第三次竞赛说明文档

验证码识别步骤：

首先要将图片转化为灰度图片，然后转化为二值图像，进行降噪处理，之后将图片进行分割，然后对每个小图片进行识别，之后训练，测试。

这次竞赛中有很多问题：

1. imageslist = os.listdir('train1') #返回指定的文件夹包含的文件或文件夹的名字的列表，首先，要获取训练集中的图片的名字的列表
2. 之后，要将图片的扩展名去掉，我是通过循环去掉的。
3. 将图片进行分割，我发现图片都是150\*30的，而且数字字母分布均匀，所以，我就将图片横向平均分成五等分
4. 之后对每一个小图片进行识别训练。
5. 我在网上查阅资料得知，可以使用pytesseract识别验证码

第一次尝试过程：

1. 首先读取图片，img=Image.open(‘train’)
2. 之后对图片进行灰度化处理，img1 = img.convert(‘L’)
3. 之后是二值化处理，首先获取图片中像素点数量最多的像素，即背景像素，之后将该像素作为阈值，进行二值化处理
4. 接下来是去噪声点，

第一次尝试：

# -\*- coding: utf-8 -\*-

"""

Created on Tue Nov 19 09:31:39 2019

@author: DELL

"""

import os

import pytesseract

import matplotlib.pyplot as plt

from PIL import Image

from collections import defaultdict

import pandas as pd

# tesseract.exe所在的文件路径

pytesseract.pytesseract.tesseract\_cmd = 'E:\\tesseract\Tesseract-OCR\\tesseract.exe'

# 获取图片中像素点数量最多的像素

def get\_threshold(image):

pixel\_dict = defaultdict(int)

# 像素及该像素出现次数的字典

rows, cols = image.size

for i in range(rows):

for j in range(cols):

pixel = image.getpixel((i, j))

pixel\_dict[pixel] += 1

count\_max = max(pixel\_dict.values()) # 获取像素出现出多的次数

pixel\_dict\_reverse = {v:k for k,v in pixel\_dict.items()}

threshold = pixel\_dict\_reverse[count\_max] # 获取出现次数最多的像素点

# =============================================================================

# for i in range(rows):

# for j in range(cols):

# pixel = image.getpixel((i, j))

# if pixel == threshold:

# pixel\_dict[pixel] = 255

# =============================================================================

# =============================================================================

# print('threshold')

# print(threshold)

# =============================================================================

return threshold

# 按照阈值进行二值化处理

# threshold: 像素阈值

def get\_bin\_table(threshold):

# 获取灰度转二值的映射table

table = []

for i in range(256):

rate = 0.1 # 在threshold的适当范围内进行处理

#if i < 140:

if threshold\*(1-rate)<= i <= threshold\*(1+rate):

table.append(1)

else:

table.append(0)

return table

# 去掉二值化处理后的图片中的噪声点

def cut\_noise(image):

rows, cols = image.size # 图片的宽度和高度

change\_pos = [] # 记录噪声点位置

# 遍历图片中的每个点，除掉边缘

for i in range(1, rows-1):

for j in range(1, cols-1):

# pixel\_set用来记录该点附近的黑色像素的数量

pixel\_set = []

# 取该点的邻域为以该点为中心的九宫格

for m in range(i-1, i+2):

for n in range(j-1, j+2):

if image.getpixel((m, n)) != 1: # 1为白色,0位黑色

pixel\_set.append(image.getpixel((m, n)))

# 如果该位置的九宫内的黑色数量小于等于4，则判断为噪声

if len(pixel\_set) <= 4:

change\_pos.append((i,j))

# 对相应位置进行像素修改，将噪声处的像素置为1（白色）

for pos in change\_pos:

image.putpixel(pos, 1)

return image # 返回修改后的图片

# 识别图片中的数字加字母

# 传入参数为图片路径，返回结果为：识别结果

def OCR\_lmj(img\_path):

image = Image.open(img\_path) # 打开图片文件

# =============================================================================

# plt.figure(1)

# plt.imshow(image)

# =============================================================================

imgry = image.convert('L') # 转化为灰度图

# =============================================================================

# plt.figure(2)

# plt.imshow(imgry)

# =============================================================================

# 获取图片中的出现次数最多的像素，即为该图片的背景

max\_pixel = get\_threshold(imgry)

