**《大数据架构与技术》**

**课程报告**

学 院： 计算机学院

专业班级：计算机科学与技术1班，计算机科学与技术6班，计算机科学与技术卓越1班，物联网工程1班

年 级： 2021

姓 名（所有人）： 刘铠，卢俊岩，李宗岳，黄奕铭

学 号（所有人）：20215295，20215289，20215290，20215341

完成时间： 2023 年 12 月 04 日

成 绩：

指导教师： 冯永

**目录**

[1 小组成员分工 1](#_Toc152884664)

[2 数据集采集及数据预处理 1](#_Toc152884665)

[2.1 数据集来源 1](#_Toc152884666)

[2.2 最后数据集格式 2](#_Toc152884667)

[2.3 数据预处理流程 3](#_Toc152884668)

[2.3.1 车牌检测 3](#_Toc152884669)

[2.3.2 字符分割 6](#_Toc152884670)

[2.4 数据集使用方法 7](#_Toc152884671)

[3 平台搭建详细过程 7](#_Toc152884672)

[3.1 集群1 7](#_Toc152884673)

[3.2 集群2 8](#_Toc152884674)

[4 测试案例运行过程及结果 12](#_Toc152884675)

[4.1 测试案例说明 12](#_Toc152884676)

[4.2 测试代码分析 12](#_Toc152884677)

[4.3 Map-Reduce算例执行过程与结果 12](#_Toc152884678)

[5 并行化算法设计运行的过程和结果 14](#_Toc152884679)

[5.1 多层感知机算法设计与实现 14](#_Toc152884680)

[5.1.1算法设计 14](#_Toc152884681)

[5.1.2 算法设计 15](#_Toc152884682)

[5.1.3 算法结果分析 16](#_Toc152884683)

[5.2 CRNN 17](#_Toc152884684)

[5.2.1 CNN 18](#_Toc152884685)

[5.2.2 RNN 19](#_Toc152884686)

[5.2.3 CTC loss 20](#_Toc152884687)

[5.2.4 CRNN算法实现 21](#_Toc152884688)

[5.2.5 算法结果分析 25](#_Toc152884689)

[6 实验结果可视化 26](#_Toc152884690)

[6.1 损失函数曲线与准确率曲线 26](#_Toc152884691)

[6.2 GUI设计 28](#_Toc152884692)

[7 课程总结 29](#_Toc152884693)

# 1 小组成员分工

本次课程项目我们小组的分工如下：

刘铠：算法设计，GUI可视化设计，课堂展示；

卢俊岩：数据预处理，PPT制作；

李宗岳：项目总设计，算法设计，数据预处理，PPT制作，报告撰写；

黄奕铭：PPT制作，数据集采集。

# 2 数据集采集及数据预处理

## 2.1 数据集来源

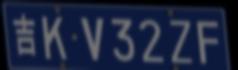
我们首先调查了车牌识别领域被广泛使用的两个数据集：

CCPD2019：中科大开源数据集，图片采集于合肥市停车场，只有蓝色车牌，并且车牌的分布不平衡，以皖为主。此数据集是完整汽车的图像，需要先进行车牌定位。

CBLPRD：高质量33万张各类车牌，包括普通蓝色车牌，新能源绿色车牌等等，各省样本分布平衡且无需进行车牌定位。

我们查看了上面两个数据集，发现车牌的亮度普遍较高，查阅数据集说明发现采集时间为白天，因此我们认为这并不能代表所有情况下采集到的车牌，所以我们决定使用代码生成部分车牌来模拟夜间的情况，下面简述代码生成逻辑：

首先，从一个设定的字符库中随机挑选元素拼接出虚拟的车牌号码，然后将这些号码以特定的字体和大小渲染到图像上以模仿实际的车牌。接着，这个图像会经历一系列的处理，比如透视变换、色彩调整和模糊等操作，以模拟车牌在现实世界中因视角，光照和运动等因素产生的变形和失真。最后，处理完的车牌图像连同车牌上的文本信息一起保存下来，以便用来训练或验证自动车牌识别系统。



**图表 1 代码生成车牌示例**

可以看到我们代码生成的图片在很大程度上模仿了夜间光线较暗的条件下拍摄的车牌，并且有一定的模糊度，倾斜度，更加符合实际。

## 2.2 最后数据集格式

经过上述数据集收集过程，我们项目使用的数据集如下：

选取CBLPRD中所有的蓝色普通车牌，选取部分CCPD中的车牌，共计8万余张车牌图片，同时生成了2万张车牌图片，模拟夜间拍摄的得到的车牌，以此增加模型的泛化能力。

训练集：92462张图片，测试集：8600张图片

我们的数据集有以下特点：

（1）数量大：该数据集包含了10万多幅中国城市车牌图像

（2）多样化：这些车牌照片涉及多种复杂环境，包括模糊、倾斜、亮度低。

该数据集所有图片都为jpg格式，图片命名规则如下：

首先我们定义了列表，这个列表中包含所有车牌中可能出现的汉字，字母，数字（其中字母O，I不会出现在车牌中，因为和数字0，1的相似度过高）列表如下：

**["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", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "J", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]**

然后，我们将车牌上的汉字，字母和数字通过此列表映射到它们在列表中的索引值，元素之间使用下划线连接，最终得到的图片名称举例如下：

**0\_42\_35\_54\_57\_44\_31.jpg**



**图表 2 部分数据集截图**

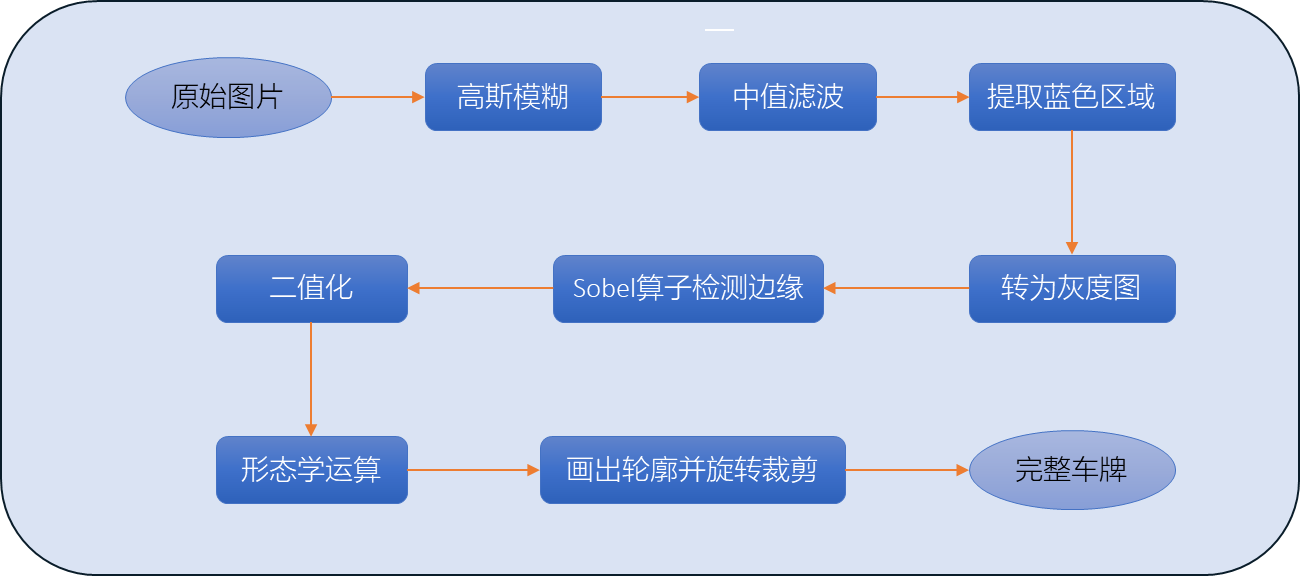
我们可以看到上图中比较暗的车牌为我们使用代码生成的车牌，它经过我们代码中的处理，已经接近现实生活中真实存在的车牌。并且每一张车牌的角度，亮度，在图片中所占的比例和位置都各不相同，这证明我们的数据集贴合现实情况，而使用我们的数据集训练出的模型具备一定的泛化能力。

## 2.3 数据预处理流程

基于我们的数据集，我们的数据预处理流程包含两个部分车牌检测与字符切割，车牌检测为从原图片中定位车牌的位置并切割出车牌区域，字符分割为将完整车牌中的字符进行分割，得到单个字符图片。

### 2.3.1 车牌检测

此步骤我们主要基于Python的openCV库进行操作，由于原始图片差异化明显，所以采用什么操作处理图片，这些操作的顺序如何对于最终的检测效果尤为重要，通过不断地尝试和修改，最终选择效果最好的车牌检测的流程图如下：



**图表 3 车牌检测流程图**

下面解释每一步所做的操作以及对应的作用：

高斯模糊：通过在图像上应用高斯函数来减少图像的高频细节信息，从而实现图像的模糊效果。此步骤用来减少图像噪声、平滑边缘、降低图像的细节。

中值滤波：使用像素周围邻域内的中值来代替原始图像中的每个像素值。与其他滤波器不同，中值滤波器不会改变图像的边缘和轮廓信息，而是通过消除图像中的孤立噪声点和离群像素来减少图像中的噪声。中值滤波器可以有效地去除椒盐噪声、斑点噪声等随机噪声，恢复图像的清晰度和细节。

提取蓝色区域：我们此次项目针对蓝色普通车牌，所以定义蓝色在HSV颜色空间中的范围，用于检测蓝色，并创建掩码以提取图像中的蓝色区域，通过实验证明此方法效果优于其他方法。

