追踪平台



京A5J512


在2023-5-24 18:03:11在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:12在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:13在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:14在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:15在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:16在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:17在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:18在1处发现车牌为京A5J512的车辆
 在2023-5-24 18:03:21处未发现



京津高速北出口


在2023-5-24 18:03:17处未发现
 在2023-5-24 18:03:18处未发现
 在2023-5-24 18:03:19处未发现
 在2023-5-24 18:03:20处未发现
 在2023-5-24 18:03:21处未发现
 在2023-5-24 18:03:22处未发现
 在2023-5-24 18:03:28处未发现
 在2023-5-24 18:03:29处未发现
 在2023-5-24 18:03:30处未发现

在2023-5-24 18:03:18 - 2023-5-24 18:03:21内，在京津高速北出口处发现车牌为京A5J512的车辆
 暂未检测到固定车辆
 暂未检测到游荡车辆



京津高速北出口

在2023-5-24 18:03:21处未发现
 在2023-5-24 18:03:22处未发现
 在2023-5-24 18:03:23处未发现
 在2023-5-24 18:03:24处未发现
 在2023-5-24 18:03:27处未发现
 在2023-5-24 18:03:29处未发现



京津高速北湾隆出口

在最左侧，用户可以上传待检测车辆的车牌照，在右侧通过将视频部署在终端上，实时获取各个地点的摄像头信息。通过车牌照识别算法，能实时的选取车牌照与视频中的图像进行比对，如果发现有待检测车辆经过，即车牌照识别结果和视频中每隔一秒抽取的视频的识别结果相同，就显示该记录，最后就能知道车辆从出现到经过的时间，若不相同同样显示不相同的记录，同样能够知道车辆在此时间段内没有经过该地方。最后总结识别结果。得出车辆的经过和未经过的地方。了解车辆在何时经过何地的位置信息，完成追踪。

第五章 展望

第一节 系统未来发展分析

本毕设实现的算法在实际应用的时候仍然有很多缺点，中国的大多数路段的摄像质量并不高，且大多数距离地面高度超过摄像头清晰拍摄条件，外加部分外界因素，例如树枝遮挡，雨水渗透镜头玻璃盖，阳光直射等等问题，仅仅通过摄像头进行车辆的追踪远远不能满足需求，为了更好的辅助基于车牌照的车辆追踪系统，结合上述关于当今的车辆追踪的相关分析，笔者认为理想的办法是加入基于车辆特征的车辆追踪系统的开发。本章节首先对基于车辆特征的车辆追踪的实现的进一步探讨，接着分析算法存在的问题和当今社会的需要，提出未来的进一步展望。

第二节 优化系统方法畅想

基于车辆特征的研究，进而做到车辆的实时定位和识别。基于车辆特征的追踪系统是一种利用车辆的整体所有特征信息进行整体匹配，分析进而实现追踪的技术。该系统通过对车辆的车型、颜色，形状等整体特征进行识别和记录，实现对车辆的实时追踪和监控。

实现基于车辆特征的车辆追踪系统的工作和设计流程和机遇车牌照的车辆追踪系统大致相同，主要区别在于，如果通过提取车辆特征与含有原车型的车辆特征进行比对，进而判断视频中是否含有该车辆。接下来对这一技术进行详细介绍。

第三节 优化算法的详细分析

第一步：使用 yolo 目标检测算法^[25]识别出图像中的汽车区域，并将其裁剪出来。如图 5-1 所示。



图 5-1 yolo 目标检测模型

通过 yolov5 算法，能够很快速的将一段视频中的车辆实时识别定位并且裁剪出来，得到包含车辆的图片。

第二步：对图片中的汽车区域图片进行比对和识别，判断是否与待检测车辆是同一车辆。该问题可以转化为图片相似度计算问题：

图像相似度计算是指对两张或多张图像进行比较，以确定多张图片之间的相似度或差异度。常用的图像相似度计算技术包括对比度、互相关、SSIM、PSNR 等。这些技术对于旋转、缩放、平移等变换具有较好的鲁棒性，对于光照、噪声等干扰具有一定的容错能力。但是这些技术同样具有缺点，需要提取特征点，计算量较大，对于图像中特征点分布不均匀的情况效果不佳[26]。

