车牌自动识别软件

成员：李旋、方陈、朱瑞祥

指导老师：刘晋

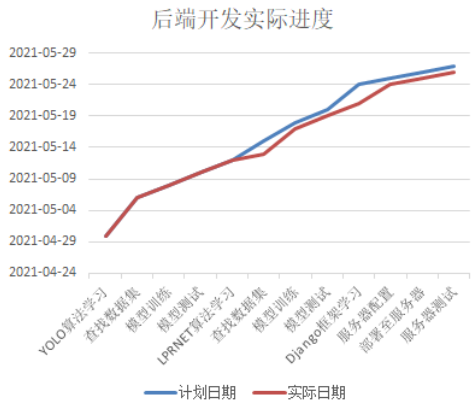
1. 项目背景

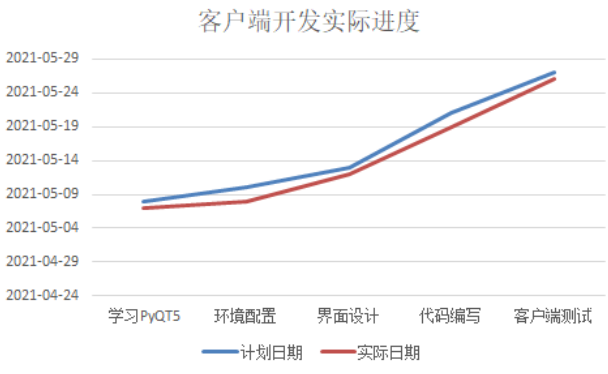
车牌识别项目主要用于对拍摄的汽车牌照或视频中的汽车进行车牌的识别检测。因为当前汽车的数量也众多，由汽车而引起的交通事故也很多，因此对于车牌精准的识别就显得十分重要且意义重大。车牌识别系统是智能交通系统的重要组成部分，有着广泛的应用， 同时也是计算机视觉、 图像处理和模式识别等交叉学科的研究热点。 车牌识别系统广泛应用于高速公路自动收费和超速监管系统、 公路流量监控系统、 停车场收费 管理系统、安防系统以及小区物业管理系统等等

2. 项目结构

1. 客户端：使用PyQt5设计客户端。客户端包括图像采集，车牌检测，车牌编码，车牌上传，接收服务端返回信息等部分。
    1. 图像采集：调用本地摄像头，或者网络摄像头（下一版本会加入网络摄像头功能，目前只是先调用本地摄像头）。
    2. 车牌检测：程序每隔0.5s会调用YOLO模型对当前图像帧进行检测，如果没检测到车牌就跳过，如果检测到车牌就会切割车牌部分图像，并将切割得到车牌图像呈现在客户端界面上，然后进行下一步处理。
    3. 车牌编码：本部分将车牌检测得到的车牌部分图像进行Base64编码。
    4. 车牌上传：将得到的图像编码通过POST方式发送至服务端。
    5. 接收返回信息：服务端进行处理之后，返回带有车牌字符的json格式文件，对其进行处理之后将车牌呈现在客户端界面上。
2. 服务端：使用Django框架搭建服务端，然后将服务端部署在阿里云服务器上。包括接收图像，车牌预测，返回信息
    1. 接收图像：接收来自客户端的车牌Base64编码，然后解码为图像。
    2. 车牌预测：使用LPRNet对解码后的图像进行预测，返回字符串。
    3. 返回信息：将预测得到的车牌字符封装为json格式文件然后返回给客户端。

