**一、猫狗大战（弃坑）**

**现在及以后的策略：**

先选出一篇最适合的，按照他的方法实现。再看其他的，从而进行修改。

os模块包含操作系统相关的功能，可以处理文件和目录这些我们日常手动需要做的操作

os.listdir(path)：返回指定路径下的文件和文件夹列表

np.hstack():在水平方向上平铺

tf.split(dimension, num\_split, input)：

dimension：输入张量的哪一个维度，如果是0就表示对第0维度进行切割。

num\_split：切割的数量，如果是2就表示输入张量被切成2份，每一份是一个列表。

tensorlayer的tl.iterate.minibatches只产生batch\_size整除的样本，最后剩下的不足batch\_size大小的样本没有用上

Shape must be rank 0 but is rank 1 for 'ReadFile\_1' (op: 'ReadFile') with input shapes: [12500]：

Input：

Right：Tensor("input\_producer/Gather:0", **shape**=(), dtype=string)

False：Tensor("Cast/x:0", shape=(12500,), dtype=string)

tf.train.slice\_input\_producer将list输出为tensor，且shape=()

..=sess.run([])记得加方括号

Tensorflow修改全局变量：global声明

'NpzFile' object attribute 'item'：查看了一下大小，npy文件用的网盘以及pandownload，中途大小出了问题。

Tf.slice\_input\_producer 没有指定, num\_epochs=1，结果无限遍历tensor\_list

The name '1' refers to an Operation not in the graph：还是把train和test分开写，同时参考别人的函数。另外参见：<https://blog.csdn.net/qq_41185868/article/details/82903223，tf.identity>

Plt操作

目前来看，出错主要在于：

1. test时，怎样加载模型？p的类型是怎样的（run…[0]）？怎样将p保存？
2. 最大的疑问：怎么加载，是否用get-tensor-byname，还是直接用原变量？
3. 用keras是否更好？网上大都用的。
4. 怎么验证？验证集怎么生成？

我的建议：回来之后暂时不搞这个了，先换一个项目。之后首先确定框架！

**二、植物幼苗分类**

**Os模块：**

1. **os.path.exists(path)**

如果path存在,返回True;如果path不存在,返回False

1. **os.listdir()** 方法

用于返回指定的文件夹包含的文件或文件夹的名字的列表

1. **os.mkdir(**'Test\\path\_01\\path\_02\\path\_03'**)**

创建路径中的最后一级目录，即：只创建path\_03目录，而如果之前的目录不存在并且也需要创建的话，就会报错。（4）**os.makedirs()**

创建多层目录，即：Test,path\_01,path\_02,path\_03如果都不存在的话，会自动创建

**if not os.path.exists**(pict\_Path): #还是很有必要的

os.makedirs(pict\_Path)

（5）**os.path.basename(path)**

返回path最后的文件名。若path以/或\结尾，那么就会返回空值。

os.path.basename(path='D:\CSDN')=CSDN os.path.basename('c:\test.csv')='test.csv'

（6）**os.path.dirname(path)**

去掉文件名，返回上一级目录

os.path.dirname("E:/Read\_File/read\_yaml.py")=E:/Read\_File

（7）

os.path.isdir()和os.path.isfile()：

判断是否为目录/文件。传入的参数是绝对路径，

~~常见错误：os.listdir()返回的只是一个某个路径下的文件和列表的名称.直接使用os.listdir()的返回值当做os.path.isdir()和os.path.isfile()的入参~~

正确用法：需要先使用python路径拼接os.path.join()函数，将os.listdir()返回的名称拼接成文件或目录的绝对路径再传入os.path.isdir()和os.path.isfile().

（8）**os.walk**方法，主要用来遍历一个目录内各个子目录和子文件。

os.walk(top, topdown=True, onerror=None, followlinks=False)

可以得到一个三元tupple(dirpath, dirnames, filenames),

第一个为**起始路径**，第二个为起始路径下的**文件夹**，第三个是起始路径下的**文件**。

**区别：**

情况1：在一个目录下面只有文件，没有文件夹，这个时候可以使用os.listdir

情况2：递归的情况，一个目录下面既有目录也有文件，使用os.walk：

（9）**os.getcwd()** 方法用于返回当前工作目录

一种很常见的提取路径、之后shuffle之代码：**特别注意，下面的file 、key自身左右是没有斜杠的**

image\_list = []

label\_list = []

# 载入数据路径并写入标签值

for file in os.listdir(file\_dir): #file\_dir以’/’结尾

for key in os.listdir(file\_dir + file):

image\_list.append(file\_dir + file + r'/' + key) #或者’\\‘

label\_list.append(name\_dic[file])

X\_train,X\_test, y\_train, y\_test =**train\_test\_split**(train\_data,train\_target,test\_size=0.3, random\_state=0 ,shuffle,stratify)

参数解释：

train\_data：被划分的样本特征集。可以时list，也可以是np.array！

train\_target：被划分的样本标签

test\_size：如果是浮点数，在0-1之间，表示样本占比；如果是整数的话就是样本的数量

random\_state：是随机数的种子。

stratify是为了保持split前类的分布。比如有100个数据，80个属于A类，20个属于B类。如果train\_test\_split(... test\_size=0.25, stratify = y\_all), 那么split之后数据如下：

training: 75个数据，其中60个属于A类，15个属于B类。

testing: 25个数据，其中20个属于A类，5个属于B类。

用了stratify参数，training集和testing集的类的比例是 A：B= 4：1，等同于split前的比例（80：20）。通常在这种类分布不平衡的情况下会用到stratify。

**Opencv**:

(1)**cv2.imread()**：

读入图片，类似于tf.gfile.fastgfile、keras.preprocessing.image.load\_img。读进来直接是BGR 格式数据格式在 0~255，通道格式为(W,H,C),之后记得用img\_to\_array

共两个参数，第一个参数为要读入的图片文件名，第二个参数为如何读取图片，包括cv2.IMREAD\_COLOR：读入一副彩色图片；cv2.IMREAD\_GRAYSCALE：以灰度模式读入图片；cv2.IMREAD\_UNCHANGED：读入一幅图片，并包括其alpha通道。

转换成rgb：

im = cv2.imread(im\_file)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

(2) **cv2.resize**: <https://blog.csdn.net/jningwei/article/details/76019940>

参数输入是 （x轴，y轴），即 宽×高

(3)cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)：

图像颜色格式转换

可用于语义分割，此时和mask = cv2.inRange(hsv, lower, upper) 搭配使用：低于lower或高于upper变为0，其余变为255.