基于像素的相似度计算：该方法通过比较两张图像中每个像素的颜色值来计算图片之间的相似度。这种方法简单易懂，但对于颜色、亮度、对比度等变化较大的图像效果不佳。对于旋转、缩放、平移等变换不具有鲁棒性。

基于特征的相似度计算：该方法通过提取图像中的特征点或特征描述符，如 SIFT、SURF、ORB 等[27] [28]来计算图片之间的相似度。这种方法对于图像的旋转、缩放、平移等干扰具有较好的鲁棒性，同时对于光照、噪声等干扰具有一定的容错能力。缺点：需要提取特征点，计算量较大，对于图像中特征点分布不均匀的情况效果不佳。

基于深度学习的相似度计算方法利用深度学习模型，对图像进行相似度计算。这种方法在大规模数据集上具有较好的效果，同时可以自动学习特征，不需要手动提取特征点。但是需要大量的计算资源和训练数据。缺点：需要大量的训练数据和计算资源，对于小样本数据集效果不佳，且模型可解释性较差。

由于的车辆特征具体情况不清晰，且模型需要很好的泛化性，因此在这里使用基于深度学习的相似度计算。在这使用 2-channels 模型进行车辆相似度计算。

Zhou 等人 2016 年在 IEEE 上发表论文并提出了新的方法来计算图片相似度。其主要思想是利用 2-channels 模型来进行图片相似度比对。2-channels 模型计算图片相似度的原理是基于两个通道的特征提取和比较。其中一个通道是待检测图片，另一个通道是匹配图像。通过比较两个图片在这两个通道中的特征，模型可以计算出图片的相似度。具体实现上，模型会使用卷积神经网络来提取图片的特征，并使用余弦相似度来计算图片的相似度。本质上为将两个单通道的图片合并成一个双通道的图片，并直接使用 CNN 对这两个通道的图片进行训练。然后计算图片的相似度[29]。传统的图像比较方法是先提取图像的特征值，然后创建一个函数对特征值进行分析，最后得出两幅图像是否相似。但是 2-channels 模型没有单独提取特征值的过程，只是将两幅图片组成一幅双通道图片，直接输入到神经网络中进行图像相似度的训练和计算。这种方式在提高模

型泛化性，特别是从总体上提取特征，尤其是一些难以人工提取的特征方面，具有得天独厚的优势，并且模型在很多数据集上也有非常好的效果。

一、构建数据集

首先需要对含有不同车型的数据集照片进行预处理，通过 `opencv` 和 `image` 库，将数据集中的大小统一尺寸，这有利于下一步的模型训练过程。之后利用收集到的车辆照片来构建的车牌照数据集。具体实现相关代码如图 5-2 所示。

```
# Load the mnist dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Resize the image
x_trains = []
x_tests = []
for i in range(len(x_train)):
    crop_size = (720, 720)
    x_train_1 = x_train[i]
    img_new1 = cv2.resize(x_train_1, crop_size, interpolation = cv2.INTER_CUBIC)
    x_trains.append(img_new1)
for i in range(len(x_test)):
    crop_size = (720, 720)
    x_test_1 = x_test[i]
    img_new2 = cv2.resize(x_test_1, crop_size, interpolation=cv2.INTER_CUBIC)
    x_tests.append(img_new2)

#creat dataset
dataset_numbers0 = [] #y_train0 5923
...

# Classify the dataset according to the labels
for i in range(len(y_train)):
    if y_train[i] == 0:
        dataset_numbers0.append(x_trains[i])
    ...

# Build similar/dissimilar datasets (label 1 and label 0)
dataset_len = 1000
image_sample_true_channel1 = []
image_sample_true_channel2 = []

image_sample_false_channel1 = []
image_sample_false_channel2 = []

for i in range(dataset_len):
    temp_label = random.randint(0, 9)
    temp_dataset_numbers = eval('dataset_numbers'+str(temp_label))

    for j in range(2):
        temp_number = random.randint(0, len(temp_dataset_numbers)-1)# randint (a,b) ranges from [a,b],So minus 1 is
        really necessary
        if j == 0:
            image_sample_true_channel1.append(temp_dataset_numbers[temp_number])
        if j == 1:
            image_sample_true_channel2.append(temp_dataset_numbers[temp_number])

for i in range(dataset_len):
    temp_labels = random.sample(range(0, 10), 2)
    temp_dataset_numbers_channel1 = eval('dataset_numbers'+str(temp_labels[0]))
    temp_dataset_numbers_channel2 = eval('dataset_numbers'+str(temp_labels[1]))

    temp_number_channel1 = random.randint(0, len(temp_dataset_numbers_channel1)-1)
    temp_number_channel2 = random.randint(0, len(temp_dataset_numbers_channel2)-1)#randint(a,b) yield [a,b]
    image_sample_false_channel1.append(temp_dataset_numbers_channel1[temp_number_channel1])
    image_sample_false_channel2.append(temp_dataset_numbers_channel2[temp_number_channel2])

for i in range(dataset_len):
    cv2.imwrite("./img/train/true/" + str(i) + ".jpg",
                cv2.merge([image_sample_true_channel1[i], image_sample_true_channel2[i], np.zeros([32, 32],
dtype="uint8"])))
for i in range(dataset_len):
    cv2.imwrite("./img/train/false/" + str(i) + ".jpg",
                cv2.merge([image_sample_false_channel1[i], image_sample_false_channel2[i], np.zeros([32, 32],
dtype="uint8"])))
```

