# 实验五 车牌识别

小组成员：

蒲尧2020E8017782032，王国帅2020E8017782034，吴文霞 2020E8017782060

## 一、实验目的

1. 掌握卷积神经网络中卷积层、卷积步长、卷积核、池化层、交叉熵损失等概念。
2. 进一步熟悉深度学习框架的使用，包括利用框架构造基本的卷积神经网络模型，并结合收集到的相关数据实现模型的训练和测试。

## 二、实验要求

1. 利用Python语言和深度学习框架（我们小组采用TensorFlow框架），构造简单的卷积神经网络模型，以实现中国普通机动车车牌字符的识别。
2. 车牌识别的准确率不低于85%。
3. 如果选择做本实验的话，按规定时间在课程网站提交实验报告、代码以及PPT。

## 三、实验原理

卷积神经网络相关概念参考课程《深度学习》课上讲授及课件内容。为简化问题，本实验仅考虑最简单的中国普通机动车车牌的识别，车牌示意如图1所示，可以将车牌划分为**三个区域**，如下：

（1）省、自治区、直辖市简称

（2）发牌机关代号

（3）车牌序号

为简化实验，在该实验中默认车牌字符已经得到划分，因此车牌识别可以分解为三个区域的字符识别任务（多分类任务），共实现**7个字符**的识别。



图1 车牌示意图

## 四、实验所用工具及数据集

本次实验我们采用了两个数据集：第一个是老师课程网站老师提供的数据集，可以利用该数据集，实现车牌的图片分割、图像校准、图像按单个字符切成块等任务。该数据集划分如下：

LPD\_dataset

|—train

|—via\_region\_data.json

|—京A88731.jpg

|—…

|—黑AB4444.jpg

|—val

|— via\_region\_data.json

|—浙A03168.jpg

|—…

|—粤X30479.jpg

json文件中包含有车牌的定位信息。

第二个数据集是来自GitHub相关开源项目提供的数据，以及CSDN相关博客所采用的部分数据集。该数据集的目录划分如下：

dataset-train&val

|— training-set

|— province

|— area

|— letter

|— validation-set

|— province

|— area

|— letter

因为数据集数据量不是很大，因此仅将其划分为训练集train和验证集val，均包含province、area和letter三个目录，且每个目录下按照其所包含的字符划分目录存储对应字符的灰度图，大小为~~20\*20~~ 32\*40。三个区域理论上包含的字符如下部分，各31,26,34类，但是由于网上收集到的数据有限，故省份类别只有("沪", "京", "闽", "苏", "粤", "浙")这6类。

Province：

("皖", "沪", "津", "渝", "冀", "晋", "蒙", "辽", "吉", "黑", "苏", "浙", "京", "闽", "赣", "鲁", "豫", "鄂", "湘", "粤", "桂", "琼", "川", "贵", "云", "藏", "陕", "甘", "青", "宁", "新")

Area：("A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z")

Letter：

("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")

## 五、实验步骤与方法

对车牌三个区域进行字符识别分类的卷积神经网络模型可以自行设计，本实验指导书给出一个简单的参考模型（三个区域识别分类采用同一模型结构，利用对应的数据分别进行训练），模型结构如图2所示。我们根据实际实验数据和测试结果进行了简单的修改，最终网络结构如图3所示。