3. 项目进度





项目总体符合预期，并且在计划的任务节点完成。

4. 机器学习模型

YOLO目标检测模型

1. 使用darknet框架训练YOLO模型，实现检测车牌
2. 训练YoLo模型的数据集来自于Gitee

名称	修改日期	类型	大小
0096.jpg	2019-03-26 20:27	JPG 文件	24 KB
0096.txt	2021-05-09 9:33	文本文档	1 KB
0097.jpg	2019-03-26 20:27	JPG 文件	16 KB
0097.txt	2021-05-09 9:33	文本文档	1 KB
0098.jpg	2019-03-26 20:27	JPG 文件	38 KB
0098.txt	2021-05-09 9:33	文本文档	1 KB
0099.jpg	2019-03-26 20:27	JPG 文件	43 KB
0099.txt	2021-05-09 9:33	文本文档	1 KB
0100.jpg	2019-03-26 20:27	JPG 文件	41 KB
0100.txt	2021-05-09 9:33	文本文档	1 KB
0101.jpg	2019-03-26 20:27	JPG 文件	35 KB
0101.txt	2021-05-09 9:33	文本文档	1 KB
0102.jpg	2019-03-26 20:27	JPG 文件	48 KB
0102.txt	2021-05-09 9:33	文本文档	1 KB
0103.jpg	2019-03-26 20:27	JPG 文件	21 KB
0103.txt	2021-05-09 9:33	文本文档	1 KB
0104.jpg	2019-03-26 20:27	JPG 文件	17 KB
0104.txt	2021-05-09 9:33	文本文档	1 KB
0105.jpg	2019-03-26 20:27	JPG 文件	39 KB
0105.txt	2021-05-09 9:33	文本文档	1 KB