转为灰度图：简化图像处理过程，减少计算复杂度，并保留图像的大部分结构信息，便于后续边缘检测。

Sobel算子检测边缘：通过计算图像中像素点的亮度梯度，找出图像中的边缘区域。突出图像中的边缘轮廓，用于图像处理中的边缘检测、特征提取和图像分割等应用。

二值化：将一幅灰度图像转换为只有两个值的图像。通过设定一个阈值，将图像中的像素值分为两类：低于阈值的像素被设为0（黑色），高于阈值的像素被设为255（白色）。便于后续的形态学运算。

形态学运算：将图像中的目标区域扩张，以增强目标的大小、连通性和形状。膨胀操作可以用于填充空洞、连接断裂的边缘，使得图像轮廓更加明显和清晰。

画出轮廓并旋转裁切：利用矩形参数计算出变换矩阵，接着用该变换矩阵对原始图像进行旋转操作，再进行裁剪后返回旋转后的图像。如果裁剪得到的图像宽度小于高度，则额外进行一个顺时针旋转90度的操作。

下面为我们车牌检测阶段每一步操作的效果图以及最终的效果图：



**图表 4 车牌检测效果图（1）**



**图表 5 车牌检测效果图（2）**



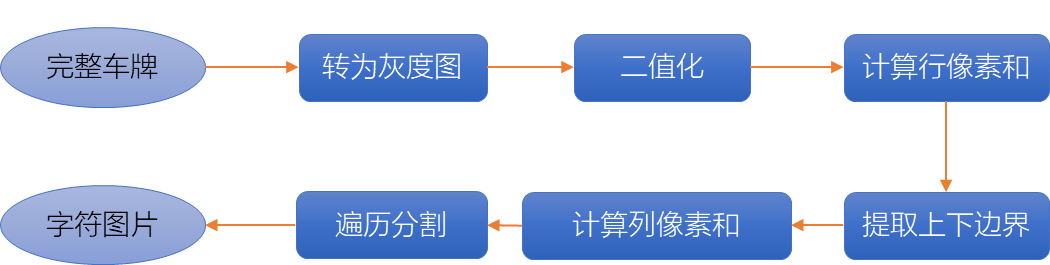
**图表 6 车牌检测结果图示例**

由于我们的数据集来源于三个部分，所以在我们进行车牌检测后，我们统一图片大小为238×70，以便于后续操作。

### 2.3.2 字符分割

后续，我们使用两种方法训练模型，其中，对于多层感知机（MLP），我们的输入为单个字符向量，所以，此处我们需要对上一步裁切出的完整车牌进行字符分割以获取单个字符。

我们字符分割的流程图如下：



**图表 7 字符切割流程图**

转为灰度图和二值化此处不再分析，重点分析提取行像素和以及遍历分割：

提取上下边缘：

对二值化图像的每一行进行求和（水平方向），得到每行的像素和，形成行直方图数据。根据行直方图和设定的阈值（由最小值和平均值得出）找到上下边界。然后根据找到的边界截取二值化图像的有效部分，即去除了顶部和底部的无用边缘后的图像。

分割字符：

首先，分析处理后的二值图像，计算每一列的白色和黑色像素总和。然后确定字符是否为黑底白字还是白底黑字，取决于黑白像素统计的最大值。根据统计数据和阈值，确定每个字符的起始和结束位置。每当找到一个字符的结束位置后，将该字符的图像部分截取出来。最后将截取的字符图片大小调整为统一的尺寸，并保存到指定的文件夹路径。

最后，我们的字符分割效果如下图所示：

****

**图表 8 字符切割效果图**

然后，我们将字符图片归类存储，然后展开成为一维向量，以csv格式的文件存储。当然，由于字符分割算法无法实现将所有的车牌都正确分割，所以我们用于MLP的训练集小于未分割前的训练集。但是，我们的CRNN仍旧使用的是车牌检测后的完整车牌的训练集，即包含92462张图片的训练集。

## 2.4 数据集使用方法

我们的数据集在课程资料包中，对于MLP模型的训练，我们已经将字符图片转成了一维向量并存储在了output1.csv，output2.csv，test1.csv，test2.csv中，将其上传到hdfs系统后，修改rdd\_mlp.py文件中的对应路径即可运行。对于CRNN模型的训练，我们的训练集为final\_train文件夹，其中包含了我们已经进行了车牌检测后的完整车牌图片，测试集为final\_test文件夹，将两个文件夹上传hdfs文件系统后，同时上传final\_train.txt和final\_test.txt。这两个文件用于存储每一张图片的路径，注意修改为实际路径。最后修改mydataset.py对应读图片的路径，使得能够在hdfs中读图片，即可进行模型训练。

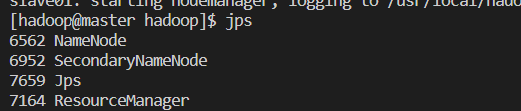
# 3 平台搭建详细过程

本项目基于Hadoop+Spark分布式集群，基于项目需要，本小组搭建了两个集群，下述搭建集群的详细过程。

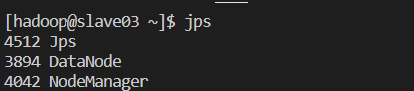
## 3.1 集群1

此集群为实验时已经部署好的Hadoop+Spark集群，采用完全分布式部署。Hadoop版本为2.10.2，Spark版本为3.3.3。此集群的所有节点均为只有CPU的云服务器，满足了数据预处理，传统机器学习模型的分布式训练的需要。在每台云服务器上，我们进行了相应的Python库的安装，包括：visualize，jieba，wordcloud，snapshot\_selenium，pyecharts，skimage，numpy， pandas，matplotlib等。

首先配置Hadoop集群，在master节点上修改文件，然后将配置文件发送给slave节点进行配置。配置完成并启动Hadoop集群，master和slave分别输入jps命令查看，如下截图证明集群部署成功：