# 将图片进行二值化处理

table = get\_bin\_table(threshold=max\_pixel)

out = imgry.point(table, '1')

# =============================================================================

# plt.figure(3)

# plt.imshow(out)

# =============================================================================

# 去掉图片中的噪声（孤立点）

out = cut\_noise(out)

# =============================================================================

# plt.figure(4)

# plt.imshow(out)

# =============================================================================

#保存图片

# out.save('E://figures/img\_gray.jpg')

# 仅识别图片中的数字

#text = pytesseract.image\_to\_string(out, config='digits')

# 识别图片中的数字和字母

text = pytesseract.image\_to\_string(out)

# 去掉识别结果中的特殊字符

exclude\_char\_list = ' .:\\|\'\"?![],()~@#$%^&\*\_+-={};<>/¥'

text = ''.join([x for x in text if x not in exclude\_char\_list])

#print(text)

return text

path = "789.png"

text = OCR\_lmj(path)

print('text')

print(text)

def main():

# 识别指定文件目录下的图片yc

# 图片存放目录figures

dir = 'test'

correct\_count = 0 # 图片总数

total\_count = 0 # 识别正确的图片数量

# 遍历figures下的jpg文件

for file in os.listdir(dir):

if file.endswith('.jpg'):

#print(file)

image\_path = '%s/%s'%(dir,file) # 图片路径

answer = file.split('.')[0] # 图片名称，即图片中的正确文字

recognizition = OCR\_lmj(image\_path) # 图片识别的文字结果

print(recognizition)

print((answer, recognizition))

if recognizition == answer: # 如果识别结果正确，则total\_count加1

correct\_count += 1

total\_count += 1

print('Total count: %d, correct: %d.'%(total\_count, correct\_count))

'''

# 单张图片识别

image\_path = 'E://figures/code (1).jpg'

OCR\_lmj(image\_path)

'''

test = pd.DataFrame(data = recognizition,index = range(1,20001),columns=['y'])

test.index.name = 'id'

test.to\_csv('yanzhengma1.csv')

#main()

1. 我通过在网上查阅资料得知，可以使用tesseract直接对验证码进行识别，于是下载了tesseract-OCR,下载了整整一天，因为其中的语言包太多了。
2. 之后我对验证码图片进行处理，首先是将彩色图片灰度化，但是我发现因为验证码中内容与背景颜色及其相似，灰度化后会导致好多字母数字不能分辨出；于是我想是否可以先将背景转化为白色，之后在灰度化。
3. 灰度化后，我就找出图片中像素值最多的像素点，就是背景像素点，将其作为阈值，进行二值化处理，在阈值小范围附近的当做背景。
4. 之后是降噪，去除图片中的噪声点，获取每个点周围的九个像素点，如果该位置九宫格内黑色像素点少于一个阈值就将该点当做噪声点，将噪声点处像素值置为1.
5. 使用text = pytesseract.image\_to\_string(out)语句，识别验证码图片，text存储识别结果，
6. 使用exclude\_char\_list = ' .:\\|\'\"?![],()~@#$%^&\*\_+-={};<>/¥'
7. text = ''.join([x for x in text if x not in exclude\_char\_list])去除结果中的特殊字符。
8. 但是这个实验正确率非常低，而且我自己用非常清晰的没有噪声的背景纯白的验证码去验证也会识别错误，所以这次尝试行不通，要在这个基础上进行更改。
9. 最后加一句，Tesseract对于彩色图片的识别效果没有黑白图片的效果好。
10. Tesseract-ORC 对于这种弱验证码识别率还是可以，大部分字符能够正确识别出来。只不过有时候会将数字 8 识别为 0。如果图片验证码稍微变得复杂点，识别率大大降低，会经常识别不出来的情况。我自己也尝试收集 500 张图片来训练 Tesseract-ORC，识别率会有所提升，但识别率还是很低。
11. 如果想要做到识别率较高，那么需要使用 CNN (卷积神经网络)或者 RNN (循环神经网络)训练出自己的识别库。正好机器学习很火爆很流行，学习一下也无妨。
12. 在网上查阅资料得知tesseract对复杂验证码识别度不高，要用CNN训练自己的识别库，
13. CNN缺点：灵活性不高
14. CNN识别验证码不同于将图片进行分割等处理的识别方法，而直接对整个图片进行学习。因此只要验证码的风格与原样本不同时，识别准确率就比较低
15. 在CNN识别验证码中，卷积层对于图像是没有尺寸限制要求的。卷积仅于自身的卷积核大小，维度有关，输入向量大小对其无影响。但全连接层的输入是固定大小的，如果输入向量的维数不固定，那么全连接的权值参数的量也是不固定的，就会造成网络的动态变化，无法实现参数训练的目的。因此即使是同样的验证码如果改变了图片的大小，程序就会报错。不过这个问题貌似是可以解决的，具体请参见 https://blog.csdn.net/zhangjunhit/article/details/53909548。