图2 参考模型结构图

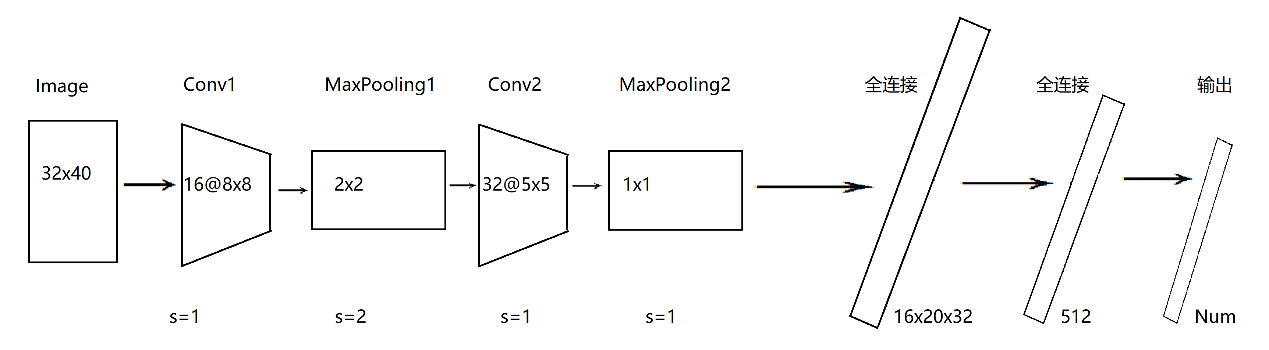


图3 实际模型结构图

示例模型结构可以分为4层：输入为32\*40的二值图像，第一层为卷积层，包含16个卷积核，尺寸为8\*8，卷积步长为1，后接一个尺寸为2\*2，步长为2的最大池化；第二层为卷积层，包含32个卷积核，尺寸为5\*5，卷积步长为1，后接一个尺寸为1\*1，步长为1的最大池化（本质数据未产生变化，为了迎合自行编写的卷积层函数，见模型构建部分）；然后将第二层输出扁平化为16\*20\*32，第三层为全连接层，将图像特征映射到512维向量；第四层为全连接Softmax分类层，Num分别为三个区域包含类别数。

实验可简单地划分为数据准备、模型构建和模型训练三个步骤。

（1）数据准备

我们先根据json文件的’ regions’定位字段，将每幅图像的车牌位置进行切分出来并且矫正，可以使用如下代码实现：

**import** json

**import** cv2

**import** os

**from** PIL **import** Image

**import** numpy **as** np

**from** imutils**.**perspective **import** four\_point\_transform

**with** **open(**'./LPD\_dataset/train/via\_region\_data.json'**,** 'r'**,** encoding**=**'utf-8'**)** **as** load\_f**:**

load\_dict\_train **=** json**.**load**(**load\_f**)**

**print(len(**load\_dict\_train**))**

load\_dict\_train\_key **=** **list(**load\_dict\_train**.**keys**())**

**for** i **in** **range(len(**load\_dict\_train\_key**)):**

filename **=** load\_dict\_train**[**load\_dict\_train\_key**[**i**]][**'filename'**]**

**print(**filename**)**

points **=** **[[**load\_dict\_train**[**load\_dict\_train\_key**[**i**]][**'regions'**][**0**][**'shape\_attributes'**][**'all\_points\_x'**][**j**],**load\_dict\_train**[**load\_dict\_train\_key**[**i**]][**'regions'**][**0**][**'shape\_attributes'**][**'all\_points\_y'**][**j**]]** **for** j **in** **range(**4**)]**

**print(**points**)**

img\_path **=** './LPD\_dataset/train/' **+** filename

img **=** Image**.open(**img\_path**)**

# 4个角点矫正

rect **=** four\_point\_transform**(**cv2**.**cvtColor**(**

np**.**asarray**(**img**),** cv2**.**COLOR\_RGB2BGR**),** np**.**array**(**points**))**

rect\_resize **=** cv2**.**resize**(**rect**,** **(**440**,** 140**))**

# 3通道变1通道灰度图

gray **=** cv2**.**cvtColor**(**rect\_resize**,** cv2**.**COLOR\_BGR2GRAY**)**

gray\_Img **=** Image**.**fromarray**(**cv2**.**cvtColor**(**gray**,** cv2**.**COLOR\_BGR2RGB**))**