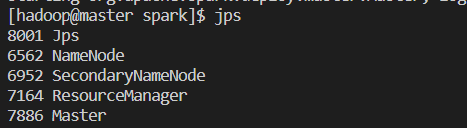
****

**图表 9 Master节点输入jps检验Hadoop集群启动**

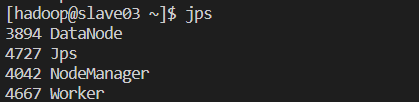


**图表 10 Slave节点输入jps检验Hadoop集群启动**

配置相应的Spark文件，然后启动Spark集群，如下截图证明配置成功：

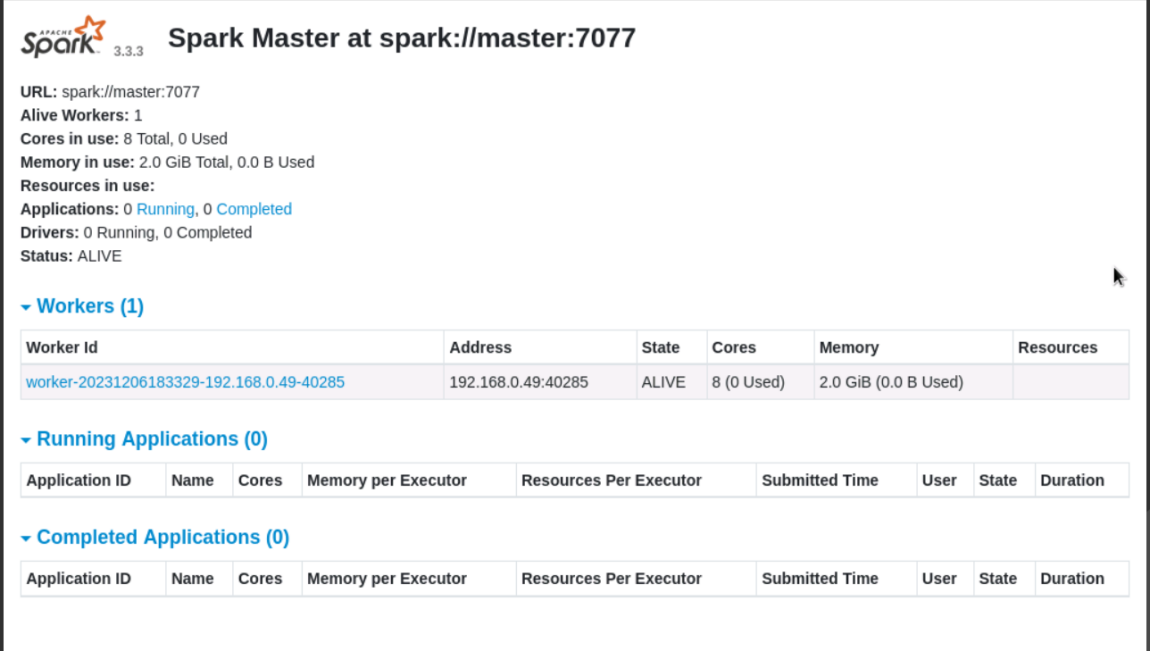


**图表 11 Master节点输入jps检验Spark集群启动**



**图表 12 Slave节点输入jps检验Spark集群启动**

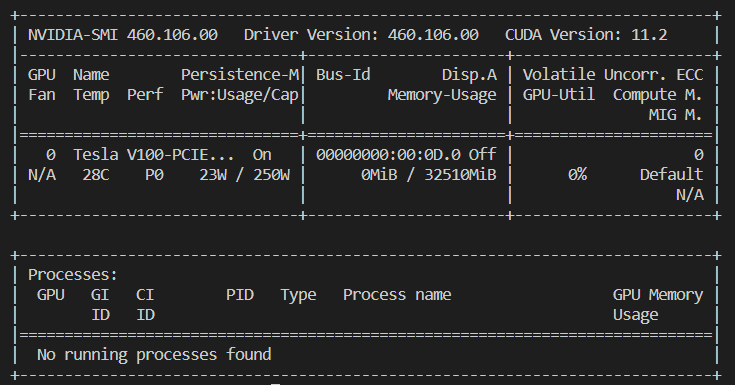
在web UI输入master:8080显示如下截图，证明配置成功：



**图表 13 Web UI查看Spark集群启动**

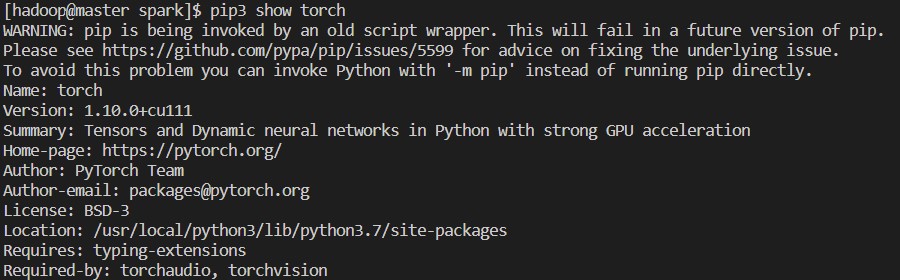
## 3.2 集群2

为了实现分布式环境下训练深度学习模型，我们重新搭建了Hadoop+Spark集群，此集群的节点我们使用了两台GPU云服务器，显卡型号为 NVIDIA V100-PCIe-32G，此显卡适用于深度学习计算。首先我们配置了cuda和cuda driver：



**图表 14 查看cuda和cuda driver**

基于此cuda版本，我们首先安装了pytorch深度学习框架，安装完成后，输入相关命令查看Pytorch信息：



**图表 15 查看Pytorch版本**

可以看到Pytorch版本为1.10.0，对应的cuda版本为11.1，由于Pytorch所支持的版本的必须兼容服务器cuda版本，所以此处使用11.1版本的正确的。

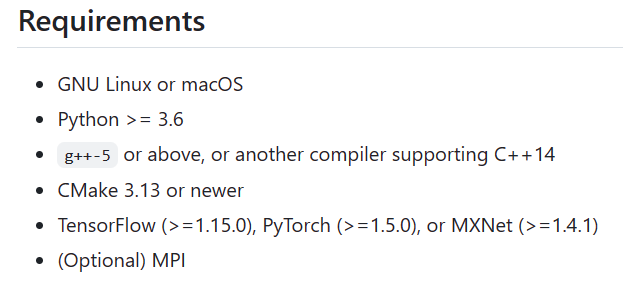
由于Pytorch编写的代码无法在分布式环境下直接运行，我们需要使用其他的分布式深度学习框架，在经过多种方案的对比后，我们选择了使用Horovod分布式深度学习框架。

Horovod的核心卖点在于使得在对单机训练脚本尽量少的改动前提下进行并行训练，并且能够尽量提高训练效率。它支持不同的前端训练框架和底层通信库（英伟达的NCCL以及Intel的oneCCL），同时也可以支持运行在Spark/Ray集群上。

Horovod.Spark通过将Horovod集成到Spark的数据处理流程中，实现了从数据加载、分布式训练到模型保存的端到端流程。它能够有效地利用Spark集群的计算资源，实现高效的深度学习训练。

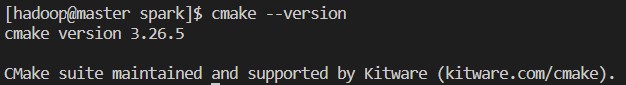
本项目中，我们使用Horovod将Pytorch编写的代码在Spark集群中运行。

但是，在安装Horovod之前，基于官方安装指南，我们必须配置相应的环境来支持Horovod，以下为Horovod的环境配置要求：

****

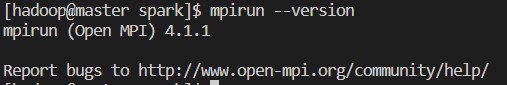
**图表 16 查看Horovod安装环境依赖**

所以我们首先安装了CMake 3.26.5：



**图表 17 查看CMake版本**

安装OpenMPI 4.1.1：

****

**图表 18 查看OpenMPI版本**

同时，由于我们最重要实现的是多GPU分布式计算，所以还需要安装NCCL，NCCL是一种在多GPU之间进行高效、高性能的通信和数据传输的库，可用于加速深度学习训练，提高训练效率和成果：

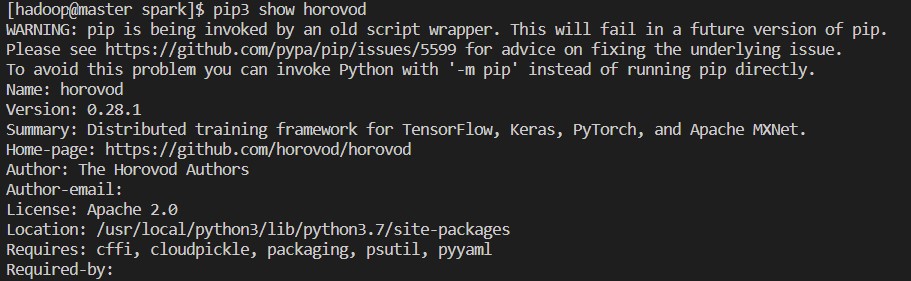


**图表 19 查看NCCL版本**

最后使用如下命令安装Horovod：

**HOROVOD\_GPU\_OPERATIONS = NCCL pip install --no-cache-dir horovod[pytorch,spark]**

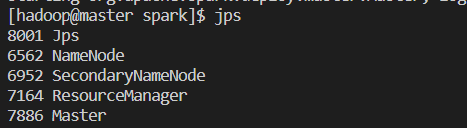
安装完成后，使用相关命令查看是否安装成功：

****

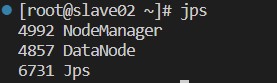
**图表 20 查看Horovod版本**

可以看到我们安装的是0.28.1版本的Horovod，与我们前面配置的环境兼容。

配置Hadoop集群和Spark集群与集群1的步骤相同，配置完成后，启动Hadoop、集群和Spark集群，输入相关命令查看：

****

**图表 21 Master节点输入jps检验Spark集群启动**

****

**图表 22 Slave节点输入jps检验Spark集群启动**

证明集群2配置完成。

至此，我们用于此项目的两个完全分布式集群搭建完成，后续，我们在集群1上进行了Map-Reduce算例以及MLP模型的分布式训练，在集群2上进行了深度神经网络的训练。

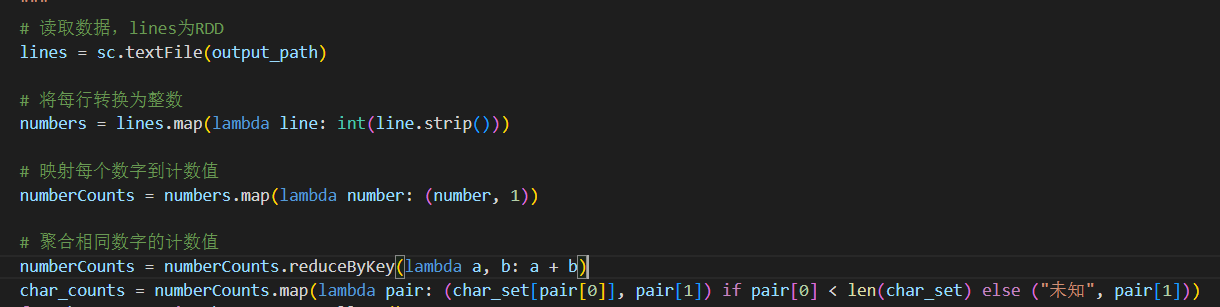
# 4 测试案例运行过程及结果

## 4.1 测试案例说明

在部署完分布式环境，完成数据集采集和相应的预处理后，我们首先编写了一个wordcount算例，运用了map函数和reduce函数，统计我们的数据集中各省或自治区、直辖市的车牌数量，验证环境部署成功，同时查看我们数据集样本的分布是否均衡。

## 4.2 测试代码分析

词频统计部分，首先读取对应的文件并以RDD的形式进行存储，然后使用map函数将值转换为数字，接着使用map函数将每个数字映射成计数值，然后使用reduceByKey聚合相同数字的计数值，得到每个省的车牌数量。