每张含有车牌的`jpg`照片对应一个`txt`文本标注文件

例如：

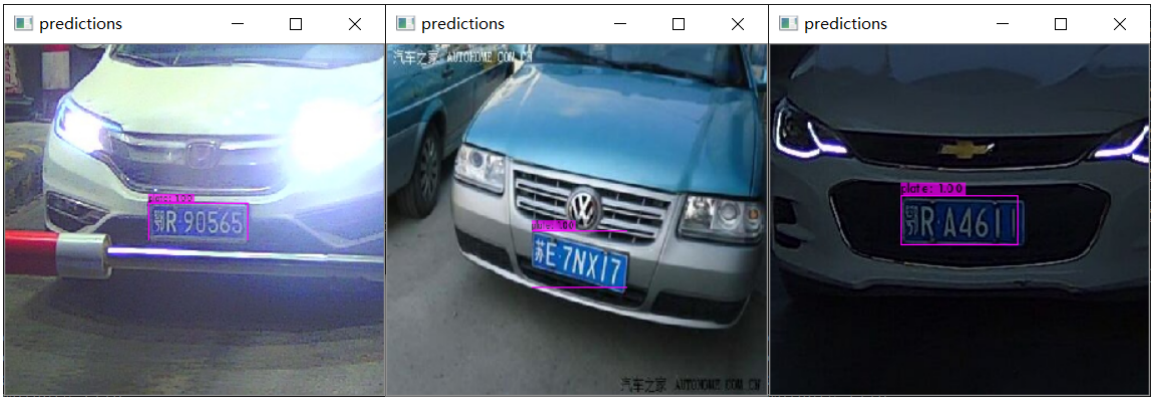
对应的文本标注文件就是

> 0 0.5112903225806451 0.5192307692307693 0.5387096774193548 0.3034188034188034

对应的解释如下：

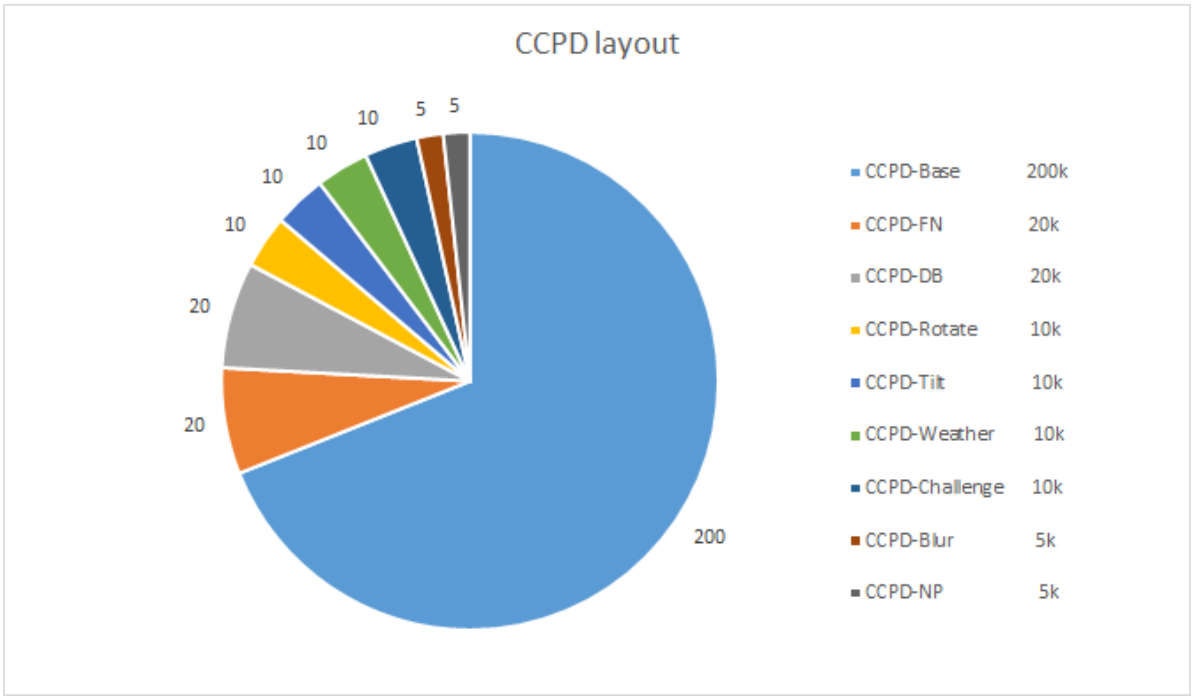
- 第一个数字`0`：**第0类**（本数据集只设置一个车牌类别所以只有0）
- 第二个数字`0.5112903225806451`：**中心点横坐标**
- 第三个数字`0.5192307692307693`：**中心点纵坐标**
- 第四个数字`0.5387096774193548`：**标注框宽度**（相对宽度）
- 第五个数字`0.3034188034188034`：**标注框高度**（相对高度）

检测效果



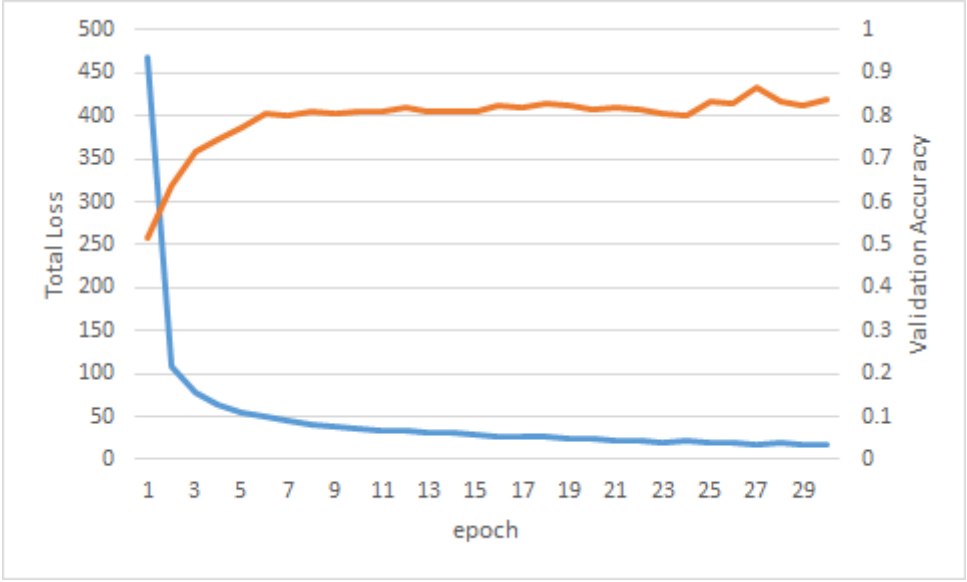
LPRNet车牌识别模型

1. 使用PyTorch训练车牌识别模型
2. 数据集来自于CCPD[1]，数据集由中科大人员进行制作，该数据集在合肥市的停车场采集得来的，采集时间早上7:30到晚上10:00.涉及多种复杂环境。包含的各种状况如下表：