# 将一个车牌划分为7个部分

**for** i **in** **range(**7**):**

patch**[**i**]** **=** gray\_Img**.**crop**((**x**[**i**],** y**[**i**],** x**[**i**]+**width**,** y**[**i**]+**height**))**

效果图如下：



图4 车牌校准



图5 字符切分

（2）模型构建

参考模型结构图对模型进行构建，TensorFlow框架下，卷积神经网络中常用的有tf.nn.conv2d、tf.nn.relu、tf.nn.max\_pool等函数。

具体代码如下：

（i）卷积层函数和全连接层函数

# 定义卷积函数

**def** conv\_layer**(**inputs**,** W**,** b**,** conv\_strides**,** kernel\_size**,** pool\_strides**,** padding**):**

L1\_conv **=** tf**.**nn**.**conv2d**(**inputs**,** W**,** strides**=**conv\_strides**,** padding**=**padding**)**

L1\_relu **=** tf**.**nn**.**relu**(**L1\_conv **+** b**)**

**return** tf**.**nn**.**max\_pool**(**L1\_relu**,** ksize**=**kernel\_size**,** strides**=**pool\_strides**,** padding**=**'SAME'**)**

# 定义全连接层函数

**def** full\_connect**(**inputs**,** W**,** b**):**

**return** tf**.**nn**.**relu**(**tf**.**matmul**(**inputs**,** W**)** **+** b**)**

（ii）模型构建代码

with tf.Session() as sess:

# 第一个卷积层

W\_conv1 = tf.Variable(tf.truncated\_normal([8, 8, 1, 16], stddev=0.1), name="W\_conv1")

b\_conv1 = tf.Variable(tf.constant(0.1, shape=[16]), name="b\_conv1")

conv\_strides = [1, 1, 1, 1]

kernel\_size = [1, 2, 2, 1]

pool\_strides = [1, 2, 2, 1]

L1\_pool = conv\_layer(x\_image, W\_conv1, b\_conv1, conv\_strides, kernel\_size, pool\_strides, padding='SAME')

# 第二个卷积层

W\_conv2 = tf.Variable(tf.truncated\_normal([5, 5, 16, 32], stddev=0.1), name="W\_conv2")

b\_conv2 = tf.Variable(tf.constant(0.1, shape=[32]), name="b\_conv2")

conv\_strides = [1, 1, 1, 1]

kernel\_size = [1, 1, 1, 1]

pool\_strides = [1, 1, 1, 1]

L2\_pool = conv\_layer(L1\_pool, W\_conv2, b\_conv2, conv\_strides, kernel\_size, pool\_strides, padding='SAME')

# 全连接层

W\_fc1 = tf.Variable(tf.truncated\_normal([16 \* 20 \* 32, 512], stddev=0.1), name="W\_fc1")

b\_fc1 = tf.Variable(tf.constant(0.1, shape=[512]), name="b\_fc1")

h\_pool2\_flat = tf.reshape(L2\_pool, [-1, 16 \* 20 \* 32])

h\_fc1 = full\_connect(h\_pool2\_flat, W\_fc1, b\_fc1)

# dropout层

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

# 输出层

W\_fc2 = tf.Variable(tf.truncated\_normal([512, NUM\_CLASSES], stddev=0.1), name="W\_fc2")

b\_fc2 = tf.Variable(tf.constant(0.1, shape=[NUM\_CLASSES]), name="b\_fc2")

y\_conv = tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2

# 定义优化器和训练op

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y\_conv))

train\_step = tf.train.AdamOptimizer((1e-4)).minimize(cross\_entropy)

模型采用分类任务常用的交叉熵损失；训练优化其采用Adam，初始学习率设置为1e-4，进行实验。

（3）模型训练

模型的训练过程则是一个循环过程，每一步循环准备训练的数据并送入模型。为了检测模型训练过程，我们可以每间隔一段时间步，将验证集数据送入模型中，获取模型在验证集上的准确率并输出。代码如下：