****

**图表 23 测试算例相关代码（1）**

后续调用Pyecharts库中的Bar绘制柱形图，相关代码如下：



**图表 24 测试算例相关代码（1）**

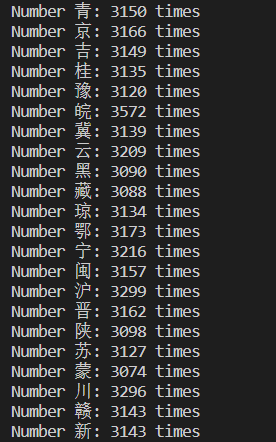
## 4.3 Map-Reduce算例执行过程与结果

算例执行过程中，Spark UI查看发现wordcount进程正在执行。



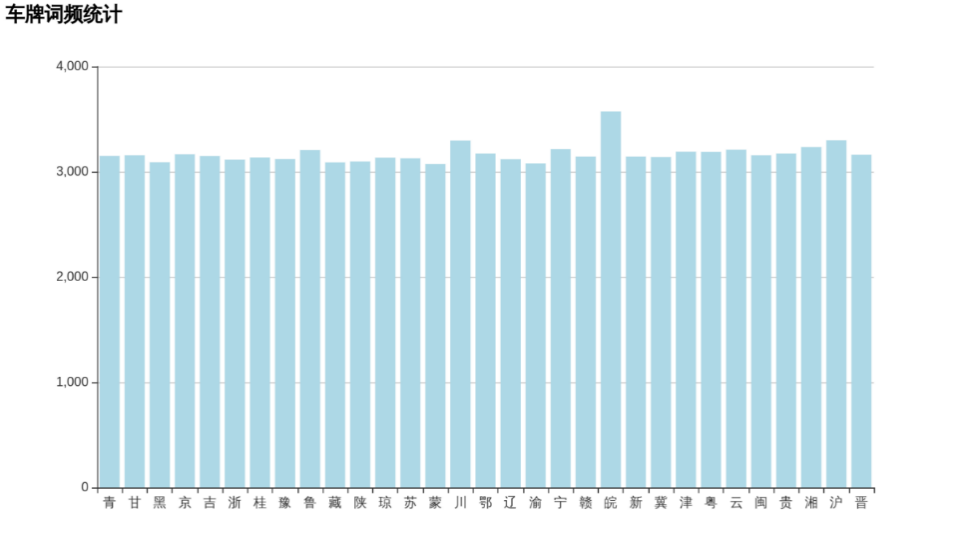
**图表 25 Spark UI 进程运行中截图**

控制台打印输出如下：

****

**图表 26 Map-Reduce算例控制台输出**

最终我们得到的结果如下：

****

**图表 27 Map-Reduce算例结果柱形图**

通过这张图，我们验证了Hadoop+Spark完全分布式部署成功，并且检验了数据集中的样本分布，可以看出我们数据集中各类样本的数量基本均衡。

同样的，我们也生成了词云图，图片如下：

****

**图表 28 Map-Reduce算例结果词云图**

# 5 并行化算法设计运行的过程和结果

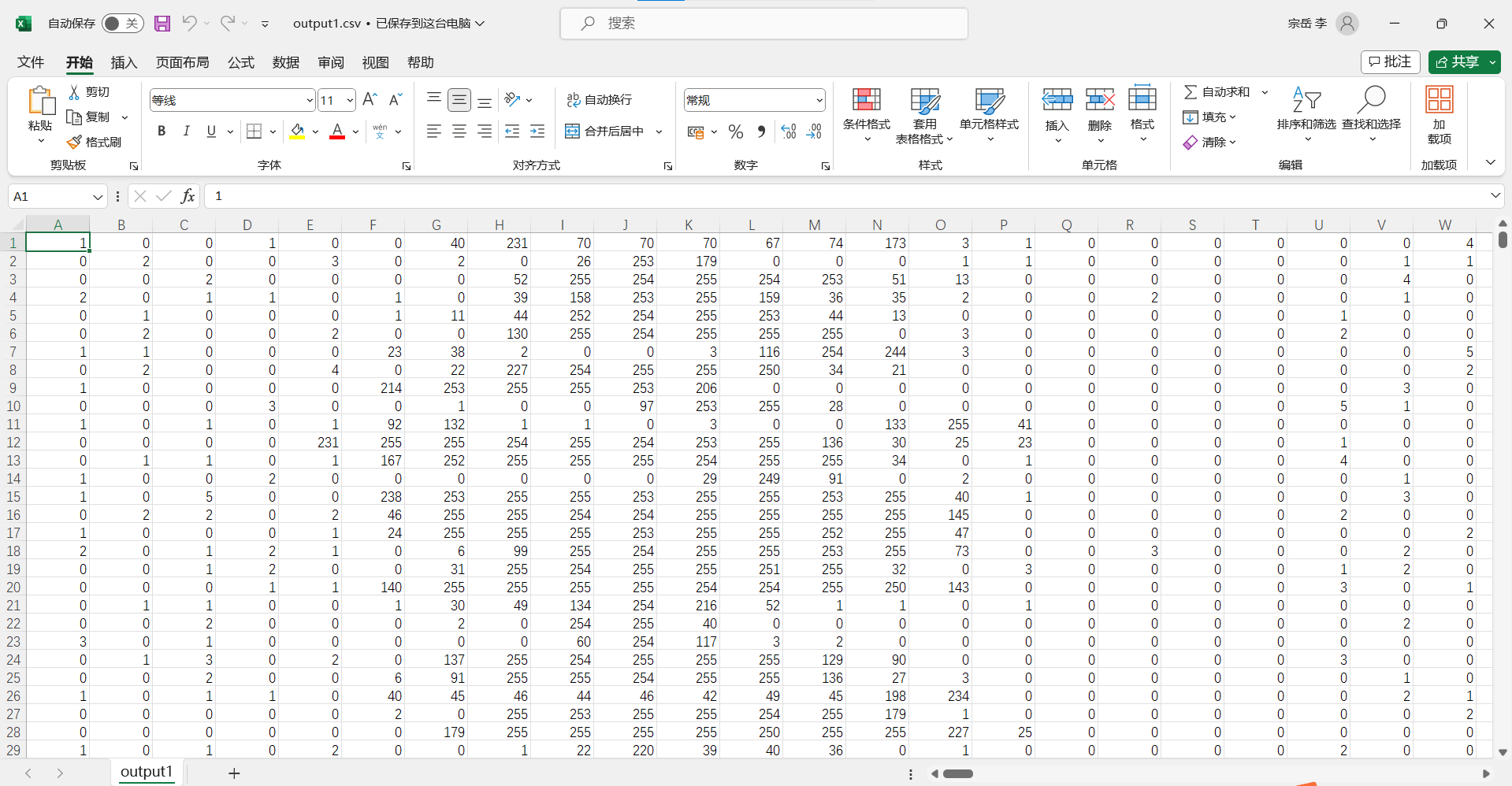
## 5.1 多层感知机算法设计与实现

### 5.1.1算法设计

我们首先使用Spark.MLlib里的Multilayer Perceptron Classifier构造多层感知机分类器，由于车牌第一位是汉字其他位为数字或字母，所以我们构造了两个分类器，用于汉字分类和数字字母分类。