(4)这次的项目还涉及一些和形态学相关的opencv函数，略。

**img\_to\_array**就是将图片转化成数组，读取图片后使用；常与load\_img连用

Numpy数组的保存与读取：

**np.save**和**np.load**是读写磁盘数组数据的两个主要函数。默认情况下，数组以未压缩的原始二进制格式保存在扩展名为npy的文件中，以数组a为例

np.save("filename.npy",a)

b = np.load("filename.npy")

pandas.crosstab：一种表格制作方式；另见crossconfusion\_matrix

使用HSV，比起RGB更容易表示color range

有时只能使用flow，不用flow\_from\_directory

DataFrame：使用字典创建;此类有to\_csv方法

混淆矩阵：pd.crosstab；用字典创建时，参数为list。

Ram莫名其妙超了，无法理喻：

之前先求images\_path🡪split🡪分别read\_image，不可；后来直接一起read\_images🡪split反而可以了。Ram占用率近似两倍关系。感觉即使是读取相同的量，一次读完也比多次读好。

Input 0 is incompatible with layer global\_average\_pooling2d\_1：直接看出形状不对。Include\_top之后没有引号！

绘图函数：

def show\_train\_history(H=model.fit):

plt.style.use("ggplot")

plt.figure()

N = len(H.epoch) # 训练周期数

plt.plot(np.arange(0, N), H.history["loss"], label="train\_loss")

plt.plot(np.arange(0, N), H.history["val\_loss"], label="val\_loss")

plt.plot(np.arange(0, N), H.history["acc"], label="train\_acc")

plt.plot(np.arange(0, N), H.history["val\_acc"], label="val\_acc")

plt.title("Loss and Accuracy ")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig('plot.png')

plt.show()

另一种绘图方法：

plt.figure(figsize=(12,8))

sns.lineplot(range(1, epochs+1), history.history['acc'], label='Train Accuracy')

sns.lineplot(range(1, epochs+1), history.history['val\_acc'], label='Test Accuracy')

plt.show()

还是要先观察一下测试集，看看与训练集有什么不同的地方。而且打印出.evaluate、混淆矩阵等。

**项目总结：**

本次项目是一个单标签多分类项目，训练集共4750张图片，测试集共794张图片。

将训练集划分成80%训练集和20%验证集。为增强模型泛化能力采用数据增强，利用keras的Imagegenerator作为生成器生成训练数据，其中各参数如下：

rotation\_range=180,

width\_shift\_range=0.3,

height\_shift\_range=0.3,

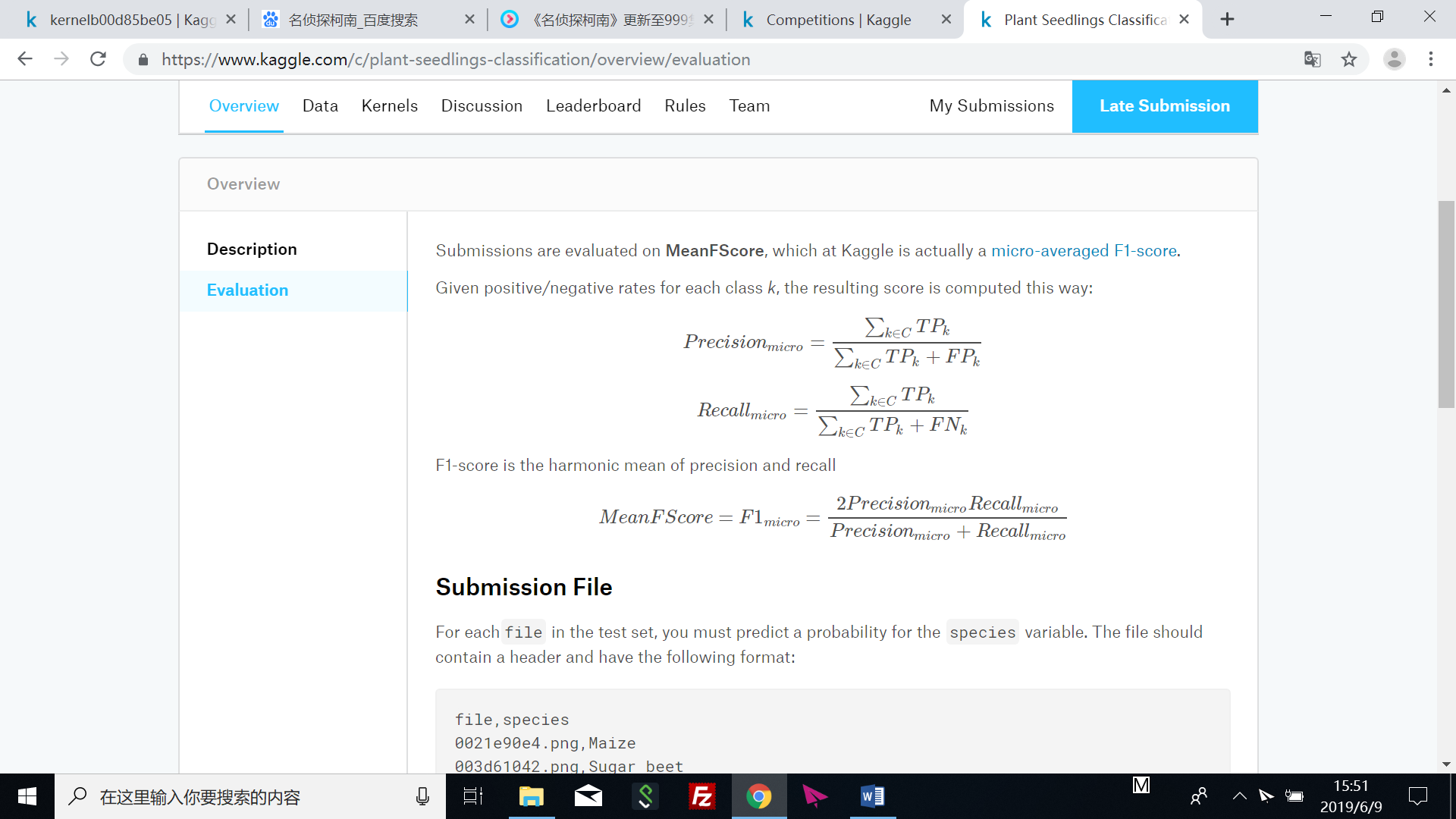
zoom\_range=0.3,

horizontal\_flip=True,

vertical\_flip=True

网络部份使用了迁移学习，是imagenet上训练好的Xception+自己设计的全连接层（每一层均可训练）。优化器是Adadelta；回调函数使用了学习率下降ReduceLROnPlateau，其检测指标为验证机准确率val\_acc；损失函数是cross\_entropy。Batch\_size=16,epochs=100。

模型评分标准为：



最终得分0.97355。

**其他：**

①测试了在预处理时，将叶片部分以外部分颜色设置为黑色，但最终score=0.93073 。推测的可能原因：关于图中绿色之范围(代码中“sensitivity”)一值设置得不好；这样的预处理会使模型更容易拟合，而我在如此处理后epochs并未减小，导致最终过拟合。

②即使是读取相同的数据，相比一次性读取完，多次读取会导致ram占用率上升。

③根据loss、acc曲线，可以采用earlystopping，不知道性能是否会上升。

④由于训练集各样本数量不一致，属于不平衡训练集，我们采用了上述的评分方法。除此之外，若能使用其他方法，包括smote supervised filter等方法，模型性能应可以更好。