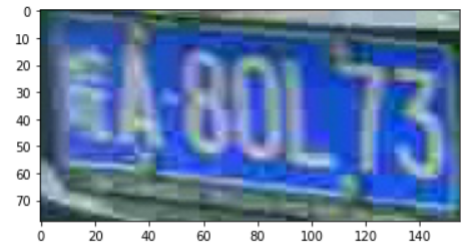
本数据集有一个缺陷：95%的数据均为安徽车牌，所有对其他地方的车牌识别的不是很好。

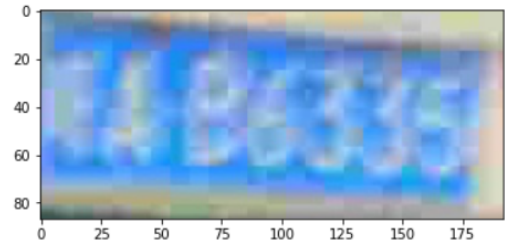
3. 模型训练：由于原始数据集较大，所以选取其中20%做训练集，10%做测试集，下图为训练结果



在测试集上的测试准确率为86.36%。

4. 案例：





5. 客户端

客户端使用PyQt进行编写。界面如下：

共分为5个区域：

1. 摄像头显示区域：负责显示摄像头获取到的视频。
2. 车牌显示区域：负责显示模型截取的车牌图像。
3. 车牌号显示区域：负责显示服务端返回来的车牌字符串
4. 开始识别按钮：点击之后系统开始进行车牌识别。
5. 停止识别按钮：点击之后系统停止车牌识别。

考虑到模型检测车牌需要耗费时间，服务器识别车牌需要耗费时间，所以本客户端设计为：每0.5s对当前摄像头捕获到的视频帧进行车牌检测，若检测到车牌存在，就将截取到的车牌图像上传到服务器进行识别。