**输入层**：由于图像本身较小，我们将其统一为20×20，然后直接将其展开成一维向量作为输入。

我们将展开后获得的一维向量存入csv文件（如下图），然后将csv文件上传到hdfs文件系统，便于分布式训练时的数据读取。

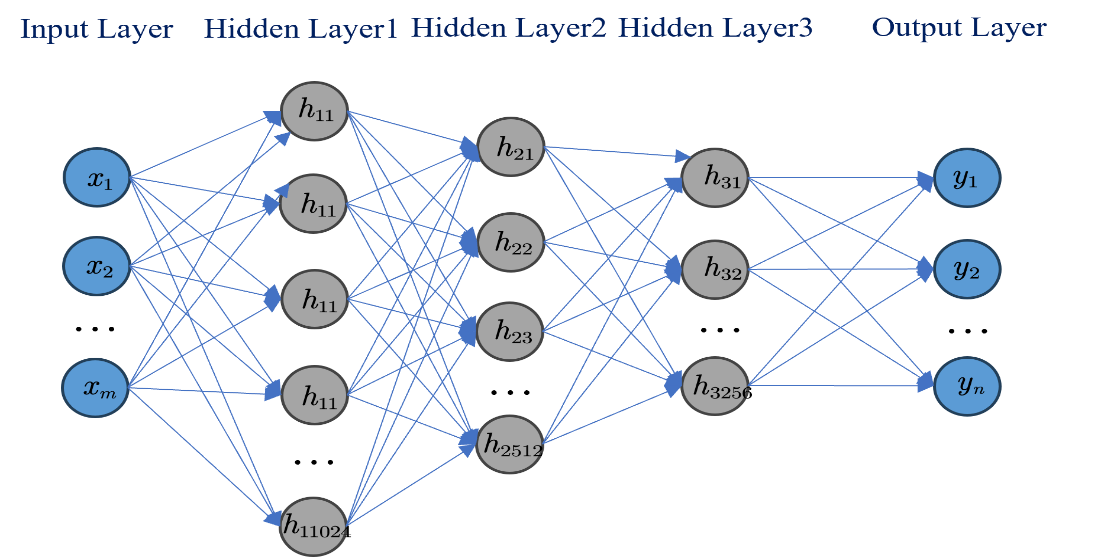
****

**图表 29 字符向量csv文件截图**

**隐藏层**：在经过对隐藏层数量和隐藏层神经元个数不断进行调整和试验后，我们发现增加隐藏层数量不一定会增加分类效果，并且隐藏层神经元也不是越多越好，最终，我们确定了隐藏层的数量为3，隐藏层神经元数量分别为1024，512，256。

**输出层**：根据汉字类别数量和数字字母类别数量分别确定。汉字为31类，数字和字母为34类。

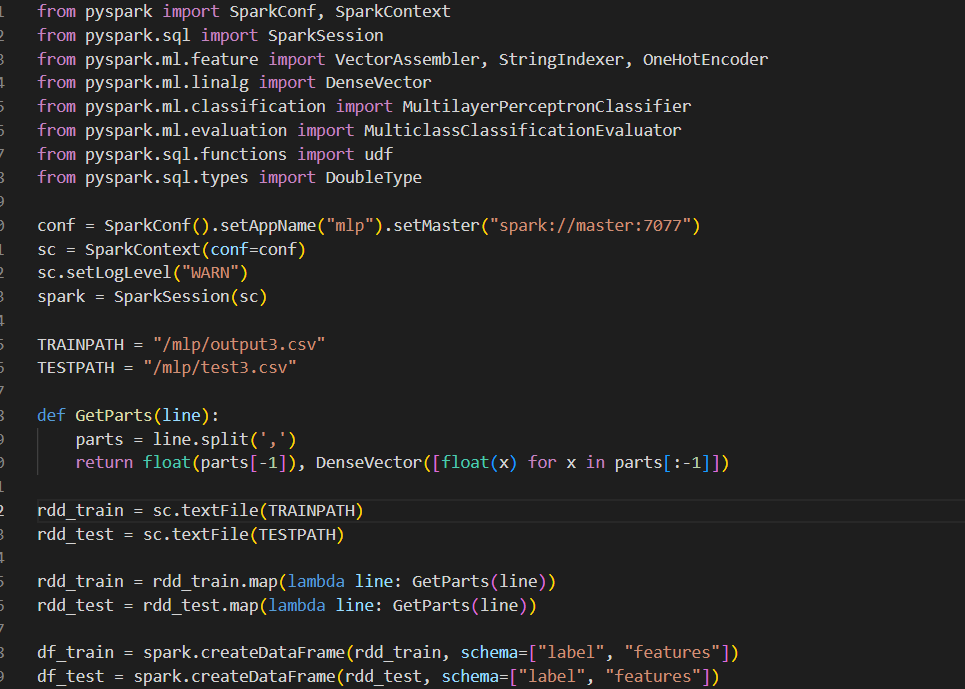
以下为我们构造的多层感知机分类器（MLP）的结构图：

****

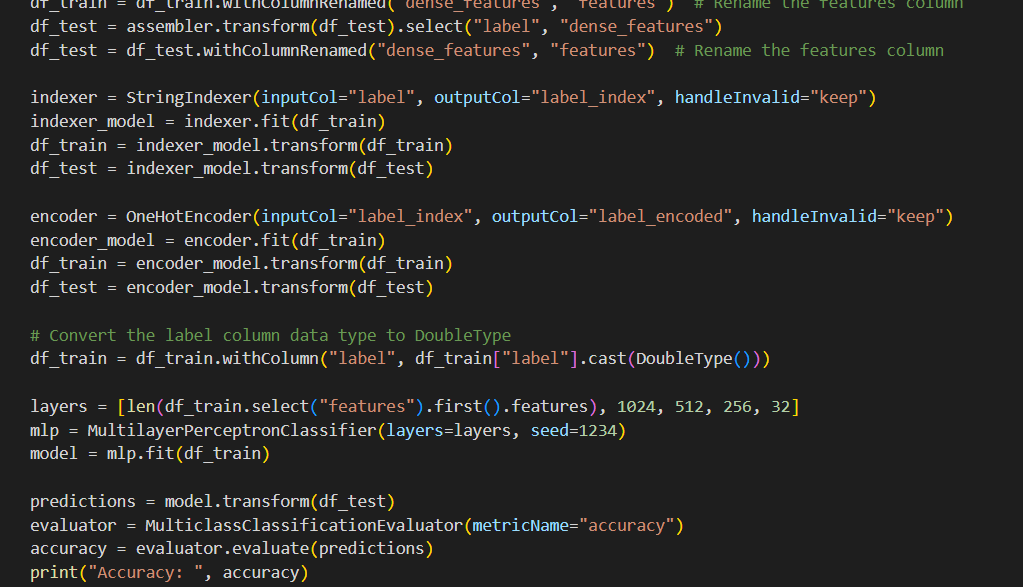
**图表 30 多层感知机分类器结构图**

### 5.1.2 算法设计

以下是相关代码截图：

****

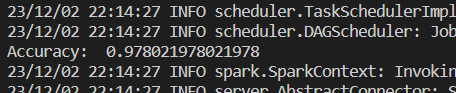
**图表 31 MLP代码截图（1）**

****

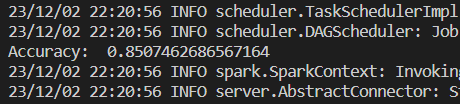
**图表 32 MLP代码截图（2）**

### 5.1.3 算法结果分析

我们分别训练和测试了汉字分类器和数字字母分类器，两分类器在测试集上的效果如下图所示：

****

**图表 33 数字字母分类器准确率**

****

**图表 34 汉字分类器准确率**

我们发现，多层感知机（MLP）分类器在数字和字母的分类上达到了不错的效果，准确率为97%。但是，对于汉字，多层感知机（MLP）的分类效果始终未达到较好的水平，仅有85%。

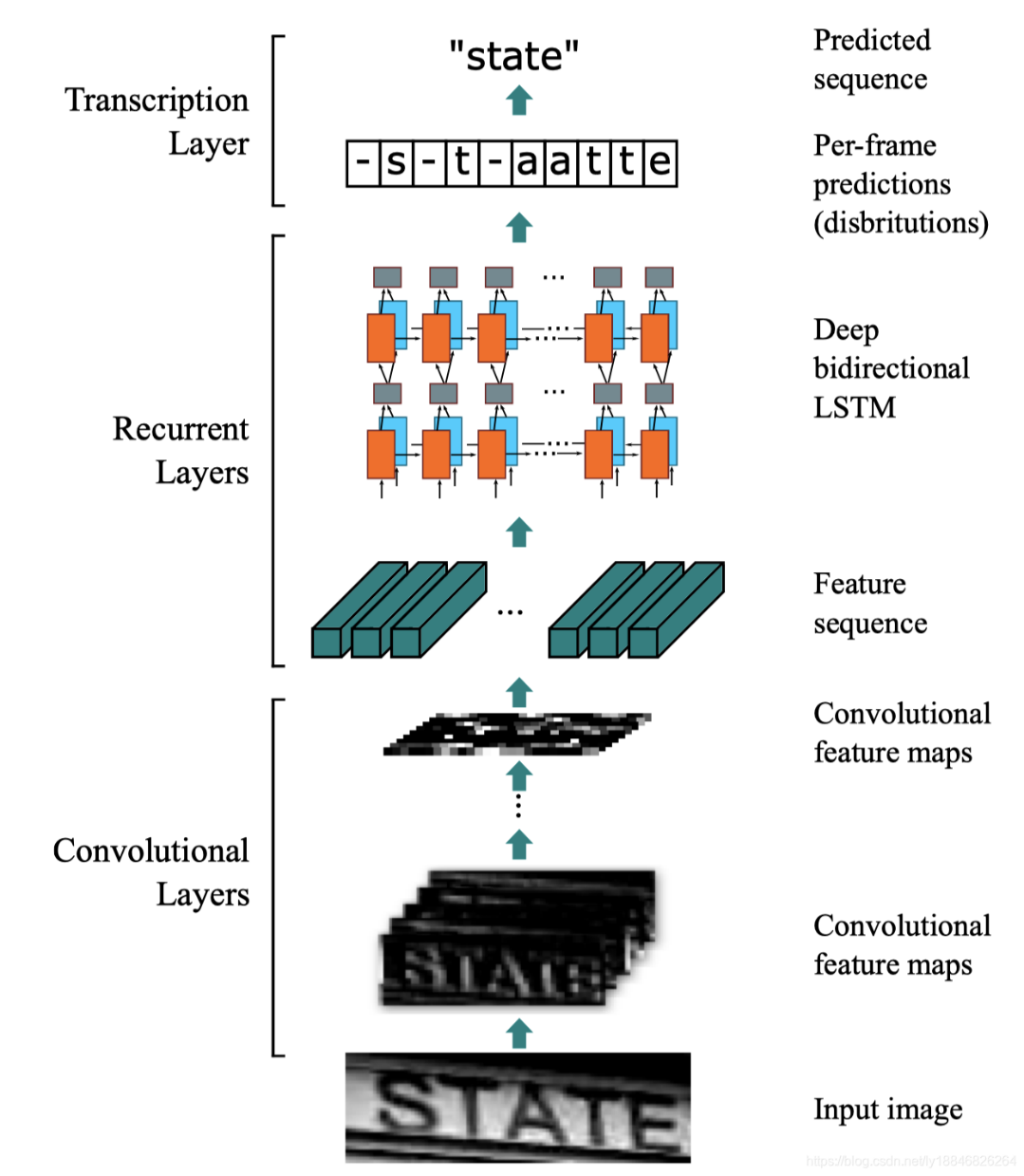
结合实际情况，对于一张普通蓝色车牌，其包含七个字符，只有七个字符全部识别正确，我们才会认为此车牌识别是正确的，那么即使假设所有字符的识别准确率都达到97%，如果计算准确率的七次方，则最终识别车牌的准确率也只有80%，这并未达到我们预期的效果。

## 5.2 CRNN

显然，MLP的识别率还有很大的提升空间，但是，受限于算法本身以及输入向量的维数，识别准确率很难通过调整参数来实现质的提高。于是，为了提高识别准确率，我们使用卷积循环神经网络（CRNN）来进行进一步的算法设计与实现，以提高准确率。