⑤如果能试一下其他的网络，或许能够得到更好的性能。看到一个高赞的kernel，cnn中使用了batch\_normalization、leaky\_relu，不知是什么原理。

**三、分心司机**

VGG输入大小有要求，貌似是224，大于则截取；还需要减去rgb平均值。

关于分批读取数据、防止内存溢出：<https://www.jianshu.com/p/5bdae9dcfc9c>

网络选择：

用Keras写了个分类程序将VGG19应用于kaggle数据state-farm-distracted-driver-detection。本来想用Xception模型的。但是发现只要使用Xception模型，基于imagenet训练好的权重fine tune在第一个epoch必然会过拟合，不知道怎么解决。使用了VGG19就没有这个问题

Python strip() 方法用于移除字符串头尾指定的字符（默认为空格或换行符）或字符序列。返回移除后剩下的字符串。

注意：该方法只能删除开头或是结尾的字符，不能删除中间部分的字符。

**glob模块：**

最简单的模块之一，内容非常少。用它可以查找符合特定规则的文件路径名

（1）glob.glob()：

返回所有满足条件（如\*）匹配的文件路径列表。

print (glob.glob(r'../\*.py'))：打印该路径下所以以.py结尾的文件

**Pickle模块**的dump()方法和load()方法

pickle.dump(obj, file, [,protocol])

序列化对象，将对象obj**保存**到文件file中去。

参数protocol是序列化模式，默认是0（ASCII协议，表示以文本的形式进行序列化），protocol的值还可以是1和2（1和2表示以二进制的形式进行序列化。其中，1是老式的二进制协议；2是新二进制协议）。file表示保存到的类文件对象，file必须有write()接口，file可以是一个以'w'打开的文件或者是一个StringIO对象，也可以是任何可以实现**write()**接口的对象。

pickle.load(file)

注释：反序列化对象，将文件中的数据解析为一个python对象。file中有**read()接口和readline()**接口

Pd.to\_json()：类似于to\_csv

numpy.random.permutation：类似于numpy.random.shuffle()，但是是实现排列，且有返回值。其参数除了 array\_like 还可以是 int 类型，如果是 int 类型，那就相当于随机打乱 numpy.arange(int)

**K折交叉验证：sklearn.model\_selection.KFold(n\_splits=3, shuffle=False, random\_state=None)**

（1）思路：

将训练/测试数据集划分n\_splits个互斥子集，每次用其中一个子集当作验证集，剩下的n\_splits-1个作为训练集，进行n\_splits次训练和测试，得到n\_splits个结果

（2）注意点：

对于不能均等份的数据集，其前n\_samples % n\_splits子集拥有n\_samples // n\_splits + 1个样本，其余子集都只有n\_samples // n\_splits样本

（3）参数说明：

n\_splits：表示划分几等份

shuffle：在每次划分时，是否进行洗牌

①若为Falses时，其效果等同于random\_state等于整数，每次划分的结果相同

②若为True时，每次划分的结果都不一样，表示经过洗牌，随机取样的

random\_state：随机种子数

（4）属性：

①get\_n\_splits(X=None, y=None, groups=None)：获取参数n\_splits的值

②split(X, y=None, groups=None)：将数据集划分成训练集和测试集，返回下标索引生成器

如：

data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])

kfold = KFold(n\_splits=3, shuffle = True, random\_state= 1)

for train, test in kfold**.split**(data):

print('train: %s, test: %s' % (data[train], data[test]))

train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]

train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]

train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]

keras.layers.Add：张量相加

pd.read\_csv：

WRNs：Wide Residual Networks之应用

一句x\_train=np.array(x\_train0).astype('float32')/255.0瞬间使ram超了，真是无法理喻！

Iamges要除以255，首先要转成np.array, 否则unsupported operand type(s) for /: 'list' and 'int'

其实有时候不除以255也没有关系，除了还可能出bug

本次要求输出是各类别概率值，因此使用.predict方法、之后不使用np.argmax即可。

pd.DataFrame：若用字典创建，则每一列必须是**list**！而Model.predict方法返回的是形如p=[[p11,p12,…], [p21,p22,…],…]的各类别概率值，是nparray不是list。若要保存在dataframe中，使用：list(to\_categorical(p).flatten())（f小写）。

关于**flatten**用法：<https://blog.csdn.net/maoersong/article/details/23823925>

由于数据集太大，使用predict\_generator。主要问题出在gen生成上，如下：

flow\_from\_directory：以前的理解错了！在该路径下，还应该有子文件夹，每个文件夹再包含照片，一个文件夹代表一个类。

报错：Found 0 images belonging to 0 classes

解决：生成子文件夹。I guess you can try creating a subfolder under your 'test' folder, i.e. put all your test images under this folder: 'D:\Datasets\Trell\images\new\_images\testing\**subfolder**'

As for the function flow\_from\_directory(), you'll always need a subfolder regardless of whether you are training, making validation or testing.

生成csv文件时arrays must all be same length，原因是生成generator时，不能整除部分舍弃了。

本项目中generator：指定classmode=None.

Val Loss震荡的厉害，先试一下改opt、batchsize增大。

实践表明，Adam 比其他适应性学习方法效果要好。

如何选择？

如果数据是稀疏的，就用自适用方法，即 Adagrad, Adadelta, RMSprop, Adam。

RMSprop, Adadelta, Adam 在很多情况下的效果是相似的。

Adam 就是在 RMSprop 的基础上加了 bias-correction 和 momentum，随着梯度变的稀疏，Adam 比 RMSprop 效果会好。整体来讲，Adam 是最好的选择。

很多论文里都会用 SGD，没有 momentum 等。SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。

如果需要更快的收敛，或者是训练更深更复杂的神经网络，需要用一种自适应的算法。

Score太大了，18+！后来和data对比，发现标签和预测对不上！.filenames方法确实好用。

目前来看，filenames、save\_dir方法都试过了。打算最后一试自定义生成器法，还不行的话那就不碰这个项目了。

**自己写的generator**：（总算好了！） ? **labels = (train\_generator.class\_indices)**

def self\_generator(paths=paths, samples=len(paths), bs=32):

#global names

"""

if samples % bs ==0:

num = samples/bs

else:

num=samples//bs+1

"""

num=samples//bs+1

while 1:

for i in range(num): #这句话真的是必须的吗？

x=[]

list=paths[i\*bs : (i+1)\*bs]

#没有必要进行范围判断，list中超出范围不碍事！

““”

特别注意**下标索引范围**：

a=[1,2,3]

print(a[1:**5**])

>>>[2,3]

