**第三次竞赛**

**过程一：**首先使用k近邻法，即将每个验证码图片分割成5个小部分，每个小部分分别与62个数字和大小写字母进行比较，与哪个最相似就是哪个，但是需要获取足够数量的62个数字和大小写字母，耗费太长时间，而且效果也不是很好，所以不使用k近邻法；

**过程二：**通过搜集网上识别验证码的资料，确定使用卷积神经网络来进行识别；

**过程三：**在用train进行训练的时候，由于数据量太大，经过多次运行还是无法得到结果，所以先对train数据进行了削减，再传到卷积神经网络中进行训练。

1. 建立一个字典，将大小写字母转成数字：

import string

label\_dict={}

characters = string.digits + string.ascii\_letters

for i,x in enumerate(characters):

label\_dict[x]=i

2. 对验证码图片进行处理，进行灰度转化、二值化、高斯滤波后再次进行二值化，最终得到需要的输入数据：

imagepath='train/'

def get\_x\_data(filepath):

data=[]

image\_name=[]

for filename in os.listdir(filepath):

image\_name.append(filename)

im=cv2.imread(filepath+filename)

#灰度

im\_gray=cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

#二值化

ret, im\_inv= cv2.threshold(im\_gray,127,255,cv2.THRESH\_BINARY\_INV)

#高斯滤波去除颗粒噪声

kernel = 1/16\*np.array([[1,2,1], [2,4,2], [1,2,1]])

im\_blur = cv2.filter2D(im\_inv,-1,kernel)

#再次进行二值化

ret, im\_res = cv2.threshold(im\_blur,127,255,cv2.THRESH\_BINARY)

im\_data=im\_res.reshape(1,30\*150)/255

image\_data=im\_data.tolist()

data.append(image\_data)

return data,image\_name

data,image\_name=get\_x\_data(imagepath)

3. label\_data=get\_y\_data(image\_name)调用get\_y\_data函数，获取验证码的标签：

def get\_y\_data(imagename):

label\_data=[]

for x in imagename:

labels=np.zeros([5,62])

result1,result2=x.split('.')

for i,chara in enumerate(result1):

col\_index=label\_dict[chara]

labels[i,col\_index]=1

label\_data.append(labels.reshape(1,-1).tolist())

return label\_data

4. 将整个数据集以1:5的比例划分成训练集和测试集

train\_x,test\_x,train\_y,test\_y=train\_test\_split(new\_data,new\_label,test\_size=0.2)

5. 搭建输入层—>隐藏层1—>隐藏层2—>隐藏层3—>全连接层—>输出层的神经网络结构：

卷积函数

def conv2d(x,w):

return tf.nn.conv2d(x,w,strides=[1,1,1,1],padding='SAME',name='conv2d')

池化函数

def max\_pool(x):

return tf.nn.max\_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME',name='maxpool')

权重生成函数

def weight\_variable(shape):

return tf.Variable(tf.truncated\_normal(shape,stddev=0.1))

偏置生成函数

def bias\_variable(shape):

return tf.Variable(tf.constant(0.1,shape=shape))redictResult = list(predictResult) 将预测结果转成list，

m, = np.shape(predictResult)

第一层卷积，卷积核为5\*5

w\_conv1=weight\_variable([5,5,1,32])

b\_conv1=bias\_variable([32])

x\_image=tf.reshape(xs,[-1,30,150,1],name='x-input')

h\_conv1=tf.nn.relu(conv2d(x\_image,w\_conv1)+b\_conv1)

pool\_1=max\_pool(h\_conv1)

pool\_1=tf.nn.dropout(pool\_1,keep\_prob)

第二层卷积

w\_conv2=weight\_variable([5,5,32,64])

b\_conv2=bias\_variable([64])

h\_conv2=tf.nn.relu(conv2d(pool\_1,w\_conv2)+b\_conv2)

pool\_2=max\_pool(h\_conv2)

pool\_2=tf.nn.dropout(pool\_2,keep\_prob)