相比于MLP，CRNN综合了CNN和RNN（LSTM），所以对于序列数据和图像数据，拥有更好的处理能力，而车牌正是一种图像中的序列数据。

以下是我们设计的CRNN的整体结构图：

****

**图表 35 CRNN的整体结构图**

CRNN网络结构包含以下三部分：

（1）CNN（卷积层），使用深度CNN，对输入图像提取特征，得到特征图；

（2）RNN（循环层），使用双向RNN（BLSTM）对特征序列进行预测，对序列中的每个特征向量进行学习，并输出预测标签（真实值）分布；

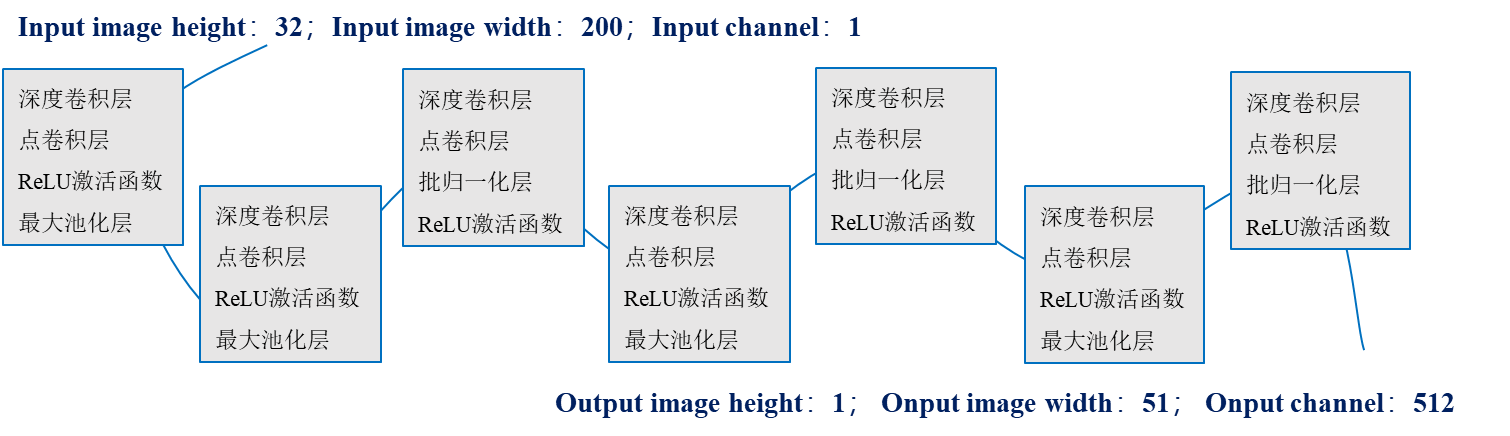
（3）CTC loss（转录层），使用 CTC 损失，把从循环层获取的一系列标签分布转换成最终的标签序列。

### 5.2.1 CNN

我们构造了一个包含多个卷积层，池化层，归一化层以及激活函数的卷积神经网络。这个卷积神经网络可以被划分为七个层次。

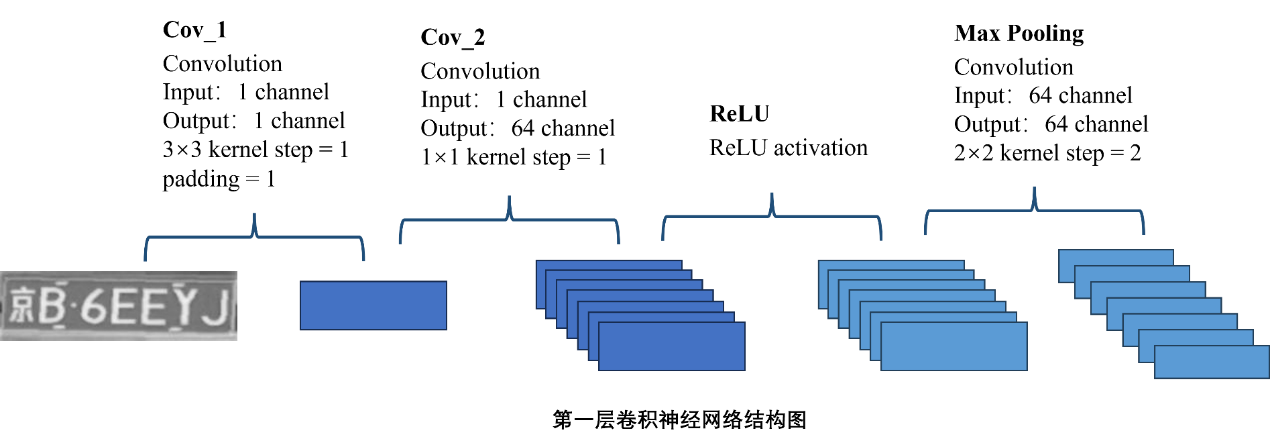
我们在读图片时将其的高度设定为32，宽度设定为200，并且以灰度图的方式读图片，使得其通道数为1。

此卷积神经网络的七层具体结构如下：

****

**图表 36 卷积神经网络七层具体结构图**

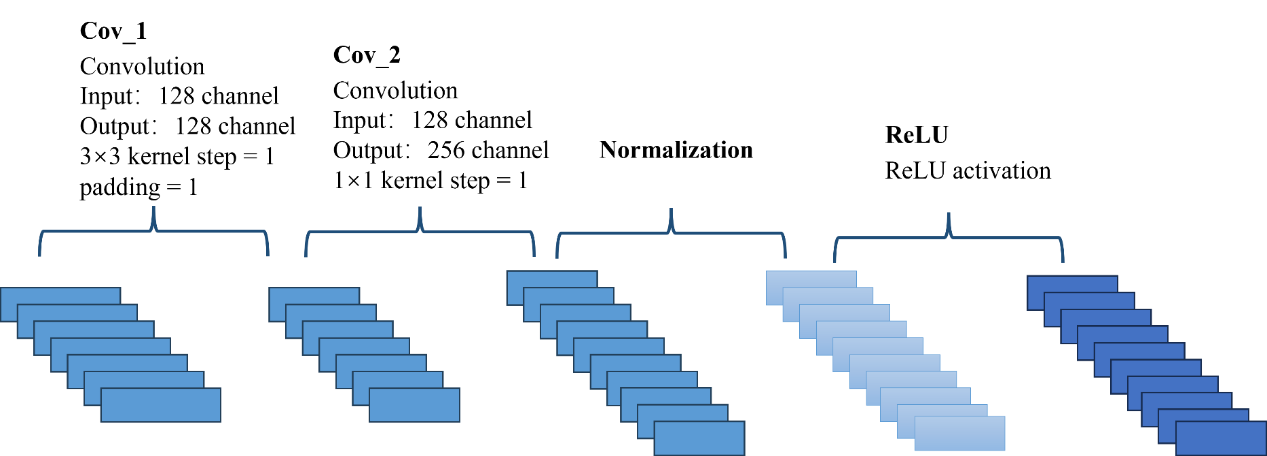
对于第1,2,4,6层，其结构如下（以第一层为例，其余层结构相同但通道数不同）：

****

**图表 37 第一层卷积神经网络结构图**

第一层为深度卷积层，主要用来提取空间信息，第二层为点卷积层，主要用来变换通道数，第三层为ReLU激活函数，第四层为最大池化层。

对于第3,5,7层，其结构如下（以第三层为例，其余层结构相同但通道数不同）：

****

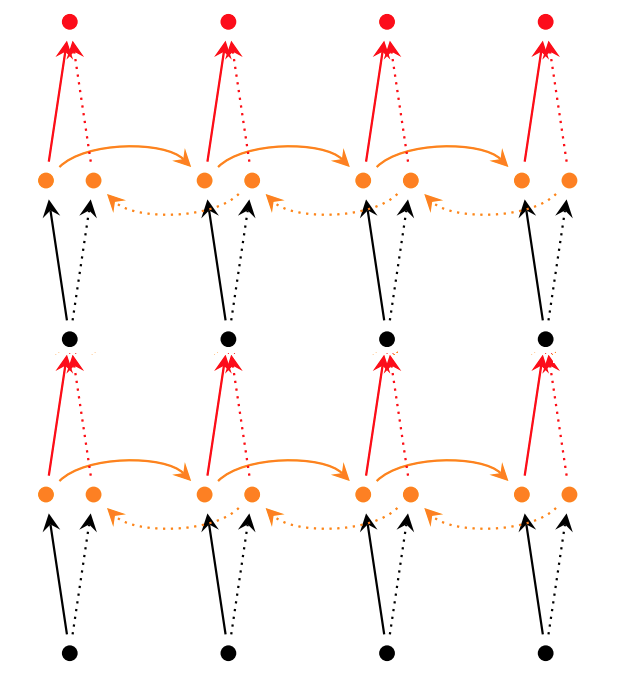
**图表 38 第二层卷积神经网络结构图**

其中，与1,2,4,6层的不同之处在于将最大池化层换成了批归一化层，使用此归一化层的目的主要是为了加速收敛，降低训练时间。

经过上述的CNN网络，得到的输出的维度为1×51×512，此输出将会作为输入进行后续BLSTM的训练。

### 5.2.2 RNN

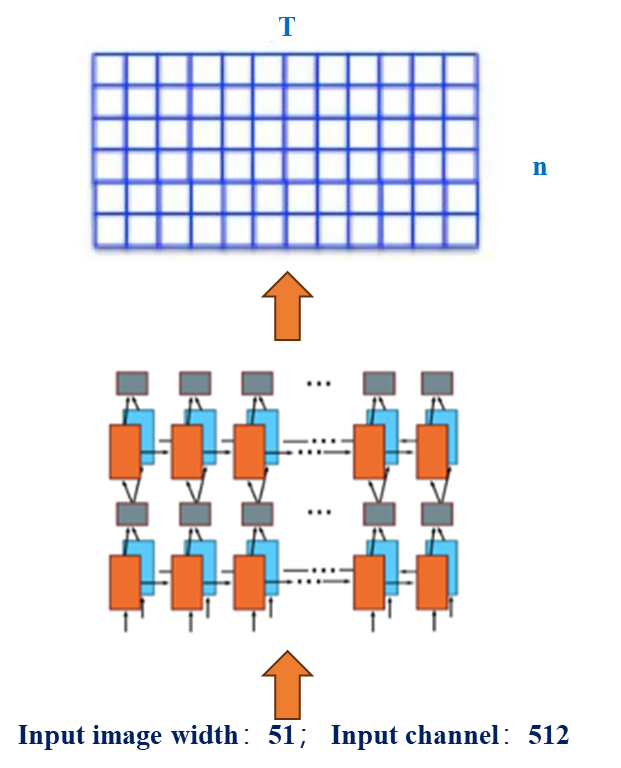
在循环层，我们使用深层双向LSTM网络，在卷积特征的基础上继续提取文字序列特征。网络结构如下图所示。