tessedit\_char\_whitelist 0123456789-.

第二次尝试：

观察图片，发现每一个数字或者字母的颜色是一致的，可以先将验证码图片分成五张小图片，然后对小图片进行预测，其中像素值最多的是背景，第二多的是目标区域。

通过尝试，发现，背景不是只有一个颜色，像素值第二多的依旧是背景，不能通过像素点多少找到目标区域

由于训练集数据庞大，每次试验都要四五个小时。

第三次尝试

使用Keras和CNN

# -\*- coding: utf-8 -\*-

"""

Created on Fri Nov 22 13:02:13 2019

@author: DELL

"""

import numpy as np

import os

import win\_unicode\_console

win\_unicode\_console.enable()

from keras.preprocessing import image

from keras.models import Model

from keras.layers import Dense, Dropout, Flatten, Input, concatenate ,BatchNormalization

from keras.layers.convolutional import Conv2D, Convolution2D, MaxPooling2D

from keras.callbacks import ModelCheckpoint

from keras.optimizers import Adadelta

from keras.utils.vis\_utils import plot\_model

import tensorflow as tf

import glob

# 验证码所包含的字符 \_表示未知

captcha\_word = ['0','1','2','3','4','5','6','7','8','9','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','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']

# 图片的长度和宽度

width = 150

height = 30

# 每个验证码所包含的字符数

word\_len = 5

# 字符总数

word\_class = len(captcha\_word)

samples = glob.glob(r'train/\*.jpg')

# 验证码素材目录

train\_dir = 'train'

# 生成字符索引，同时反向操作一次，方便还原

char\_indices = dict((c, i) for i, c in enumerate(captcha\_word))

indices\_char = dict((i, c) for i, c in enumerate(captcha\_word))

# 验证码字符串转数组

def captcha\_to\_vec(captcha):

# 创建一个长度为 字符个数 \* 字符种数 长度的数组

vector = np.zeros(word\_len \* word\_class)

# 文字转成成数组

for i, ch in enumerate(captcha):

idex = i \* word\_class + char\_indices[ch]

vector[idex] = 1

return vector

# 把数组转换回文字

def vec\_to\_captcha(vec):

text = []

# 把概率小于0.5的改为0，标记为错误

vec[vec < 0.5] = 0

char\_pos = vec.nonzero()[0]

for i, ch in enumerate(char\_pos):

text.append(captcha\_word[ch % word\_class])

return ''.join(text)

# 自定义评估函数

def custom\_accuracy(y\_true, y\_pred):

predict = tf.reshape(y\_pred, [-1, word\_len, word\_class])

max\_idx\_p = tf.argmax(predict, 2)#这个做法牛逼，不用再做stack和reshape了，2，是在Charset那个维度上

max\_idx\_l = tf.argmax(tf.reshape(y\_true, [-1, word\_len,word\_class]), 2)

correct\_pred = tf.equal(max\_idx\_p, max\_idx\_l)

\_result = tf.map\_fn(fn=lambda e: tf.reduce\_all(e),elems=correct\_pred,dtype=tf.bool)

return tf.reduce\_mean(tf.cast(\_result, tf.float32))