# training...

**for** it **in** **range(**iterations**):**

**for** n **in** **range(**batches\_count**):**

train\_step**.**run**(**feed\_dict**={**x**:** input\_images**[**n**\***batch\_size**:(**n**+**1**)\***batch\_size**],**

y\_**:** input\_labels**[**n**\***batch\_size**:(**n**+**1**)\***batch\_size**],** keep\_prob**:** 0.5**})**

**if** remainder **>** 0**:**

start\_index **=** batches\_count **\*** batch\_size

train\_step**.**run**(**feed\_dict**={**x**:** input\_images**[**start\_index**:**input\_count**-**1**],**

y\_**:** input\_labels**[**start\_index**:**input\_count**-**1**],** keep\_prob**:** 0.5**})**

# check acc >= 99.99% or it >= iterations per 10 times

iterate\_accuracy **=** 0

**if** it **%** 10 **==** 0**:**

iterate\_accuracy **=** accuracy**.eval(**

feed\_dict**={**x**:** val\_images**,** y\_**:** val\_labels**,** keep\_prob**:** 1.0**})**

**print(**'第 %d 次训练迭代: 准确率 %0.5f%%' **%** **(**it**,** iterate\_accuracy**\***100**))**

**if** iterate\_accuracy **>=** 0.9999 **or** it **>=** iterations**:**

**break**

## 六、示例代码说明及实验结果

dataset-train&val目录下为该实验使用到的数据集； train-license-province.py、train-license-area.py、train-license-letter.py分别为车牌三个区域的模型构建、训练和测试代码，三个文件代码有很大的重复。测试数据在test\_images文件夹里如图12，分别为1.bmp~16.bmp。其中1~3为3个province字段单个字块，4~6为3个area字段单个字块，7~11为1个车牌的letter的5个字块，12~16为另外1个车牌的letter的5个字块。

模型训练命令：

python train-license-\*.py train

e.g：python train-license-province.py train

模型测试命令：

python train-license-\*.py predict

e.g. python train-license-province.py predict

上面两个命令输出结果可见下面几页图6~图11，province，area，letter字段准确率分别能达到**100%，100%，98.5%**。具体可见log.txt文件。