****

**图表 39 RNN网络结构图**

LSTM 能够捕获长距离依赖，而双向LSTM能够利用上下文两个方向的信息。我们将CNN层的输出做简单处理，由于高度为1，我们将此维度省略，然后将CNN的输出切分为51个，每一个有512维，至此，我们得到了RNN层的输入。输入宽度为51，即RNN的最大时间长度T=51，每一个输入有512维。

经过BLSTM中的一系列变换，最后 在经过Softmax之后，最终得到T×n维概率矩阵，其中n为类别数量，此处和MLP不同，我们不将汉字与数字字母分开，所以此处的n取值为66（数字和字母34个+省份31个+blank），其中，blank会在转录层解释作用。上述过程如下图所示：

****

**图表 40 RNN输入输出图**

### 5.2.3 CTC loss

CRNN使用CTCloss作为转录层，接收RNN得到的概率矩阵，寻找最大概率路径，并且对每个特征向量所做的预测转换成标签序列。

我们使用的CRNN是一种端对端不定长文本识别网络，也就是说，我们无法限制其输出长度，像我们使用的CRNN，由于RNN层的时间步长为51，所以理论上，输出的文本长度可以在1-51之间。此时，如果我们硬性要求输入序列和目标序列进行明确的对齐，这在实践中是非常困难的，有时甚至是不可能的。于是，使用CTC作为转录层就解决了这个问题。

CTC通过下面的策略来解决序列对齐问题：

多对一映射： CTC允许网络生成多个序列映射到同一目标序列。例如，对于目标序列“京BBB66EEEEEYYJJ”，CTC将会把同一连续字符合并输出，于是我们得到的结果是“京B6EYJ”。

引入空白标签：上面得到的输出显然是不对的，因为简单的多对一映射会在原标签序列就存在连续同一字符时出现错误，基于此情况，CTC引入blank字符，在此项目中，我们将其设为空格。在RNN层，我们提到有一个blank字符，它也作为分类中的一类，于是在预测时，如果模型经过良好的训练，那么它输出的序列应该为“京 BBB 66 EE EEE YY JJ”，于是，在进行多对一映射时，由于空格的存在，合并后的序列为“京B6EEYJ”，此时映射的序列是正确的。总而言之，blank字符的引入解决了连续同一字符的多对一映射问题。

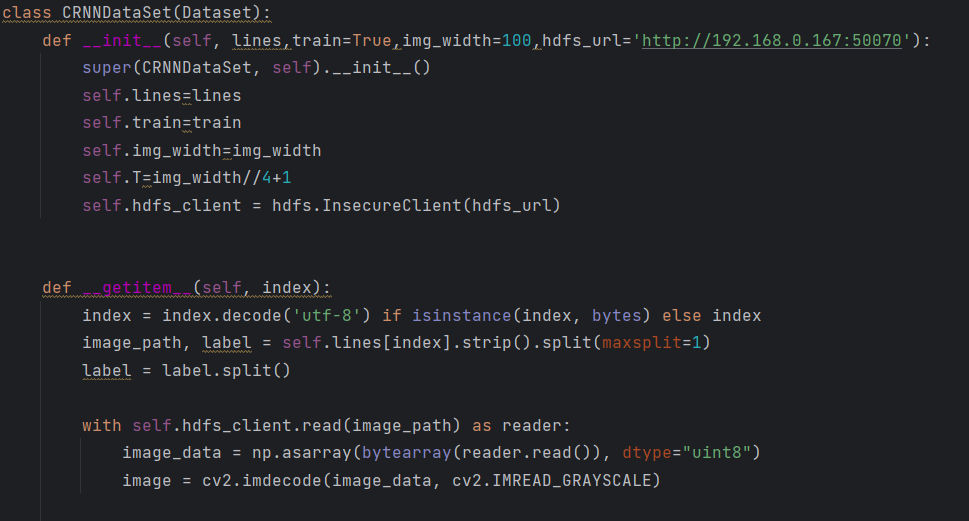
最终损失：CTC损失函数是通过对所有可能的对齐路径的概率的负对数似然进行求和来计算的。这可以使用一种称为前向-后向算法的动态规划技术来有效地计算。

使用CTC Loss训练网络时，不需要事先对序列进行对齐。在训练过程中，网络学习通过输入序列的概率生成正确的标签序列输出。而在预测过程中，CTC寻找最大概率路径并进行转录得到输出序列。

### 5.2.4 CRNN算法实现

为了实现CRNN模型，我们定义了CRNNDataset类，CRNN类，以及训练和验证代码。

CRNNDataset类（继承自Dataset）：用于加载和处理训练和测试的图片。

****

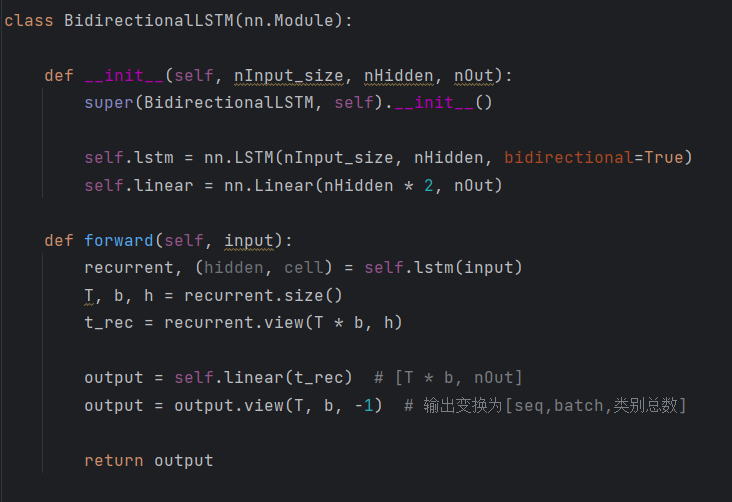
**图表 41 Dataset类部分代码截图**

图片上传HDFS文件系统，使用Python的hdfs库构建hdfs\_client读图片。CRNNDataset类同样完成了图片的归一化以及大小调整。

CRNN类：定义模型结构。包含CNN和BLSTM。

****

**图表 42 CNN类部分代码截图**

****

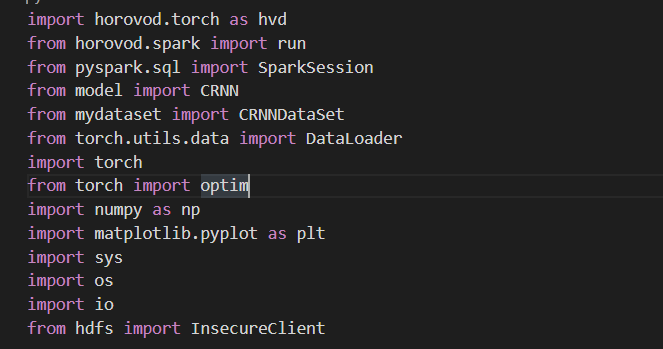
**图表 43 BLSTM类部分代码截图**

****

**图表 44 CRNN类部分代码截图**

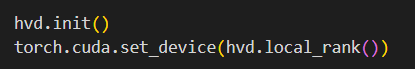
我们的训练代码是使用Pytorch深度学习框架和Horovod分布式深度学习框架编写的，此处我们仅对Horovod在分布式环境下对代码的相关修改做解释说明：