“”“

for file in list:

img=cv2.imread(file)

img=cv2.resize(img,(224,224))

x.append(img)

x=np.array(x) #其他项目有人要np.concatenate。**注意：x必须有这步，y是list即可，但要onehot，#to\_categorical**

yield x

**项目总结：**

本次项目是一个单标签多分类问题，使用了作者自定义网络，优化方法为adam。最终在验证集上准确率约为92%+，得分2+。

本次遇到的主要问题：

①验证集上loss、acc震荡，换了优化器、batchsize，均无明显效果，我觉得主要是网络问题。另外，testsize似乎有些大了。【关于震荡问题，用正则化可能会有效果】

②由于数据太大，因此采用分批读取的生成器。而在imagedatagenerator中，flowfromdirectory方法，要求文件夹中还有文件夹，且生成的bs顺序和读进去的路径不一致，最终改用我自己写的gen。

**四、火鸡**

**DataFrame**：

①由pd.read\_json、pd.read\_csv、pd.read\_excel等得到；

③形如：

one two three

22 1 2 3

33 4 5 6

44 7 8 9

④操作详见<https://blog.csdn.net/maliang_1993/article/details/50907983>

⑤dataframe .head()会将excel表格中的第一行看作列名，并默认输出之后的五行，在head后面的括号里面直接写你想要输出的行数也行，比如2，10，100之类的。

⑥dataframe.info()查看基本信息

⑦dataframe[‘col’].values [后面不接括号哦]：只返回值，去掉行index

**Seaborn**：

统计数据可视化

set\_style( )是用来设置主题的

barplot (条形图) &countplot（分类直方图）&pointplot（散点图）：这里不深究，以后遇到再说。

Python的apply,map,lambda函数应用：

<https://blog.csdn.net/m0_37712157/article/details/84331493>

总结：要对数据进行应用函数时，先看数据结构是DataFrame还是Series,Seriesj结构直接用map(),DataFrame结构的话再看是要按行还是按列进行操作来选择对应的函数即可

Vggish音频特征提取，没有要求理解。

有人还使用了attention机制，没学过就不搞了。

Len求列表长度：l=[[1,2],[3,2],[5,1]]，len(l)=3

使用了CudnnGRU（针对gpu优化的循环层），不了解

plt.figure(figsize=(12,8))

figsize:指定figure的宽和高，单位为英寸

**项目总结**：

**先搞清楚数据是什么样的**（格式、大小、排列、分类及各类别数量。。。）。这次的数据（audio\_embedding），一个列表中有x个列表(x<=10)，其中每一个列表长度为128 。

有地方与之前不同，最后在预测前将train、val数据合并，又训练了相同epochs。不过，这次训练使用的权重是初始化权重！

使用的网络：

* BatchNormalization : Normalizes the Data
* Two Bidirectionnal CuDNNGRU : They are recurrent layers optimized for GPU. The idea is that they can interprete the temporal aspect of our data.
* Pooling Layers : Select the Average or Maximum value in the temporal axis. I use both and concatenate the outputs.
* Dense + ReLu : To get some extra info
* Dense + sigmoid : The output layer, to get the score between 0 and 1

**五、乳腺癌**

**网络：**

* input layer: [., 50, 50, 3]
* layer: Conv1 -> ReLu -> MaxPool: [., 25, 25, 36]
* layer: Conv2 -> ReLu -> MaxPool: [., 13, 13, 36]
* layer: Conv3 -> ReLu -> MaxPool: [., 7, 7, 36]
* layer: FC -> ReLu: [., 576] -> dropout
* output layer: FC -> ReLu: [., 2]
* k折验证

**思维导图：**

代码有很多新奇的地方，但逻辑混乱

网络里一些相同的构造方式为简便就写成了一个类函数

#打乱数据，用np.random.shuffle，但只适用于一维。因此注意是结合get\_state还是直接shuffle下标。（X）

Np格式的shape，直接**.Shape**

技巧：

imgs\_0 = x\_images[y\_images == 0] *# 0 = no cancer*

imgs\_1 = x\_images[y\_images == 1] *# 1 = cancer*

**plt**用法：

plt.figure(figsize=(20,20))

for i **in** range(30):

plt.subplot(5,6,i+1)

plt.title('IDC = **%d**'%y\_images[i])

plt.imshow(x\_images[i])

dense转换成onehot：

def dense\_to\_one\_hot(labels\_dense, num\_classes):

"""Convert class labels from scalars to one-hot vectors."""

num\_labels = labels\_dense.shape[0]

index\_offset = np.arange(num\_labels) \* num\_classes

labels\_one\_hot = np.zeros((num\_labels, num\_classes))

labels\_one\_hot.flat[index\_offset + labels\_dense.ravel()] = 1

return labels\_one\_hot

numpy有argmax，但没有**onehot**，自己写（cs231写过）：

a = np.array([1, 0, 3])

n\_class = np.max(a) + 1

np.eye(n\_class)[a]

>>>

array([[ 0., 1., 0., 0.],

[ 1., 0., 0., 0.],

[ 0., 0., 0., 1.]])

**Tf中的数据增强**：一个很好的tf+keras的例子：

# generate new images via rotations, translations, zoom using keras 数据增强

def generate\_images(self, imgs):

print('generate new set of images')

# rotations, translations, zoom

image\_generator = keras.preprocessing.image.**ImageDataGenerator**(

rotation\_range = 10, width\_shift\_range = 0.1 , height\_shift\_range = 0.1,

zoom\_range = 0.1)

# get transformed images 一次读入所有的imgs，注意.next方法

images = image\_generator.**flow**(imgs.copy(),**batch\_size**=len(imgs), shuffle = False).**next**()

# np.zeros(len(imgs)),多余的

return images #[0]

**总结**：imgs是这次的训练数据，我想加强他，则：

**Imgs\_aug=imagedatagenerator.flow(imgs,bs=len(imgs)).next()**

**tf.reset\_default\_graph**函数用于清除默认图形堆栈并重置全局默认图形。

注意：默认图形是当前线程的一个属性。该tf.reset\_default\_graph函数只适用于当前线程。当一个tf.Session或者tf.InteractiveSession激活时调用这个函数会导致未定义的行为。调用此函数后使用任何以前创建的tf.Operation或tf.Tensor对象将导致未定义的行为。

**tf.get\_default\_graph()**

功能：获取当前默认计算图

Tf.Variable()：

validate\_shape：默认为True，形状不接受更改，如果需要更改，validate\_shape=False。

**加载持久化模型**的三种方式：

第一，saver.restore：属于当前代码计算图已经定义，我需要将持久化模型中变量的值加载到当前代码计算图中的变量中去。所以，两者只能是持久化模型中的变量是当前代码计算图中变量集合的一个子集，必须是一种严格包含的关系。

第二，**import\_meta\_graph**则完全不允许当前代码计算图已经定义的变量节点和要加载的持久化模型中的节点存在冲突，因为它加载的是整个图(.mata)。不是单纯将变量的值进行加载。常与**get\_tensors\_by\_name**连用：

tf.reset\_default\_graph()

saver = tf.train.import\_meta\_graph(‘./.mata’)

saver.restore(sess,’./.ckpt’)

graph = tf.get\_default\_graph()

h\_conv1\_tf = graph.get\_tensor\_by\_name('h\_conv1\_tf:0')