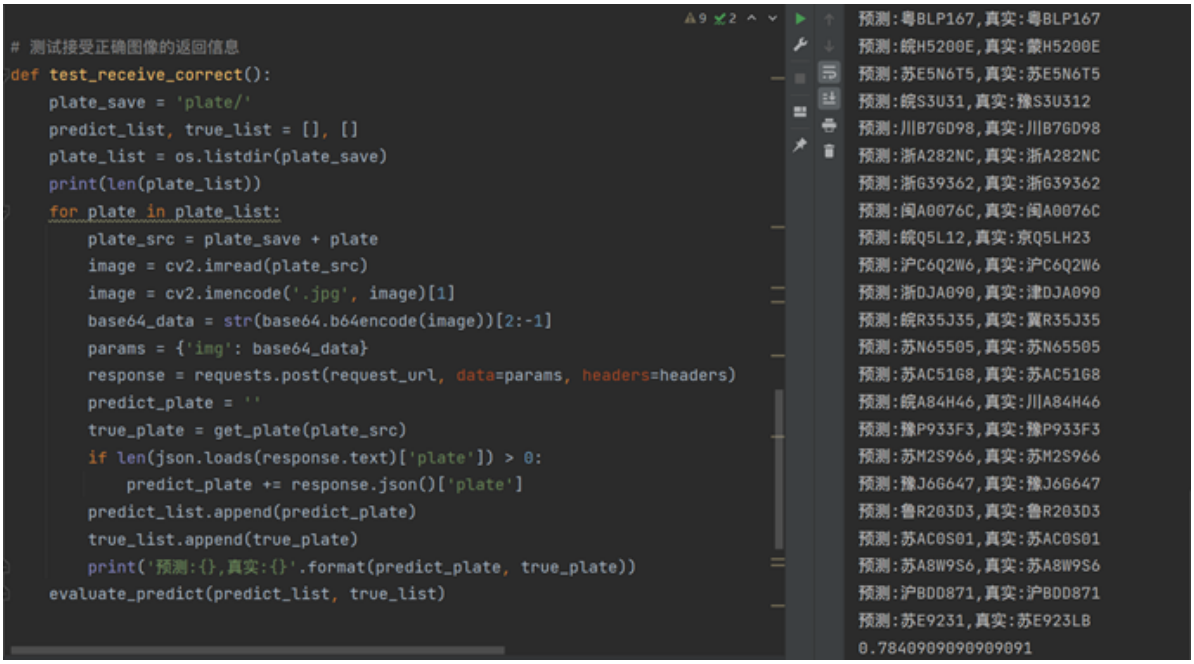
6. 存在的问题：

1. **问题描述**：LPRNet模型的数据集的广度不够，由于采用CCPD数据集，本数据集来自于中科大，所以本数据集95%的车牌数据都是皖A的车牌，所以在训练完成之后，再识别一些非常不清楚的外地车牌或者非车牌图像的时候，容易产生过拟合，将其错误地识别为皖A的车牌。
   解决方案：使用别的数据集，或者使用车牌生成器

2. **问题描述**：在设计软件之初，没有考虑到YoLo模型、LPRNet模型以及客户端服务器传输数据的耗时问题，所以制作了一个单线程的程序出来，在测试的时候页面非常卡顿，远远没有达到预期。
   解决方案：将耗时操作使用子线程来完成。
3. **问题描述**：在编写服务端代码的时候，为了快速完成基本功能，所以将来自客服端发送过来的车牌图像编码解码之后，直接存储为一张图片，然后客户端程序再进行读取图片操作。造成额外的IO时间。
   解决方案：使用`BytesIO`模块将解码后的图像直接使用字节流读进内存，再使用`Image`模块读取为图像，然后进行识别。

7. 测试

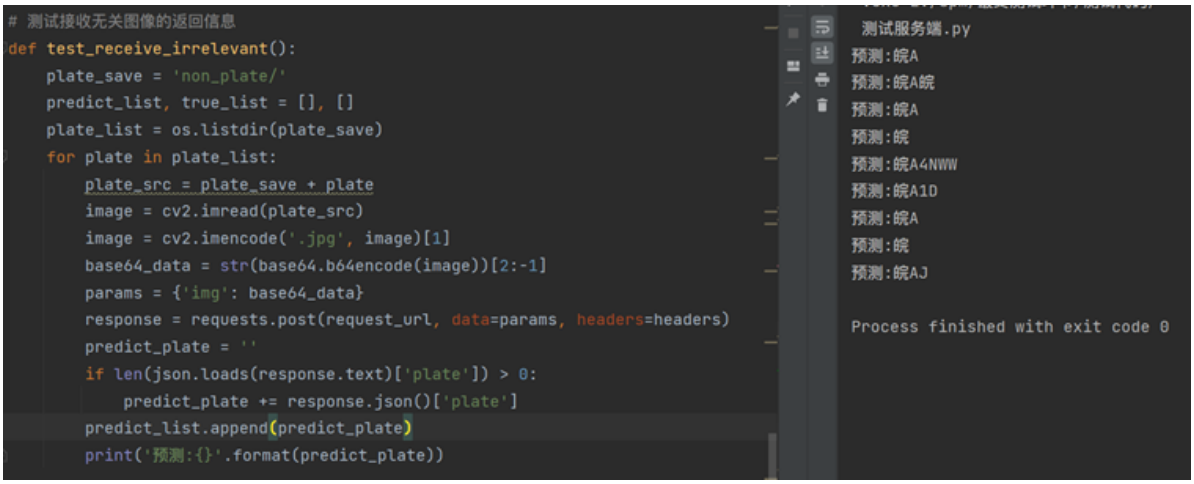
1. 测试服务端接收车牌图像进行识别：测试连续向服务端发送若干张车牌图像，最终得到的车牌识别的准确率为78%（下图为测试代码及结果截图）



```python
# 测试接受正确图像的返回信息
def test_receive_correct():
    plate_save = 'plate/'
    predict_list, true_list = [], []
    plate_list = os.listdir(plate_save)
    print(len(plate_list))
    for plate in plate_list:
        plate_src = plate_save + plate
        image = cv2.imread(plate_src)
        image = cv2.imencode('.jpg', image)[1]
        base64_data = str(base64.b64encode(image))[2:-1]
        params = {'img': base64_data}
        response = requests.post(request_url, data=params, headers=headers)
        predict_plate = ''
        true_plate = get_plate(plate_src)
        if len(json.loads(response.text)['plate']) > 0:
            predict_plate += response.json()['plate']
        predict_list.append(predict_plate)
        true_list.append(true_plate)
        print('预测:{},真实:{}'.format(predict_plate, true_plate))
    evaluate_predict(predict_list, true_list)
```

```
预测:粤BLP167,真实:粤BLP167
预测:皖H5200E,真实:蒙H5200E
预测:苏E5N6T5,真实:苏E5N6T5
预测:皖S3U31,真实:豫S3U312
预测:川B7GD98,真实:川B7GD98
预测:浙A282NC,真实:浙A282NC
预测:浙G39362,真实:浙G39362
预测:闽A0076C,真实:闽A0076C
预测:皖Q5L12,真实:京Q5LH23
预测:沪C6Q2W6,真实:沪C6Q2W6
预测:浙DJA090,真实:津DJA090
预测:皖R35J35,真实:冀R35J35
预测:苏N65505,真实:苏N65505
预测:苏AC51G8,真实:苏AC51G8
预测:皖A84H46,真实:川A84H46
预测:豫P933F3,真实:豫P933F3
预测:苏M2S966,真实:苏M2S966
预测:豫J66647,真实:豫J66647
预测:鲁R203D3,真实:鲁R203D3
预测:苏AC0S01,真实:苏AC0S01
预测:苏A8W9S6,真实:苏A8W9S6
预测:沪BDD871,真实:沪BDD871
预测:苏E9231,真实:苏E923LB
0.7840909090909091
```