导入horovod.torch和horovod.spark：

****

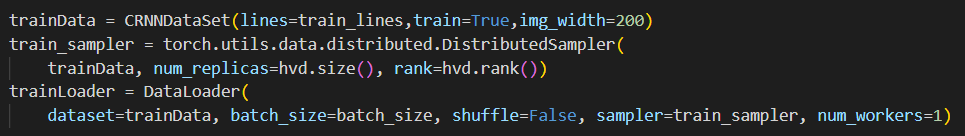
**图表 45 训练代码截图（1）**

初始化并实现GPU加速：

****

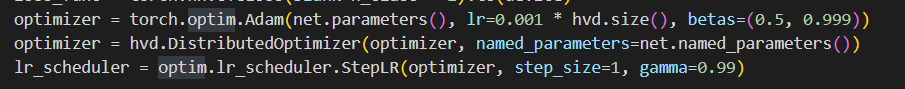
**图表 46 训练代码截图（2）**

在原本数据加载的基础上，我们需要创建分布式采样器，用于在分布式环境中对训练数据进行加载和划分：

****

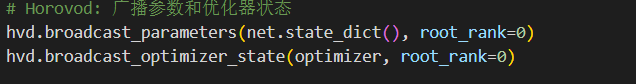
**图表 47 训练代码截图（3）**

由于是分布式训练，我们需要在训练过程中保证参数的同步性，所以创建分布式优化器，确保参数同步更新：

****

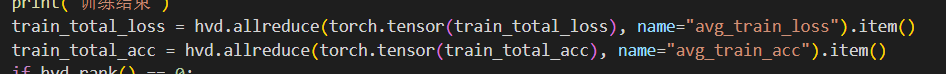
**图表 48 训练代码截图（4）**

广播参数和优化器状态，使得每一个节点都获得相同的参数：

****

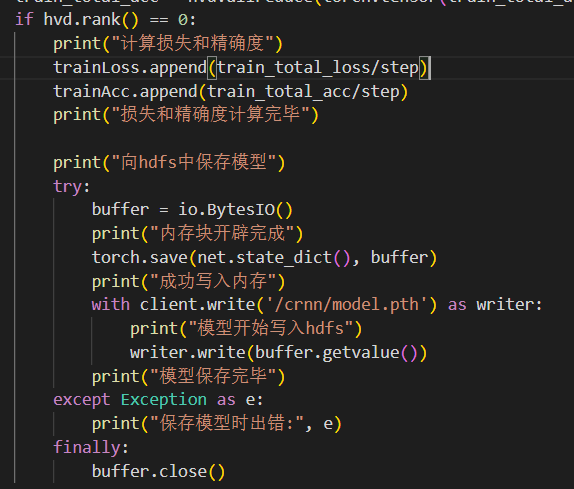
**图表 49 训练代码截图（5）**

全局规约损失值，确保结果一致：

****

**图表 50 训练代码截图（6）**

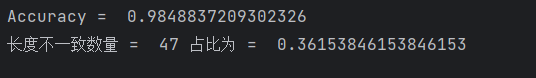
由于存在多个进程，我们只让其中一个进程保存模型，进程号为0，保存模型到hdfs文件系统：

****

**图表 51 训练代码截图（7）**

### 5.2.5 算法结果分析

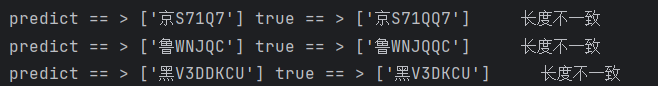
此模型在测试集上的准确率达到了98.4%，在8600张图片的测试集上仅有130张图片识别错误，相比于MLP有很大提升。

****

**图表 52 CRNN预测准确率图**

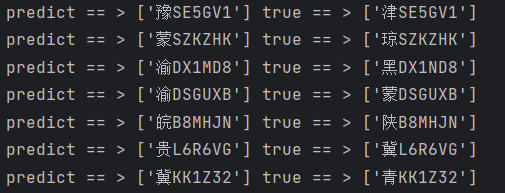
我们将结果中的错误分为两种错误类型：长度不一致导致的错误和长度一致但是预测错误，对于130张识别错误的图片，我们将其打印输出，统计长度一致和不一致的错误数量，得出预测长度不一致导致的错误的数量占比为36%。

对于长度不一致的错误，经过分析，我们发现大多数是因为相同的字母或数字连续出现而导致了预测数量不正确，这说明RNN得到的概率矩阵中，blank并没有分割开相同的连续字符，使得在CTC转录时并没有正确转录。

****

**图表 53 长度不一致错误截图**

对于长度一致的错误，我们发现预测错误往往出现在汉字，这说明汉字的分类确实难于数字和字母的分类。

****

**图表 54 长度一致错误截图**

经过对原图的查看，我们发现这些图片在数据预处理时裁切的车牌精细度较差，进而导致这些图片的预测难度确实较高。在这种情况下，优化数据预处理流程能降低此类型错误出现的概率。

****

**图表 55 出错原图（1）**

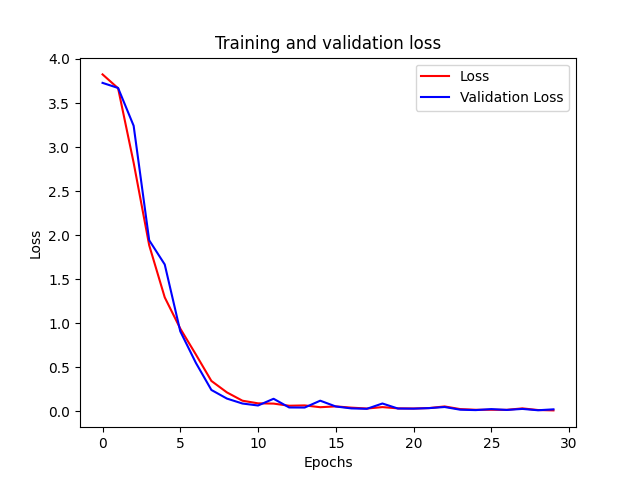
****

**图表 56 出错原图（2）**

# 6 实验结果可视化

## 6.1 损失函数曲线与准确率曲线

我们首先绘制了CRNN模型训练过程中的损失和准确率，得到如下图片。其中CRNN训练过程损失函数曲线如下：

****

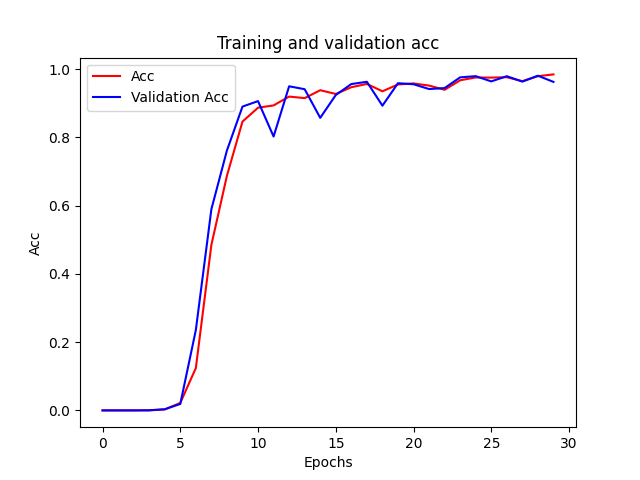
**图表 57 CRNN训练过程损失函数曲线图**

红色曲线为训练损失，蓝色曲线为验证损失

在最初的几个epochs中，两条曲线都迅速下降，这表明模型在这个阶段学习效率很高，损失快速减少。随着epochs的增加，损失下降的速度放缓，两条曲线开始趋于平稳。模型开始收敛。

从曲线的趋势可以看出，模型没有出现过拟合的迹象。过拟合通常会表现为训练损失持续下降，而验证损失在某点后开始上升。在这张图中，两条曲线都呈下降趋势，且趋势相似，这表明模型在训练数据和验证数据上都有好的表现，并且具有较好的泛化能力。

CRNN训练过程准确率曲线如下：

****

**图表 58 准确率曲线图**

红色曲线为训练准确率，蓝色曲线为验证准确率。

在训练的初始阶段，准确率急剧上升，表明模型在学习过程中迅速改善其性能。经过几个epoch后，两条曲线趋于平稳，增长缓慢，且准确率大于0.9，证明模型性能很好。训练准确率和验证准确率在高水平上非常接近并且几乎重叠，两条曲线都呈上升趋势，且趋势相似，这表明模型在训练数据和验证数据上都有好的表现，并且具有较好的泛化能力。

## 6.2 GUI设计

使用tkinter库创建的简单的车牌识别应用程序。它包括一个用户界面，用户可以从文件系统中选择要识别的图像。选择图像后，程序使用预训练的模型来识别车牌，并在界面上显示车牌位置和识别结果。

****

**图表 59 车牌识别应用程序图**

此界面允许用户选择一张图片，并显示该图片上的车牌定位与识别结果。在实现代码中定义了一个名为Surface的类继承自ttk.Frame，该类构建了应用程序的界面，包括选择图片的按钮、显示原图的标签和展示车牌识别结果的区域。在用户选择图片之后，程序利用getPlate模块处理图片并定位车牌，接着使用predict模块读取车牌区域的图像并对其内容进行识别，最终将识别的结果展示在GUI界面中。

# 7 课程总结

经过一学期的课程学习以及最后课程项目的实践，我们小组收获颇丰。

首先，我们学习了大数据的相关原理和概念，熟悉了Hadoop + Spark下的编程环境，了解了Map-Reduce的原理，了解了HBase数据库，掌握了基本大数据可视化工具，掌握了HDFS 分布式文件系统基本操作，掌握了基本的机器学习和大数据算法。

其次，我们基于具体的应用场景构建了一个大数据项目。将理论和实践结合起来，在实践过程中，我们更好地理解理论知识，对于大数据有了更为深入的了解，掌握了更多的相关技术。

最后，我们的心态也得到了锻炼。在准备课程项目的过程中，我们遇到了很多环境上或者代码上的困难，尤其是在部署分布式深度学习环境时，某一个软件的版本不兼容就会导致一系列的问题，我们不断查看官方文档和相关教程，解决遇到的一个个困难，在此过程中提升了解决问题的能力，也锻炼了心态。

总而言之，经过本课程的学习，我们对大数据知识和技术有了更深入的了解，同时，我们也提高了自己的实践能力。本课程让我们收获满满。