图 5-2 图片的预处理相关代码

构建标签为 `same` 和 `difference` 的数据集：如果两张照片所包含的车辆是同一个车辆，认为该组合为 `same`，相反如果两张照片所包含的车辆不是同一个车

辆，则认为该组合为 **difference**。随机抽取某个标签，从该标签中随机抽取两张照片，认为这两张照片中的车辆来自于同一个标签，因此可以选择这两张图片的组合作为数据集中的 **same** 标签。反之，从不同的标签中各随机抽取一张照片，该两张照片的组合认之为数据集中标签为 **difference**。这样就构建好了的数据集。在这个过程中，由于数据集的大小不一定相同，数据集中每个数字的图像数量也不相同。最小的车辆数据集可能只有几十张，最大的车辆数据集照片可能有几百张。如果数据集规模远远大于平均数据集大小和最大数据集大小，那么数据集中可能会有很多相似的图片，这可能会影响神经网络的数据集训练过程。因此，本数据集大小限制为最大数据集的数量。此外，由于 **OpenCV** 的 **merge** 函数只支持合成三个通道，不能两个单通道图片组合成一个双通道照片，因此这里将数据集合并为一个三通道照片，其中一个通道由相同大小的单通道照片组成，所有照片的值都为 0。测试集做相同的操作。具体实现相关代码如图 5-3 所示。

```

# Build similar/dissimilar datasets (label 1 and label 0)
test_image_sample_true_channel1 = []
test_image_sample_true_channel2 = []

test_image_sample_false_channel1 = []
test_image_sample_false_channel2 = []
test_dataset_len = 1000
for i in range(test_dataset_len):
    temp_label = random.randint(0,9)
    temp_dataset_numbers = eval('test_dataset_numbers'+str(temp_label))

    for j in range(2):
        temp_number = random.randint(0, len(temp_dataset_numbers)-1)
        if j == 0:
            test_image_sample_true_channel1.append(temp_dataset_numbers[temp_number])
        if j == 1:
            test_image_sample_true_channel2.append(temp_dataset_numbers[temp_number])

for i in range(test_dataset_len):
    temp_labels = random.sample(range(0, 9), 2)
    temp_dataset_numbers_channel1 = eval('test_dataset_numbers'+str(temp_labels[0]))
    temp_dataset_numbers_channel2 = eval('test_dataset_numbers'+str(temp_labels[1]))

    test_temp_number_channel1 = random.randint(0, len(temp_dataset_numbers_channel1)-1)
    test_temp_number_channel2 = random.randint(0, len(temp_dataset_numbers_channel2)-1)
    test_image_sample_false_channel1.append(temp_dataset_numbers_channel1[test_temp_number_channel1])
    test_image_sample_false_channel2.append(temp_dataset_numbers_channel2[test_temp_number_channel2])

for i in range(test_dataset_len):
    cv2.imwrite("./img/test/true/" + str(i) + ".jpg",
                cv2.merge([test_image_sample_true_channel1[i], test_image_sample_true_channel2[i], np.zeros([32, 32],
dtype="uint8"])))
for i in range(test_dataset_len):
    cv2.imwrite("./img/test/false/" + str(i) + ".jpg",
                cv2.merge([test_image_sample_false_channel1[i], test_image_sample_false_channel2[i], np.zeros([32,
32], dtype="uint8"])))

# Loading the dataset

trainSample = []
trainLabel = []
testSample = []
testLabel = []

path = 'img/'
def loadSamples():
    sampleTrainPath1 = os.listdir('img/train/true/')
    sampleTrainPath2 = os.listdir('img/train/false/')
    for file in sampleTrainPath1:
        img = Image.open('./img/train/true/'+file)
        img = tf.keras.preprocessing.image.img_to_array(img)
        trainSample.append(img)
        trainLabel.append(1)
    for file in sampleTrainPath2:
        img = Image.open('./img/train/false/'+file)
        img = tf.keras.preprocessing.image.img_to_array(img)
        trainSample.append(img)
        trainLabel.append(0)
    print("The training dataset is loaded")
    sampleTestPath1 = os.listdir('img/test/true/')
    sampleTestPath2 = os.listdir('img/test/false/')
    for file in sampleTestPath1:
        img = Image.open('./img/test/true/' + file)
        img = tf.keras.preprocessing.image.img_to_array(img)
        testSample.append(img)
        testLabel.append(1)
    for file in sampleTestPath2:
        img = Image.open('./img/test/false/'+file)
        img = tf.keras.preprocessing.image.img_to_array(img)
        testSample.append(img)
        testLabel.append(0)
    print('The code dataset is loaded')
    return trainSample, trainLabel, testSample, testLabel
trainSample, trainLabel, testSample, testLabel = loadSamples()