2. 服务端接收非车牌图像进行识别：测试连续向服务端发送若干张非车牌的图像：由于训练LPRNet模型所使用的数据集95%的车牌数据都是皖A的车牌，所以在训练完成之后，再识别一些非常不清楚的外地车牌或者非车牌图像的时候，容易产生过拟合，将其错误地识别为皖A的车牌（下图为测试代码及结果截图）。



```python
# 测试接收无关图像的返回信息
def test_receive_irrelevant():
    plate_save = 'non_plate/'
    predict_list, true_list = [], []
    plate_list = os.listdir(plate_save)
    for plate in plate_list:
        plate_src = plate_save + plate
        image = cv2.imread(plate_src)
        image = cv2.imencode('.jpg', image)[1]
        base64_data = str(base64.b64encode(image))[2:-1]
        params = {'img': base64_data}
        response = requests.post(request_url, data=params, headers=headers)
        predict_plate = ''
        if len(json.loads(response.text)['plate']) > 0:
            predict_plate += response.json()['plate']
        predict_list.append(predict_plate)
        print('预测:{}'.format(predict_plate))
```

```
测试服务端.py
预测:皖A
预测:皖A皖
预测:皖A
预测:皖
预测:皖A4NWW
预测:皖A1D
预测:皖A
预测:皖
预测:皖AJ

Process finished with exit code 0
```

3. 服务端宕机/客户端未联网：当服务器宕机或者客户端未联网的时候，客户端的YOLO模型正常工作，可以检测到车牌，但是由于获取不到车牌字符，所以显示车牌字符的区域不会有任何字符出现（下图为测试截图）。

4. 客户端测试：检测客户端的YOLO模型的性能，输入100张含有车牌的图象，共检测到92张图像，本模型对一些特别模糊或者倾斜角度过大的图像检测能力较弱（下图为测试代码及结果截图）。

```python
# 检测Yolo模型效果
def test_yolo():
    count = 0
    num = 100
    car_img_save = 'car_img/'
    plate_save = 'plate/'
    net = yolo()
    car_img_list = get_random_car_img(num)
    shutil.rmtree(plate_save)
    os.mkdir(plate_save)
    for car_img in car_img_list:
        car_src = car_img_save + car_img
        img = cv2.imread(car_src)
        _, plates = net.return_frame(img)
        for plate in plates:
            count += 1
            cv2.imwrite(plate_save + car_img, plate)
    print('从{}张图像中获取到了{}张车牌图像'.format(num, count))
```

```
D:\Anaconda3\envs\pyqt\python
 .exe E:/spm/最终测试环节/测试代码/
 测试客户端.py
[ WARN:0] global
 C:\Users\appveyor\AppData\Local
 \Temp\1\pip-req-build-6uw63ony
 \opencv\modules\dnn\src\dnn.cpp
  (1442) cv::dnn::dnn4_v20201117
 ::Net::Impl::setUpNet DNN
 module was not built with CUDA
 backend; switching to CPU
从100张图像中获取到了92张车牌图像

Process finished with exit code 0
```