第三，使用pb文件的方式，即使持久化模型中的变量节点和当前代码计算图中定义的变量节点发生冲突，也是毛事没有，没有任何关系。因为pb文件的方式加载进来的计算图，会全部加上import/前缀。也就是说，从命名空间上就隔开了这种冲突。

上述三点是很重要的结论，自己调试和观察的。（是市面上书中找不到的结论。）

所以，使用别人的模型，最好的方式就是用pb文件。因为不会有冲突的可能！但是，有一个弊端就是，import命名空间下的变量不能参与到当前的训练！！！记住，不能参与训练！





常见的**acc**写法：

acc=tf.reduce\_mean(tf.cast( tf.equal(tf.argmax(self.y\_pred\_proba\_tf, 1),tf.argmax(self.y\_data\_tf, 1)),dtype=tf.float32))

print(np.**concatenate**(([1,2,3],[2,5,8]),axis=**0**))

>>>[1 2 3 2 5 8]

①注意这种**feed\_dict**：

sess = tf.Session(),

saver.**restore**(sess, mn)

train\_acc={} #字典

train\_acc = nn\_graph.train\_acc\_tf.**eval(session = sess)**

②kaggle上**绘loss、acc图**，是先将loss、acc写入tensorboard，之后再读出来，再用plt：

plt.plot(np.arange(0,len(train\_acc[mn])), train\_acc[mn],'-b', label='Training')

**混淆矩阵画法：**

cnf\_valid\_matrix['neural\_network'] = sklearn.metrics.confusion\_matrix(

y\_pred = onehot\_dense(y\_valid\_pred[mn]),

y\_true = onehot\_dense(y\_valid)).astype(np.float32)

# normalize

cnf\_valid\_matrix['neural\_network'][0,:] = cnf\_valid\_matrix['neural\_network'][0,:]/cnf\_valid\_matrix['neural\_network'][0,:].sum()

cnf\_valid\_matrix['neural\_network'][1,:] = cnf\_valid\_matrix['neural\_network'][1,:]/cnf\_valid\_matrix['neural\_network'][1,:].sum()

# plot

labels\_array = ['IDC = 0', 'IDC = 1']

fig, ax = plt.subplots(1,figsize=(5,5))

ax = sns.heatmap(cnf\_valid\_matrix['neural\_network'], ax=ax, cmap=plt.cm.Greens, annot=True)

ax.set\_xticklabels(labels\_array)

ax.set\_yticklabels(labels\_array)

**六、猫狗**

tf.shape()和get\_shape()都可以得到tensor的尺寸

tf.shape(a)中a的数据类型可以是tensor, list, array

a.get\_shape()中a的数据类型只能是tensor,且返回的是一个元组（tuple）；与as\_list连用转化为列表

tf也有flatten，但你也可以用reshape

**迁移学习：**

kernel = tf.get\_variable("weights", [3, 3, in\_channel, out\_channel], dtype=tf.float32,trainable=False)

biases = tf.get\_variable("biases", [out\_channel], dtype=tf.float32,trainable=False)

self.parameters += [kernel, biases]

……

def load\_weights(self, weight\_file, sess):

weights = np.load(weight\_file)

keys = sorted(weights.keys())

for i, k in enumerate(keys):

if i not in [30,31]:

sess.run(self.parameters[i].assign(weights[k]))

print("-----------all done---------------")

1. python sorted(dict.keys())：按照dict key的字母顺序，由小到大排列。至于为什么可以这样加载参数，应该是因为npz文件存储字典（参数名称：权重值），字典里参数名称定义时，从前往后就是按照字母顺序从大到小定义的。
2. tf.get\_variable定义的，用assign进行赋值

注意**tf.nn.conv**中weights的定义（Variable、get\_variable）、shape(四维分别是什么)、赋值(assign)

Np.random.shuffle只对第一维shuffle

Tensorflow生成batch—**tf.train.batch**用法：

常与**tf.train.slice\_input\_producer**连用，此函数有shuffle参数。slice\_input\_producer()这个函数有一个形参**num\_epochs**，通过设置它的值就可以控制**全部**数据循环输出几次

此函数的第一个参数，应该是tensor的字典或者列表，如[img,label]。但是：这里的img只能是**一个**tensor（应该类似于keras生成器），s\_i\_p恰好生成的就是一个tensor，而之前用的所有的tensor，结果错误。

那么理论上用keras gen也可以，试一下？

Tf和keras二分类问题，最后一层节点数不一样！

使用sparse\_softmax\_cross\_entropy\_with\_logits报错：Rank mismatch: Rank of labels (received 2) should equal rank of logits minus 1

解决方式：tfplaceholder中的shape，或者换一个loss函数。

报错：**Can not convert a ndarray into a Tensor or Operation.**

错误指示是run这里出了错 **，Sess左右不可同名！**

之前训练时是为了生成batch采用tf.read\_file，测试时则不再需要了，太麻烦。

得分不理想，将bs25—>32，epochs200—100

Os.listdir是不是有问题？将train、test中的照片读取路径时总会出现一个…/train/train、…/test/test路径被读进来。这导致了各种奇怪的报错，诸如：failedpreconditionerror is a directory

