**实验二 车牌识别**

1. 实验目的
2. 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。
3. 初步熟悉TensorFlow框架的使用，包括利用TensorFlow 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。
4. 实验要求

利用Python语言和TensorFlow框架（可结合Keras），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。

1. 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

1. 实验所用工具及数据集

该实验采用的数据集来自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。数据集可在课程网站中进行下载，数据集的目录划分如下：

dataset

|— train

|— province

|— area

|— letter

|— val

|— 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。三个区域包含的字符如下。

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")

1. 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。



图2 模型结构图

示例模型结构可以分为4层：输入为20\*20的二值图像，第一层为卷积层，包含16个卷积核，尺寸为3\*3，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3\*3，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，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

因为数据集为一系列20\*20的灰度图像，我们需要将图像读入并转换为TensorFlow能够进行处理的数据。

可以使用OpenCV库或者PIL库遍历数据集对应目录，打开图像，示例代码提供了利用PIL库处理数据集的操作，具体请参考提供的示例代码。

（2）模型构建

参考模型结构图对模型进行构建，TensorFlow具体函数的使用说明请自行查询官方文档或相关教程资料，卷积神经网络中常用的有tf.nn.conv2d、tf.nn.relu、tf.nn.max\_pool等函数。

示例代码中基于TensorFlow提供的API函数自行编写了简单的卷积层和全连接层的函数，如图3所示。卷积层函数需要提供卷积层权重、偏置、卷积步长、最大池化核大小及最大池化步长参数，其中卷积层权重、偏置为训练更新参数；全连接层需要提供权重矩阵和偏置参数。

# 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**)**

图3 卷积层函数和全连接层函数

**with** tf**.**Session**()** **as** sess**:**

# 第一个卷积层

W\_conv1 **=** tf**.**Variable**(**tf**.**truncated\_normal**([**3**,** 3**,** 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**([**3**,** 3**,** 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**([**10 **\*** 10 **\*** 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**,** 10 **\*** 10 **\*** 32**])**

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

# dropout

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

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

# readout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"**)**

# 定义优化器和训练op

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

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**)**

图4 模型构建代码

模型的构建代码如图4所示，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。也可自行选择其他优化器进行实验。

（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。具体实现请参考示例代码。

1. 示例代码说明：

dataset目录下为该实验使用到的数据集，也可以自行收集相关图像制作数据集；SimpleNetwork.py利用TensorFlo框架定义了简单卷积层和全连接层函数；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，也可以将其整合为一个python代码文件完成三个模型的训练。

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 “测试图像路径”

e.g. python train-license-province.py predict “./test.jpg”