#获取目录下样本列表

image\_list = []

#

for item in os.listdir(train\_dir):

image\_list.append(item)

np.random.shuffle(image\_list)

#创建数组，储存图片信息。样本个数、宽度和高度。

# 3代表图片的通道数，如果对图片进行了灰度处理，可以改为单通道 1

X = np.zeros((len(image\_list), height, width, 3), dtype = np.uint8)

# 创建数组，储存标签信息

y = np.zeros((len(image\_list), word\_len \* word\_class), dtype = np.uint8)

for i,img in enumerate(image\_list):

if i % 10000 == 0:

print(i)

img\_path = train\_dir + "/" + img

#读取图片

raw\_img = image.load\_img(img\_path, target\_size=(height, width))

#讲图片转为np数组

X[i] = image.img\_to\_array(raw\_img)

#讲标签转换为数组进行保存

y[i] = captcha\_to\_vec(img.split('.')[0])

#创建输入，结构为 高，宽，通道

input\_tensor = Input( shape=(height, width, 3))

x = input\_tensor

#构建卷积网络

#两层卷积层，一层池化层，重复3次。因为生成的验证码比较小，padding使用same

x = Convolution2D(32, 3, padding='same', activation='relu')(x)

x = Convolution2D(32, 3, padding='same', activation='relu')(x)

# x= BatchNormalization()(x)

x = MaxPooling2D((2, 2))(x)

x = Convolution2D(64, 3, padding='same', activation='relu')(x)

x = Convolution2D(64, 3, padding='same', activation='relu')(x)

# x= BatchNormalization()(x)

x = MaxPooling2D((2, 2))(x)

x = Convolution2D(128, 3, padding='same', activation='relu')(x)

x = Convolution2D(128, 3, padding='same',activation='relu')(x)

# x= BatchNormalization()(x)

x = MaxPooling2D((2, 2))(x)

#Flatten层用来将输入“压平”，即把多维的输入一维化，常用在从卷积层到全连接层的过渡。

x = Flatten()(x)

#为输入数据施加Dropout。Dropout将在训练过程中每次更新参数时随机断开一定百分比（rate）的输入神经元，Dropout层用于防止过拟合。

x = Dropout(0.25)(x)

x= BatchNormalization()(x)

#Dense就是常用的全连接层

#最后连接4个分类器，每个分类器是56个神经元，分别输出56个字符的概率。

x = [Dense(word\_class, activation='softmax', name='c%d'%(i+1))(x) for i in range(word\_len)]

output = concatenate(x)

#构建模型

model = Model(inputs=input\_tensor, outputs=output)

#这里优化器选用Adadelta，学习率0.1

opt = Adadelta(lr=0.1)

#编译模型以供训练，损失函数使用 categorical\_crossentropy，使用accuracy评估模型在训练和测试时的性能的指标

#这里使用自定义的评估模型

model.compile(loss = 'categorical\_crossentropy', optimizer=opt, metrics=['accuracy',custom\_accuracy])

#每次epoch都保存一下权重，用于继续训练

checkpointer = ModelCheckpoint(filepath="model/weights.{epoch:02d}--{val\_loss:.2f}.hdf5",

verbose=2, save\_weights\_only=True)

#开始训练，validation\_split代表10%的数据不参与训练，用于做验证集

model.fit(X, y, epochs= 3,callbacks=[checkpointer], validation\_split=0.1)

#保存权重和模型

model.save\_weights('model/captcha\_model\_weights.h5')

model.save('model/captcha\_\_model\_2.h5')

以上是训练模型

# -\*- coding: utf-8 -\*-

"""

Created on Fri Nov 22 13:22:17 2019

@author: DELL

"""

import numpy as np

import os

import logging

from keras.models import load\_model # 一系列网络层按顺序构成的栈

import glob

from keras.preprocessing import image

import tensorflow as tf

logger = logging.getLogger("forecast by model")

# 每个验证码所包含的字符数

word\_len = 5

image\_path = 'train1/'