```

图 5-3 数据集的构造相关代码

二、模型的构建和调整

在 2-channels 模型的论文中作者并没有给出具体的模型搭建建议，而是给出了模型的层序推荐，其他科研工作者对此进行进一步的探讨，范大昭、董杨、张永生等人对该模型进行进一步研究，本毕设参考先前研究工作者的研究

成果并结合显示情况的实际需要，确定好了的模型框架。具体框架如图 5-4 所示。

```

"""
Model: "sequential"

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 15, 15, 96)	4704
conv2d_1 (Conv2D)	(None, 15, 15, 96)	83040
conv2d_2 (Conv2D)	(None, 15, 15, 96)	83040
conv2d_3 (Conv2D)	(None, 15, 15, 96)	83040
max_pooling2d (MaxPooling2D)	(None, 8, 8, 96)	0
conv2d_4 (Conv2D)	(None, 6, 6, 192)	166080
conv2d_5 (Conv2D)	(None, 4, 4, 192)	331968
conv2d_6 (Conv2D)	(None, 2, 2, 192)	331968
flatten (Flatten)	(None, 768)	0
dense (Dense)	(None, 768)	590592
dense_1 (Dense)	(None, 2)	1538

```

=====
Total params: 1,675,970
Trainable params: 1,675,970
Non-trainable params: 0
=====
"""

```

图 5-4 模型结构

三、训练模型：

在构建好模型之后。首先导入了一些必要的库和数据集，然后定义了学习率调度函数，对数据进行打乱。接着打印训练集和测试集的形状，对数据进行了预处理，包括将像素值归一化和将数据 reshape 成合适的形状。最后使用 createModel 函数创建模型，对模型进行训练，绘制模型训练结果图^[30]。

四、模型效果展示

在经过 epoch 为 20，batchsize 为 10 的训练之后，模型结果如图 5-5 所示。

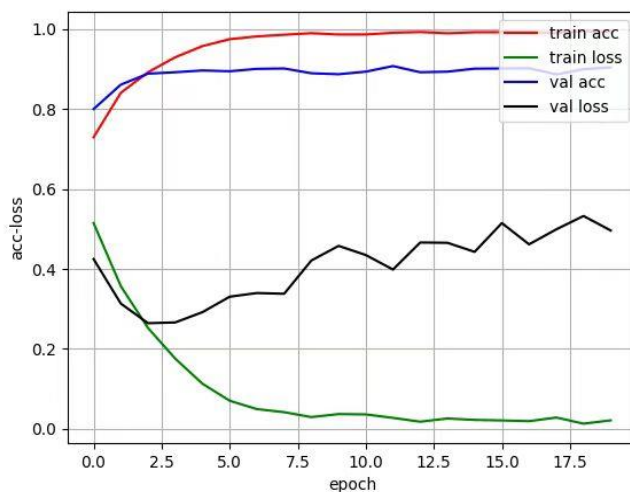


图 5-5 模型训练结果

五、预测效果

在 cifar10 数据集中的前 500 张图片中，随机抽取几张图片，让模型预测与之相似的几张图片，结果如图 5-4 所示（左边是随机抽取的图片，右边是模型，下标分别对应其在 cifar10 数据集中的位置）。



图 5-4 模型预测实际效果

第四节 算法优化方式总结

由于汽车图片具有非常多的特征，简单的模型并不能很好的提取图片特征，容易出现图片效果不好的情况，在 `mnist` 数据集上，该模型尚还具备良好性能，但是在中国汽车品牌数据集上，由于模型的深度不足以提取足够的车辆图像中特征来帮助判断是否是相同车辆，暂时该问题还存在欠缺。未来希望算力进一步提升，模型的进一步优化，可以取得理想的效果。

结 论

本次毕业设计实现了基于车牌照识别的车辆追踪系统的设计和实现。其主要分为两部分

一、基于 opencv 的车牌照的预处理，定位，识别和字符的分离

使用 opencv 对车牌照进行预处理，使用高斯模糊，图片灰度化，二值化等技术，减少模型训练量，去除噪点，提高图片的平滑度，以便于后续操作。接着使用边缘检测技术查找车牌照区域，这里使用了等值线技术，sobel 算法等技术查找可能的车牌照区域，在这个过程中，很有可能会出现车牌照断裂，车牌照方向有所旋转等等情况，为了提高算法的泛用性，增强算法在实际环境中的鲁棒性，本毕设对于车牌照断裂问题使用了闭操作来连接车牌照，对于车牌照扭曲，旋转问题，利用 opencv 设计车牌照旋转函数。之后判断所选择的车牌照是不是符合要求通过检查车牌照的大小区域和长宽比例。挑选出合适的车牌照区域。之后利用 hsv 来分析车牌照字符，中国的轿车车牌照是纯白色字体和纯蓝色背景和纯白色边框，所以可以利用 hsv 提取车牌照区域，并经过裁剪去除边框。之后根据每个字符的宽度，裁剪去出一个字符。这个过程中需要注意部分上下左右结构汉字容易被错误识别。为了避免这种错误，需要对汉字部分采用模糊处理。之后根据字符的横坐标进行排序，得到车牌照字符的区域图片数组。