答：这个函数没问题，下载下来的数据也没有问题，云主机上的dataset有问题。

得分从3降到1.但还是非常不好，该怎么改善呢？

目前想到的：

学习率；优化方式；epochs；网络不合适；bs

Epochs6—10，分数太大了!应该是过拟合。

Plt figsize不对，怎么设置呢？

现在减小lr（？试一下指数）。减小了lr，看来还是epoch太大过拟合了。

现在改用adam，lr=0.001，结果怎么回事，是卡在局部最小点了吗。减小lr0.0001，现在loss下去了，分数还是2。

这个网络有问题，应该是全连接层全都可以train，他只有最后一层。

**Loss=0.693不变**

这种情况：

**A首先还是调节lr（减小）**

B另一个可能是weight初始化问题。

C以及训练要逐渐前进。

**D另外，加上batch——normalzation**。（relu—>leaky relu）

**E在全连接层用正则化**

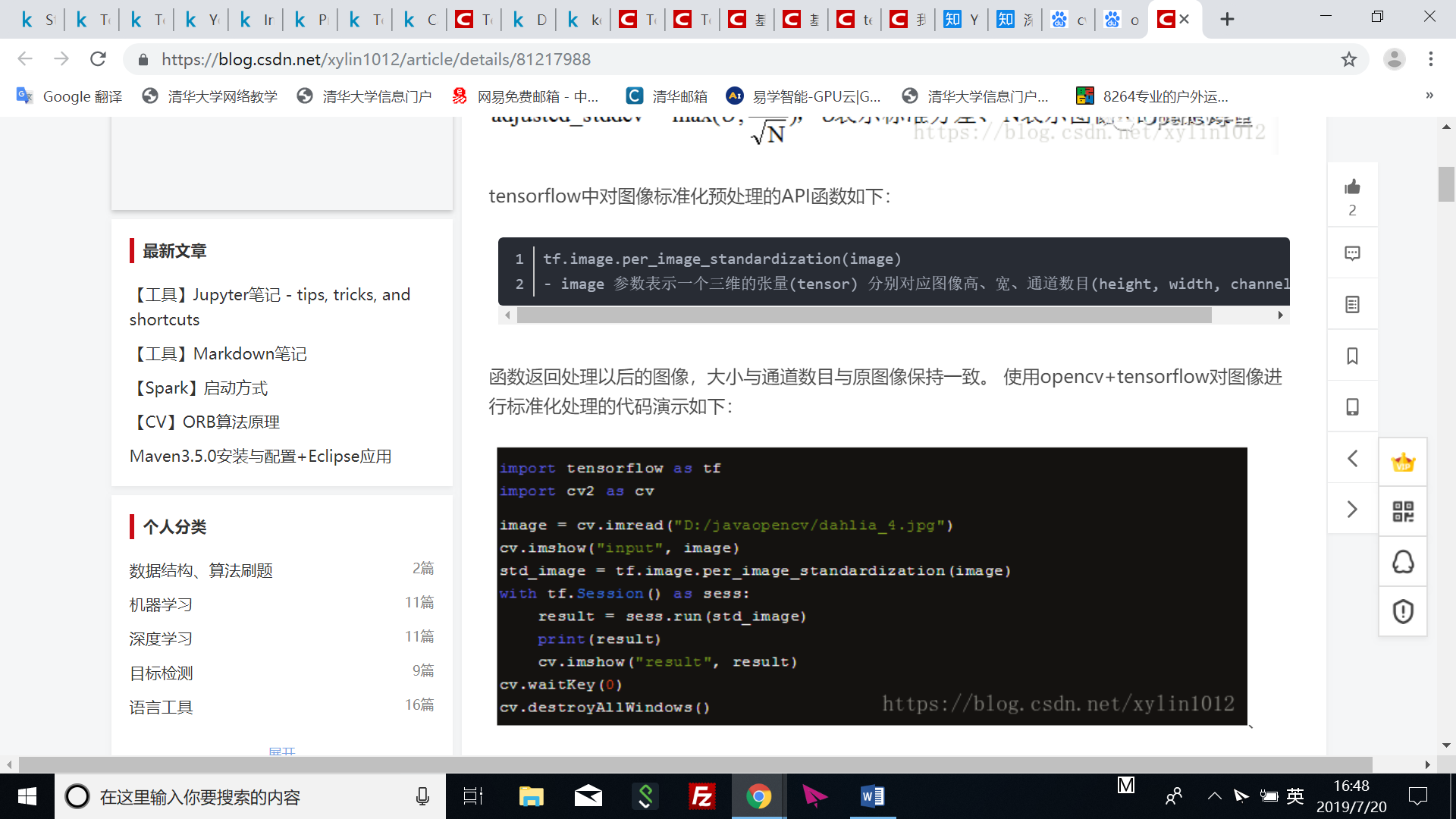
**F输入数据normalize**

现在：adam，lr=0.00001，score=2.94，loss=0，bug倒是没有了。难道是过拟合了吗？试一下epoch降低，加入正则化。

不知道为什么，现在的得分还没有原版好！----果然最大的问题是标准化，用了反而出问题了！也不能说是标准化没用吧，而是训练和测试数据处理不一致造成。

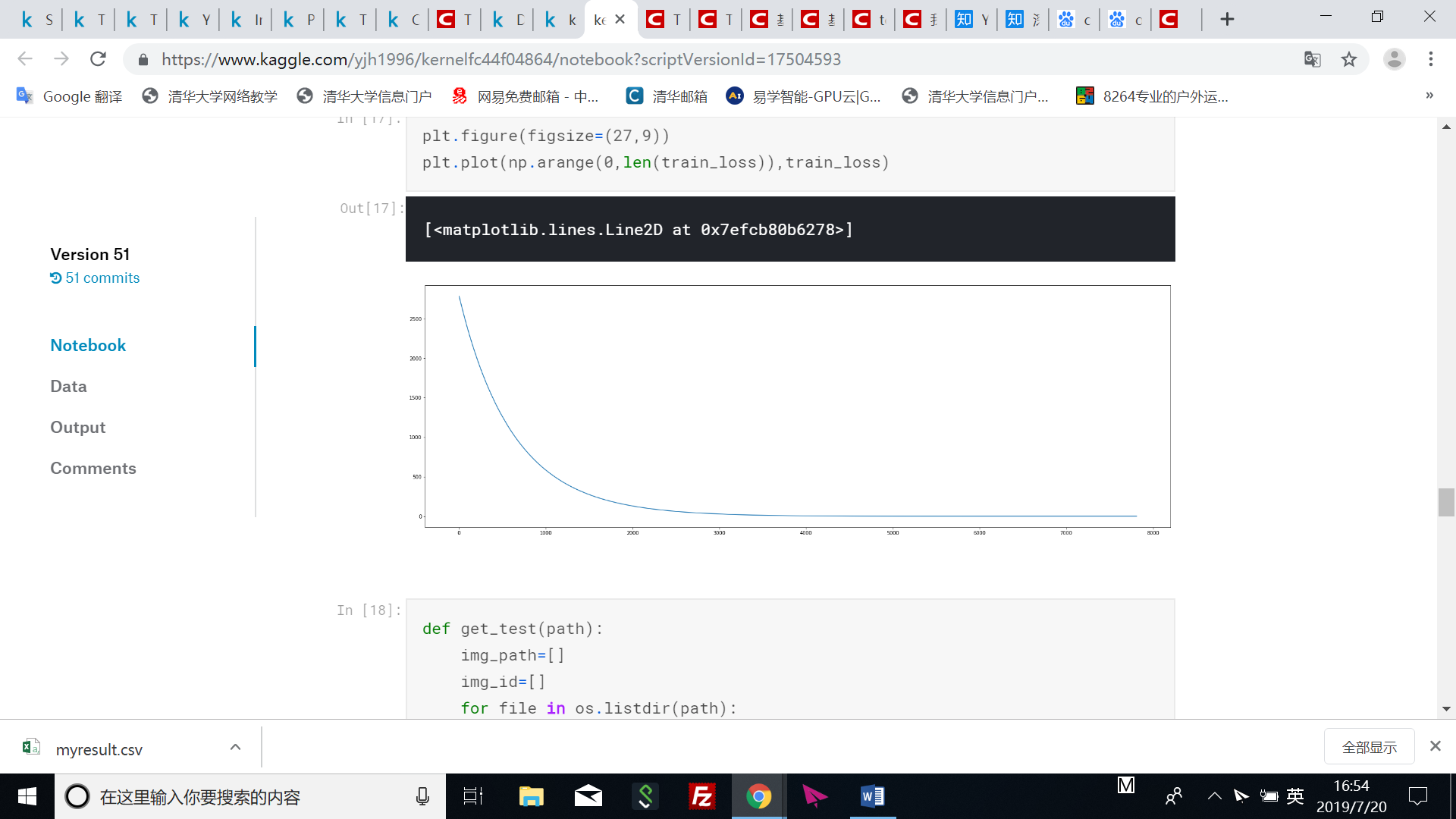
因为全连接层数据太多，我们的训练样本有点少，因此合理设置trainable参数，防止过拟合。（貌似不需要）