8. 风险分析

1. 技术上的问题：如开发环境配置不匹配等，影响项目的进行。
   应对策略：查阅资料，并向其他同学请教。
2. 其他事件影响项目的进度：如学校的一些活动如实习生招聘会、组员个人事情等会影响项目的进度。
   应对策略：组员及时开会讨论，研究应对策略，并统一时间解决项目上的问题。
3. 项目实际的工作量比预期大，在项目的实际开发中，遇到的问题比预期多，所花费的时间更长。
   应对策略：组员讨论安排时间解决项目上的问题，按照预期计划及时交付。

9. Lessons Learnt

在本次的项目开发中，我们学到了很多实际的技能。加强了团队合作，同时也对项目管理有了一个新的认识。但同时，我们也遇到了许多问题，如技术上碰到难题，其他事件也会影响项目的进度。虽然我们之前制定了风险计划，但实际中遇到问题往往更为复杂。 所以，在今后的项目开发中，我们应该制订更为详细的风险计划和应对方案，确保项目能够按时完成。

我们还学到了做项目要有管理意识，如里程碑、甘特图等可以帮助我们更好地完成项目，要按照计划的内容及时交付。

技术上需要加强。实际的开发过程中，我们意识到了自身的技术能力还有很大的不足，今后的软件项目开发中需要加大学习 时间的投入，提高自己的技术水平。

当出现严重问题时，需要根据现阶段状况重新评估，及时制定应对策略和计划，确保项目平稳进行下去。

一个项目的前期工作非常重要，若前期工作出现错误，那后面的工作就会遇到大麻烦。

要根据团队的实际能力，根据正常进度去做，项目实际开发过程中可能会碰到很多问题，组员之间要定时召开会议及时协商和沟通。

10. 未来的计划

1. 现在软件只能调用电脑本机的摄像头，所以需要改进，让软件能够扫描同一个局域网下的所有网络摄像头。能够自由选择对哪些摄像头采集到的信息进行识别
2. 目前软件的YOLO模型的检测速度大概为0.2s/张，没有调用GPU的算力，所以需要对模型进行改进，能够调用GPU，提高检测的速度。
3. 训练集大多数为蓝牌轿车，对其他类型的车牌识别效果较差，所以需要增加数据集的范围，让车牌能够识别更多的车牌类型
   扩大识别目标。
4. 在识别车牌的同时将车辆的特征信息也加入到识别范围内，还有对司机特征的识别。
5. 针对不同地方的车牌识别需求，对本软件进行优化。例如公路上需要快速的识别，所以对车牌的识别速度要求较高，停车场不需要较高的识别速度，可以将软件改成获取到车辆的触发信息然后进行识别，其他时间处于待机状态，这样也可以减少服务器的负载。
6. 优化服务端功能，目前是公开，所有用户均可向后端发送信息，然后获取返回值，后面会继续优化，对发送给服务端信息的用户进行鉴定，规范后端的使用权限。

11. Releases

本项目目前为止发布了两个版本，V0.1beta，V1.0。V0.1beta版本为单线程版本，界面卡顿。V1.0版本为正式版。从V1.0下载`.7z`压缩包，解压之后打开`.exe`文件可以直接使用(确保计算机有摄像头)。在未来的计划中会对1.0版本进行优化。

引用

- [1]. Xu Z, Yang W, Meng A, et al. Towards end-to-end license plate detection and recognition: A large dataset and baseline[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 255-271.