二、基于深度学习的车牌照识别模型的构建和训练

按照特征提取的种类可以分为传统的手工提取特征，利用基于深度学习提取特征，设计了 SVM，MLP，CNN 三种模型，并比较他们在中文和英文数字模型上的训练结果，并最终决定采用。该算法在大多数条件下可以很好的效果，准确率到达百分之 90 以上。

拟将算法部署在中国各个高速公路的摄像头终端上，通过实时采集各个摄像头的视频和照片，通过算法识别比对是否有某张照片，然后输出结果，显示车辆途径区域。结果显示该算法识别质量尚可。

参考文献

- [1]余淑秀, 卢山冰, 沈锐. 中国乘用车市场需求及兼并重组福利效应模拟[J]. 系统工程, 2018, 36(12):8.
- [2]周洪伟. 如何利用 3G4G 无线网及 GPS 定位技术实现通信工程现场监理[J]. 通讯世界, 2017(7):1.
- [3]杨华胡振陈水容唐天国王丽汤永斌. 基于 Android 的北斗导航技术和地图技术实现车辆位置定位研究[J]. 数码设计 (上), 2021, 000(021):81-87.
- [4]甘正男. 基于差分定位的 LBS 服务在智能网联汽车测试评价中的应用及优化[J]. 智能物联技术, 2021, 53(2):48-53.
- [5]薛涛. 计算机视觉技术在无人驾驶汽车中的应用[J]. 汽车测试报告, (15):67-69.
- [6]Betke, Margrit, Esin Haritaoglu, and Larry S. Davis. "Real-time multiple vehicle detection and tracking from a moving vehicle." *Machine vision and applications* 12 (2000): 69-83.
- [7]Babenko, Boris, Ming-Hsuan Yang, and Serge Belongie. "Visual tracking with online multiple instance learning." *2009 IEEE Conference on computer vision and Pattern Recognition*. IEEE, 2009.
- [8]Gil, Sylvia, Ruggero Milanese, and Thierry Pun. "Combining multiple motion estimates for vehicle tracking." *Computer Vision—ECCV'96: 4th European Conference on Computer Vision Cambridge, UK, April 15–18, 1996 Proceedings Volume II* 4. Springer Berlin Heidelberg, 1996.
- [9]Zhu, Li, et al. "Big data analytics in intelligent transportation systems: A survey." *IEEE Transactions on Intelligent Transportation Systems* 20.1 (2018): 383-398.
- [10]Thaiparnit, Sattarpoom, et al. "Tracking vehicles system based on license plate recognition." *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*. IEEE, 2018.
- [11]班利, 李冰, 李大鹏. 汽车牌照识别器在高速公路的应用[J]. 科学与财富, 2019.