在数据增强后，loss抖动太大，是epochs太小的缘故吗？那先增大epoch试试。E=20，还是在抖动。E=40，效果也不太好。



图像**标准化与归一化**是不一样的。看来tensor与nparray都可以，只是tensor最后还是需要sess.run一下。

正则化还是很神奇的！



之前loss=0.69，最终靠正则化解决了。看来还是fc层weight太大。目前e=6，lambda=0.5 。

调整图像大小前先转换类型**.astype**

**Tf.convert\_image\_dtype**几乎相当于除以255.0

使用正则化+图像标准化，分数0.6+没有只用正则化0.1+好。可能是lambda太大了，减小试试。减小也没用。现在试试只标准化不正则化：和正则化+标准化相比差别不大，loss抖动得厉害。之后又试了aug（就是翻转）+正则化，结果也不好，减少epochs也不好。

**七、tf语音识别**

Paper：

<https://blog.csdn.net/ema1997/article/details/75222138>

<https://www.zouyong.cn/?p=943>

<https://blog.csdn.net/jialilian5181/article/details/83787886>

<https://blog.csdn.net/jialilian5181/article/details/83753050>

思路/技巧：

<https://zhuanlan.zhihu.com/p/64977527>

code：

<https://www.kaggle.com/alphasis/light-weight-cnn-lb-0-74>

<https://www.kaggle.com/voglinio/keras-directory-iterator-wav-to-spectogram>

关于数据集：

1.one speaker doesn't occur in both train and test sets.

2.There are also recordings with some weird silence

3.FFT looks different for every word

4. 训练集有31个label，test集包含其中10个label.除此之外，此外还有训练集没有的unkown,　silence

5. Train data's sampling rate is 16000Hz, but for making lower computation cost, many people choose to resample to 8000hz

6.int16编码

**常用前处理方法:**

**（1）Resampling** - dimensionality reduction

先检查是否有高频信息,　若没有，可以考虑在时间维度重新采样。但这样做会丢失信息。

def custom\_fft(y, fs):

T = 1.0 / fs

N = y.shape[0]

yf = fft(y)

xf = np.linspace(0.0, 1.0/(2.0\*T), N//2)

vals = 2.0/N \* np.abs(yf[0:N//2]) # FFT is simmetrical, so we take just the first half

# FFT is also complex, to we take just the real part (abs)

return xf, vals

若采样率为f，则采样后长度len=f

**（2）**VAD （**Silence removal**）

删掉开头结尾的静音。使用webrtcvad，<https://www.kaggle.com/holzner/voice-activity-detection-example>

**（3）**Maybe **padding with 0** to make signals be **equal length**

和NLP一样，为了能够供ＮＮ使用，需要样本长度一致，当然最好在这之前先做VAD

**Np.pad(array, pad\_width, mode, \*\*kwargs)**

返回值：数组

参数解释:

array——表示需要填充的数组；

pad\_width——表示每个轴（axis）边缘需要填充的数值数目。

参数输入方式为：（**(before\_1, after\_1)**, … (before\_N, after\_N)），其中(before\_1, after\_1)表示第1轴两边缘分别填充before\_1个和after\_1个数值。取值为：{sequence, array\_like, int}

mode——表示填充的方式（取值：str字符串或用户提供的函数）,总共有11种填充模式；

‘constant’——表示连续填充相同的值，每个轴可以分别指定填充值，constant\_values=（x, y）时前面用x填充，后面用y填充，缺省值填充0

……

**（4）**短时傅里叶分析**Log** **spectrogram** (or MFCC, or PLP)

**①**开始我认为语音是一维问题，应当和NLP类似，没想到其实和cv更接近，**基本主流做法**都是将语音通过logmelspectrum ,或mfcc转化成**图像**，然后输入到vgg.densenet,rsenet等经典模型里。spectrogram要好于mfcc 。在端到端（通常是基于神经网络的）系统中，最常见的**输入特征**可能是**原始频谱图raw spectrograms或梅尔功率频谱图raw spectrograms**。

**②**Note, that we are taking logarithm of spectrogram values. It will make our plot much more clear, moreover, it is strictly connected to the way people hear. We need to assure that there are no 0 values as input to logarithm 。

**③**在转化成图像的时候，记得做**归一化**

<https://www.kaggle.com/timolee/audio-data-conversion-to-images-eda>

<https://www.kaggle.com/ybonde/log-spectrogram-and-mfcc-filter-bank-example#Feature-extraction-using-librosa>

**（5）**Features **normalization** with mean and std

**（6）**Stacking of a given number of frames to get temporal information

**(7)**数据增强：

X=wavfile.read(x)[1]

if aug: #数据增强，目前只应用白噪声#应该是randn，有负数

wn = np.random.randn(len(x))

x = x + beta\*wn

**用到的库：**

1. **Scipy**可以处理插值、积分、优化、**图像处理**、常微分方程数值解的求解、**信号处理**等问题。它用于有效计算Numpy矩阵，使Numpy和Scipy协同工作，高效解决问题。本项目用于短时傅里叶分析。

**①fs, data = scipy.io.wavfile.read('./audio.wav')**

返回一个元组，第一项为音频的采样率，第二项为音频数据的numpy数组(可通过plt.plot(data)查看波形图)。

**②F,t,s=scipy.signal.spectrogram(data, fs=1.0, window=(‘tukey’, 0.25), nperseg=None, noverlap=None, nfft=None, detrend=’constant’, return\_onesided=True, scaling=’density’, axis=-1, mode=’psd’)**

Parameters:

data:输入的时间序列，可以用wavfile.read输出的np数组

fs:float型，采样频率，默认值为1.0

window:分窗大小，默认使用参数为0.25的Tukey window

**nperseg**:int型 窗口长度 ???

**noverlap**:int型 段与段之间的重叠面积 ???

nfft : int, optional

Length of the FFT used, if a zero padded FFT is desired. If None, the FFT length is nperseg. Defaults to None.

detrend : str or function or False, optional

指定如何去除每个段的趋势。 如果detrend是一个字符串，它将作为类型参数传递给detrend函数。 如果它是一个函数，它需要一个段并返回一个去趋势段。 如果detrend为False，则不会进行去趋势。 默认为'常数'

Returns：

f:频域信息

t:时域信息

Sxx:时频图，取转置

③**xf=scipy.fftpack.fft(x, n=None, axis=-1, overwrite\_x=False)**

Return 一维离散傅里叶变换discrete Fourier transform of real or complex sequence.