# image\_path = '20190430/'

# 验证码所包含的字符 \_表示未知

captcha\_word = ['0','1','2','3','4','5','6','7','8','9','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','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']

# 字符总数

word\_class = len(captcha\_word)

#日志初始化

def init\_logger():

logging.basicConfig(

format='%(asctime)s : %(levelname)s : %(message)s',

level=logging.DEBUG,

handlers=[logging.StreamHandler()])

def custom\_accuracy(y\_true, y\_pred):

predict = tf.reshape(y\_pred, [-1, word\_len, word\_class])

max\_idx\_p = tf.argmax(predict, 2)#这个做法牛逼，不用再做stack和reshape了，2，是在Charset那个维度上

max\_idx\_l = tf.argmax(tf.reshape(y\_true, [-1, word\_len,word\_class]), 2)

correct\_pred = tf.equal(max\_idx\_p, max\_idx\_l)

\_result = tf.map\_fn(fn=lambda e: tf.reduce\_all(e),elems=correct\_pred,dtype=tf.bool)

return tf.reduce\_mean(tf.cast(\_result, tf.float32))

# load json and create model

def create\_model():

weight\_path = 'model/captcha\_\_model\_2.h5'

model = load\_model(weight\_path,

custom\_objects={'custom\_accuracy': custom\_accuracy})

return model

# 把数组转换回文字

def vec\_to\_captcha(vec):

text = []

# 把概率小于0.5的改为0，标记为错误

vec[vec < 0.5] = 0

char\_pos = vec.nonzero()[0]

for i, ch in enumerate(char\_pos):

text.append(captcha\_word[ch % word\_class])

return ''.join(text)

if \_\_name\_\_ == '\_\_main\_\_':

init\_logger()

model = create\_model()

image\_list = []

for item in os.listdir(image\_path):

image\_list.append(item)

np.random.shuffle(image\_list)

# 图片总数

image\_count = 0

# 成功次数

success\_count = 0

# 图片

for i, img in enumerate(image\_list):

if i % 10000 == 0:

print(i)

img\_path = image\_path + img

# 读取图片

raw\_img = image.load\_img(img\_path, target\_size=(30, 150))

code = img.replace('.jpg', '')

code = code.split('\_')[0]

logger.debug('正确的验证码为' + code)

X\_test = np.zeros((1, 30, 150, 3), dtype=np.float32)

X\_test[0] = image.img\_to\_array(raw\_img)

# 预测

predict = model.predict(X\_test)

n = 56 # 大列表中几个数据组成一个小列表

arr = []

arr.append(predict[0][0:56])

arr.append(predict[0][56:112])

arr.append(predict[0][112:168])

arr.append(predict[0][168:224])

arr.append(predict[0][224:280])

predictions = []

predictions.append(np.argmax(arr[0]))

predictions.append(np.argmax(arr[1]))

predictions.append(np.argmax(arr[2]))

predictions.append(np.argmax(arr[3]))

predictions.append(np.argmax(arr[4]))

# predictions = np.argmax(predict, axis=1)

# 标签字典

keys = range(56)

label\_dict = dict(zip(keys, captcha\_word))

result = ''.join([label\_dict[pred] for pred in predictions])

image\_count = image\_count + 1

if result == code:

success\_count = success\_count + 1

logger.debug("预测的结果为" + result)

logger.debug("目前正确率" + str(success\_count / image\_count))

logger.debug("总次数" + str(image\_count))

logger.debug("成功次数" + str(success\_count))

logger.debug("正确率" + str(success\_count/image\_count))

以上是测试正确率

提交CSV文件时，发现第一列数据的索引是0。

我发现读取测试集的时候，总是0,1,10,100……，而不是0,1,2,3,

读取文件的时候报错：

'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape

得知文件路径前要加上r。

无法使用placeholder：

import tensorflow.compat.v1 as tf

tf.disablel\_v2\_behavior()