- [12]杨春德, 郭帅. 改进基于 HSV 空间的阴影检测算法[J]. 计算机工程与设计, 2018, 39(1):5.
- [13]罗志勇, 郭晓惠, 宦红伦, 等. 一种基于 Sobel 算子的图像边缘检测方法:, CN108242060A[P]. 2018.
- [14]赵洋, 闵升锋, 李大舟. 基于 HSV 空间颜色和纹理特征的车牌定位算法研究[J]. 沈阳化工大学学报, 2022(036-001).
- [15]路明玉. 通过 MLP 网络结构进行手写数字识别的系统设计[J]. 科技资讯, 2019, 17(14):2.

- [16]欧先锋[1,2], 向灿群[1,2], 湛西羊[3],等. 基于 CNN 的车牌数字字符识别算法[J]. 成都工业学院学报, 2016, 19(4):6.
- [17]黄岳锐, 黄楷佳. 基于图像处理的车牌识别与字符分割及 MATLAB 实现[J]. 内蒙古科技与经济, 2019(10):3.
- [18]李培灵, 张潇, 王锋. 一种车牌定位与车牌字符分割方法:, CN110619335A[P]. 2019.
- [19]张月圆, 曾庆化, 刘建业,等. 基于 Canny 的改进图像边缘检测算法[J]. 导航与控制, 2019(1):7.
- [20]雷得超, 任守华. 基于 OpenCV 图像处理车牌识别系统分析研究[J]. 电脑与信息技术, 2022, 30(4):3.
- [21]江进. 基于灰度化及边缘检测算法的车牌识别技术研究[J]. 现代计算机:上下旬, 2014.
- [22]刘卓帆, 袁锐. 基于 OpenCV 算法实现图像处理软件"焊点截取"[J]. 电脑知识与技术, 2022(023):018.
- [23]姜柏军, 钟明霞. 改进的直方图均衡化算法在图像增强中的应用[J]. 激光与红外, 2014, 44(6).
- [24]贺智龙, 肖中俊, 严志国. 基于 HSV 与边缘信息的车牌定位与字符分割方法[J]. 山东轻工业学院学报 (自然科学版), 2019(003).
- [25]Redmon J , Divvala S , Girshick R , et al. You Only Look Once: Unified, Real-Time Object Detection[C]// Computer Vision & Pattern Recognition. IEEE, 2016.
- [26]Zhang R , Isola P , Efros A A , et al. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric[J]. IEEE, 2018.
- [27]Luo J , Oubong G . A Comparison of SIFT, PCA-SIFT and SURF[J]. International Journal of Image Processing, 2009.
- [28]Grabner M , Grabner H , Bischof H . Fast Approximated SIFT[M]. Springer Berlin Heidelberg, 2006.
- [29]Zagoruyko S , Komodakis N . Learning to Compare Image Patches via Convolutional Neural Networks[J]. IEEE, 2015.
- [30]范大昭董杨张永生. 卫星影像匹配的深度卷积神经网络方法[J]. 测绘学报, 2018, 047(006).

致 谢

本科顺利毕业后，下学期我将要前往日本读研了，感谢学校开设了中日专业给我不一样的生活方式。感谢各位老师大学对我的关照，说句掏心窝子的话，我在大学接受到的老师关爱比很多同学都多得多。在申请日本研究生的过程中，感谢人工智能学院的赵老师，任老师，梁老师，武老师，李老师，邱老师，刘老师等等很多老师都尽最大可能给了我很大帮助。

我一直坚持走一步算一步，重于体验，不太喜欢牺牲现在去为了未来，牺牲未来为了更未来。从南方来到北方，我很庆幸认识一帮有趣的人，人生是一场旅行，这四年的北方之行，还不错！

撒油啦啦！（日语音译：再见）