The returned complex array contains y(0), y(1),..., y(n-1) where y(j) = (x \* exp(-2\*pi\*sqrt(-1)\*j\*np.arange(n)/n)).sum()

**④resampled\_data=scipy.signal.resample(data, num, t=None, axis=0, window=None)**

使用FFT重采样num个点

⑤**scipy.io.wavfile.write(filename, rate, data)**

Write a numpy array as a WAV file.

Parameters

Filename：string or open file handle

Output wav file.

Rate:int

The sample rate (in samples/sec).

Data:ndarray

A 1-D or 2-D numpy array of either integer or float data-type.

| **WAV format** | **Min** | **Max** | **NumPy dtype** |
| --- | --- | --- | --- |
| 32-bit floating-point | -1.0 | +1.0 | float32 |
| 32-bit PCM | -2147483648 | +2147483647 | int32 |
| 16-bit PCM | -32768 | +32767 | int16 |
| 8-bit PCM | 0 | 255 | uint8 |

1. **Librosa**是一个用于音频、音乐分析、处理的python工具包，一些常见的时频处理、特征提取、绘制声音图形等功能应有尽有，功能十分强大。本项目用于MFCC。

**①y, sr = librosa.load(path, sr=22050, mono=True, offset=0.0, duration=None, dtype=<class 'numpy.float32'>, res\_type='kaiser\_best')**

Load an audio file as a floating point time series.

读取文件，可以是wav、mp3等格式

如果 sr 缺省，librosa会默认以22050的采样率读取音频文件，高于该采样率的音频文件会被下采样，低于该采样率的文件会被上采样。如果希望以原始采样率读取音频文件，sr 应当设为 None。具体做法为 y, sr = librosa(filename, sr=None)。

**②D=librosa.stft(audio, n\_fft=2048, hop\_length=None, win\_length=None, window='hann', center=True, dtype=<class 'numpy.complex64'>, pad\_mode='reflect')**

对audio做短时傅里叶变换stft

Returns a complex-valued matrix D such that：

**np.abs(D[f, t])** is the magnitude of frequency bin f at frame t

**np.angle(D[f, t])** is the phase of frequency bin f at frame t

**③Mfcc=librosa.feature.mfcc(audio=None, sr=22050, S=None, n\_mfcc=20, dct\_type=2, norm='ortho', \*\*kwargs)**

audio:np.ndarray [shape=(n,)] 或 None ；音频时间序列

sr:number > 0 [scalar] ；audio的采样率

S:np.ndarray [shape=(d, t)] or None ；对数功能梅尔谱图

n\_mfcc: int > 0 [scalar] ；要返回的MFCC数量

dct\_type:None, or {1, 2, 3} ；离散余弦变换（DCT）类型。默认情况下，使用DCT类型2。

norm:None or ‘ortho’ ；规范。如果dct\_type为2或3，则设置norm =’ortho’使用正交DCT基础。标准化不支持dct\_type = 1。

kwargs:额外的关键参数

参数melspectrogram，如果按时间序列输入操作

返回：

M:np.ndarray [shape=(n\_mfcc, t)]

MFCC序列

可以对返回结果进行np.mean(mfcc,axis=0)，或者plt.imshow(mfcc绘图)

**④**以下略：

librosa.feature.chroma\_stft

librosa.feature.melspectrogram

librosa.feature.spectral\_contrast

librosa.feature.tonnetz

⑤**log\_S=librosa.power\_to\_db(S, ref=1.0, amin=1e-10, top\_db=80.0)[source]**

Convert a power spectrogram (amplitude squared) to decibel (dB) units

This computes the scaling 10 \* log10(S / ref) in a numerically stable way.

⑥**librosa.display.specshow(data, x\_coords=None, y\_coords=None, x\_axis=None, y\_axis=None, sr=22050, hop\_length=512, fmin=None, fmax=None, bins\_per\_octave=12, ax=None, \*\*kwargs)[source]**

Display a spectrogram/chromagram/cqt/etc.

Parameters:

data:np.ndarray [shape=(d, n)] ，如log\_S

Matrix to display (e.g., spectrogram)

sr:number > 0 [scalar]

Sample rate used to determine time scale in x-axis.

hop\_length:int > 0 [scalar]

Hop length, also used to determine time scale in x-axis

x\_axis:None or str

y\_axis:None or str

**⑦delta\_mfcc=librosa.feature.delta(data, width=9, order=1, axis=-1, mode='interp', \*\*kwargs)[source]**

Compute delta features: local estimate of the derivative of the input data along the selected axis.

Delta features are computed Savitsky-Golay filtering.

Parameters:

data:np.ndarray

the input data matrix (eg, spectrogram)

width:int, positive, odd [scalar]

Number of frames over which to compute the delta features. Cannot exceed the length of data along the specified axis. If mode=’interp’, then width must be at least data.shape[axis].

order:int > 0 [scalar]

the order of the difference operator. 1 for first derivative, 2 for second, etc.

axis:int [scalar]

the axis along which to compute deltas. Default is -1 (columns).

mode:str, {‘interp’, ‘nearest’, ‘mirror’, ‘constant’, ‘wrap’}

Padding mode for estimating differences at the boundaries.

kwargs:additional keyword arguments

See scipy.signal.savgol\_filter

Returns:

delta\_data:np.ndarray [shape=(d, t)]

delta matrix of data at specified order

**⑧audio\_resample=librosa.core.resample(audio, orig\_sr, target\_sr, res\_type='kaiser\_best', fix=True, scale=False, \*\*kwargs)**

Resample a time series from orig\_sr to target\_sr

if f.endswith('.wav')

右侧线性部分使得ELU能够缓解梯度消失，而左侧软饱能够让ELU对输入变化或噪声更鲁棒

字典(Dictionary) **items()** 函数以列表返回可遍历的(键, 值) 元组数组

本次混淆矩阵之应用：

针对目前的问题，我打算首先画出混淆矩阵看看。（耗时太长！）

Pred=Model.predict\_generator 。返回pred格式是nparray，不过也存在读取顺序的关系。

这可比for循环快多了！

**不开vpn，数据都传不到kaggle上！**

总结一下：**dtype、astype、type**

type 获取数据类型。dtype 数组元素的类型。astype 修改数据类型。'list' object has no attribute 'dtype'

'list' object has no attribute 'dtype'

type(a)，a.dtype，a.astype(‘int32’)

为什么silence总会被分成unknown呢？仍然是0.64720，没有改观。是我们生产silence的方式不对（全0），还是e太大过拟合了？

E40->20，silence全对，应该是过拟合了。但只在训练集上改进了，测试集仍是0.64720 。另外将e缩减一半，只减少了十来分钟。

但是如果用backgroundnoise来产生，杂音也太大了。我现在最好奇，测试集silence是怎样的。用了backgroundnoise来产生，仍然是0.64720. 看来原因大概是网络瓶颈了，即使很好解决silence识别，其他类型的识别精度也达不到。

目前是哪个版本最好呢？（貌似是e=60那个）