第三层卷积

w\_conv3=weight\_variable([5,5,64,64])

b\_conv3=bias\_variable([64])

h\_conv3=tf.nn.relu(conv2d(pool\_2,w\_conv3)+b\_conv3)

pool\_3=max\_pool(h\_conv3)

pool\_3=tf.nn.dropout(pool\_3,keep\_prob)

全连接层

w\_fc\_1=weight\_variable([4\*19\*64,1024])

b\_fc\_1=bias\_variable([1024])

pool\_2\_flat=tf.reshape(pool\_3,[-1,4\*19\*64])

h\_fc\_1=tf.nn.relu(tf.matmul(pool\_2\_flat,w\_fc\_1)+b\_fc\_1)

h\_fc\_prob=tf.nn.dropout(h\_fc\_1,keep\_prob)

输出层

w\_fc\_2=weight\_variable([1024,62\*5])

b\_fc\_2=bias\_variable([62\*5])

output = tf.add(tf.matmul(h\_fc\_prob, w\_fc\_2),b\_fc\_2)

6. 设置损失函数以及优化器使得神经网络可以反馈调节

定义损失以及优化器：

loss=tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=ys, logits=output))

optimizer = tf.train.AdamOptimizer(0.001).minimize(loss)

predict = tf.reshape(output, [-1, 5, 62], name='predict')

labels = tf.reshape(ys, [-1, 5, 62], name='labels')

计算准确率：

predict\_max\_idx = tf.argmax(predict, axis=2, name='predict\_max\_idx')

labels\_max\_idx = tf.argmax(labels, axis=2, name='labels\_max\_idx')

predict\_correct\_vec = tf.equal(predict\_max\_idx, labels\_max\_idx)

accuracy = tf.reduce\_mean(tf.cast(predict\_correct\_vec, tf.float32))

7. 训练模型，每次把所有的图片填到神经网络后，计算一次神经网络在测试集上的准确率，如果大于0.5，就终止训练，保存参数：

path = 'model/'

saver=tf.train.Saver()

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for step in range(1000):

index=get\_next\_batch(train\_x,64)

order=np.random.permutation(train\_x.shape[0])

for i in range(len(index)-1):

\_,loss\_=sess.run([optimizer,loss],feed\_dict={xs:train\_x[order[index[i]:index[i+1]],:],ys:train\_y[order[index[i]:index[i+1]],:],keep\_prob:0.75})

acc=sess.run(accuracy,feed\_dict={xs:test\_x,ys:test\_y,keep\_prob:1.0})

print('step=%d,iter=%d,accuracy=%f'%(step,i,acc))

if acc>0.5:

saver.save(sess,path+'digit\_captcha.model',global\_step=step)

break

8. 读取并处理test中的验证码图片，并对test中的验证码图片你进行验证：

validpath='test/'

validata,validname=get\_x\_data(validpath)

m,k,n=np.array(validata).shape

valid\_data=np.array(validata).reshape(m,n)

labeldata=get\_y\_data(validname)

m,k,n=np.array(labeldata).shape

label\_data=np.array(labeldata).reshape(m,n)

with tf.Session() as sess:

saver.restore(sess,tf.train.latest\_checkpoint(path))

predict=sess.run(predict\_max\_idx,feed\_dict={input\_holder:valid\_data,keep\_prob\_holder:1.0})

labels= sess.run(labels\_max\_idx,feed\_dict={label\_holder:label\_data})

9.将预测验证码的结果中有的数字，转成对应的大小写字母：

digit\_character = dict(zip(label\_dict.values(), label\_dict.keys()))

predcharacter=[]

for i in range(len(predict)):

yzm=''.join(str(digit\_character[x]) for x in predict[i])

predcharacter.append(yzm)

10. 将得到的list结果转成DataFrame，并保存到csv文件中：

m, = np.shape(predcharacter)

data = pd.DataFrame(predcharacter, index=range(0, m))

data.to\_csv('train\_labels.csv')