代替

Import tensorflow as tf

在训练一个神经网络模型后，你会保存这个模型未来使用或部署到产品中。所以，什么是TF模型？TF模型基本包含网络设计或图，与训练得到的网络参数和变量。因此，TF模型具有两个主要文件：

a)meta图

这是一个拟定的缓存，包含了这个TF图完整信息；如所有变量等等。文件以.meta结束。

b)检查点文件：

这个文件是一个二进制文件，包含所有权重、偏移、梯度和所有其它存储的变量的值。这个文件以.ckpy结束。然而，TF已经在0.11版本后不再以这个形式了。转而文件包含如下文件 ：

mymodel.data-00000-of-00001

mymodel.index

.data文件包含训练变量。

除此之外 ，TF还包含一个名为“checkpoint”的文件 ，保存最后检查点的文件。

所以，综上，TF模型包含如下文件 ：

my\_test\_model.data-00000-of-00001

my\_test\_model.index

my\_test\_model.meta

checkpoint\*\*

tf.argmax()，将每一行或每一列中最大元素的下标求出。

tf.reshape(tensor，shape)#将tensor类型转化为shape形状，可以包含-1，

最后一次核心代码分析：

if \_\_name\_\_ == '\_\_main\_\_':

dc = dict()

preee = []

init\_logger()

model = create\_model()

image\_list = []

for item in os.listdir(image\_path):

image\_list.append(item)

#np.random.shuffle(image\_list)

# 图片总数

image\_count = 0

# 成功次数

success\_count = 0

# 图片

for i, img in enumerate(image\_list):

if i % 1000 == 0:

print(i)

img\_path = image\_path + img

# 读取图片

raw\_img = image.load\_img(img\_path, target\_size=(30, 150))

code = img.replace('.jpg', '')

code = code.split('\_')[0]

logger.debug('正确的验证码为' + code)

X\_test = np.zeros((1, 30, 150, 3), dtype=np.float32)

X\_test[0] = image.img\_to\_array(raw\_img)

# 预测

predict = model.predict(X\_test)

n = 62 # 大列表中几个数据组成一个小列表

arr = []

arr.append(predict[0][0:62])

arr.append(predict[0][62:124])

arr.append(predict[0][124:186])

arr.append(predict[0][186:248])

arr.append(predict[0][248:310])

predictions = []

predictions.append(np.argmax(arr[0]))

predictions.append(np.argmax(arr[1]))

predictions.append(np.argmax(arr[2]))

predictions.append(np.argmax(arr[3]))

predictions.append(np.argmax(arr[4]))

# 标签字典

keys = range(62)

label\_dict = dict(zip(keys, captcha\_word))

result = ''.join([label\_dict[pred] for pred in predictions])

image\_count = image\_count + 1

dc[int(code)] = result

if result == code:

success\_count = success\_count + 1

logger.debug("预测的结果为" + result)

preee.append(result)

logger.debug("正确率" + str(success\_count/image\_count))

def save\_csv(labels):

test\_data\_0=sorted(labels.items(),key=lambda x:x[0])

te = np.array(test\_data\_0)

#print(te)

length = len(te[:,1])

test = pd.DataFrame(data = te[:,1],index = range(0,length),columns=['y'])

test.index.name = 'id'

test.to\_csv('test8.csv')

save\_csv(dc)

首先是训练过程，我使用的CNN模型，两层卷积层，一层池化层，重复3次。

Flatten层用来将输入“压平”，即把多维的输入一维化，常用在从卷积层到全连接层的过渡。

Dropout将在训练过程中每次更新参数时随机断开一定百分比（rate）的输入神经元，Dropout层用于防止过拟合。

最后连接4个分类器，每个分类器是62个神经元，分别输出62个字符的概率。

之后是将训练过程中生成的文件用于测试。

使用onehot编码，即每个验证码有五个字符，每个字符用62个0.1表示，按顺序排列，1代表正确的字符。将测试集带入，得出结果，写到CSV中。

重难点：

1.我训练的时候将epoch设置为100，训练时间是将近14个小时。

2.训练过程及其复杂，因为数据庞大，每次训练都要很长时间，之后再做调整，从早上到晚上都不行，熄灯之后电脑还要运行。

3.保存文件的时候，需要创建一个字典，然后将其排序，按照自然数顺序排列。

4.写入CSV的时候，需要从0开始。