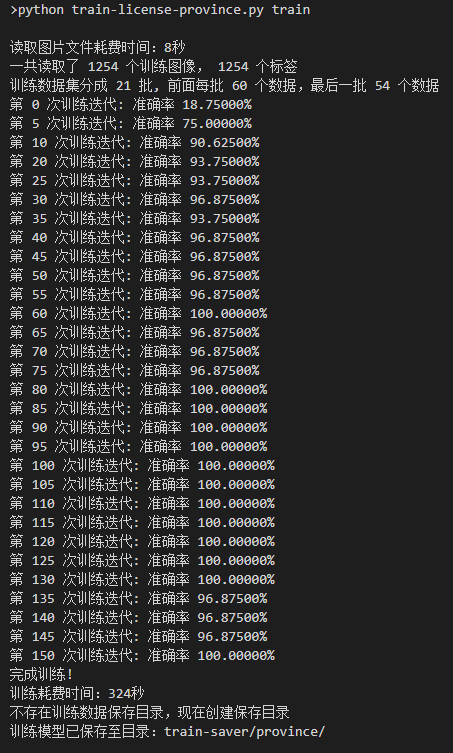


图6 province字段训练输出

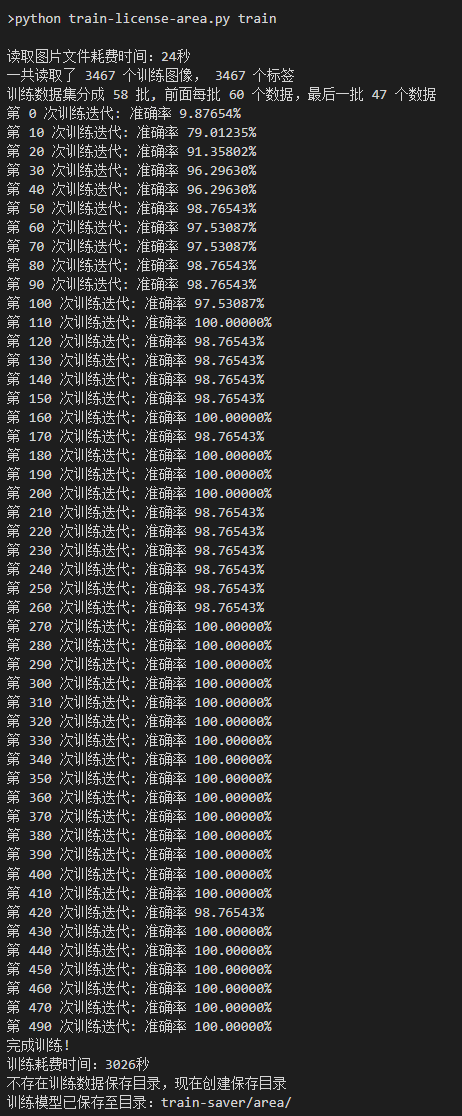


图7 area字段训练输出

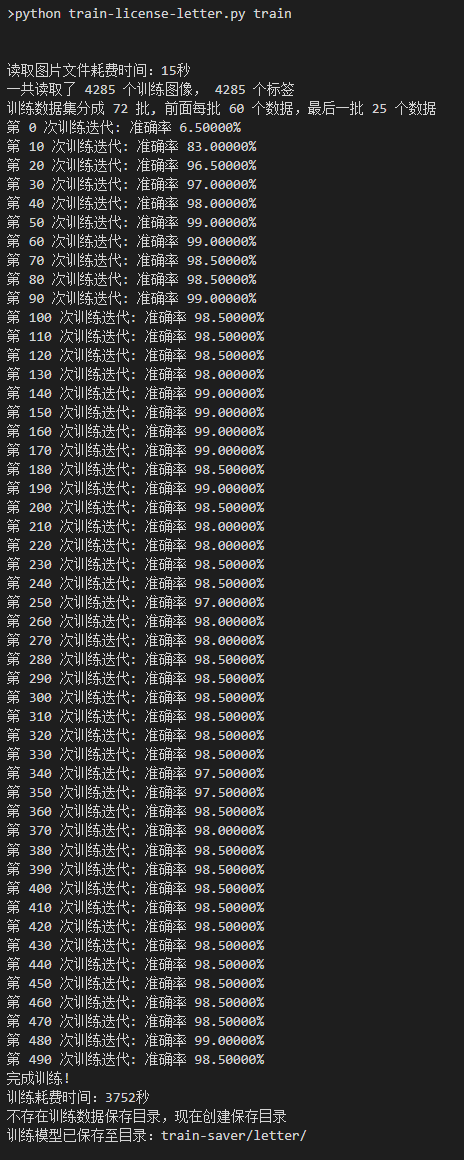


图8 letter字段训练输出

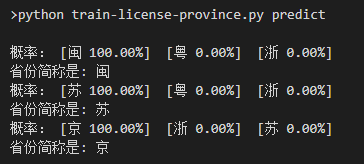


图9 province字段预测

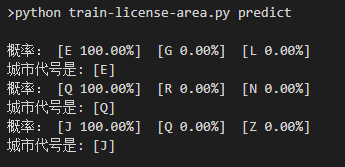


图10 area字段预测

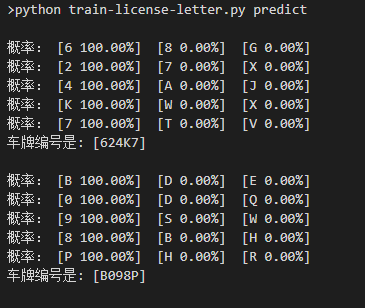


图11 letter字段预测



图12 测试字块