题目：CNN猫狗图像识别

### 完成人：

SA19225318 宋根鹏

SA19225301 乔明超

SA19225298 钱皓雪

SA19225397 吴丛明

SA19225378 王炜康

### Demo运行

数据下载地址：https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data

AlexNet预训练文件下载地址:http://www.cs.toronto.edu/~guerzhoy/tf\_alexnet/

VGG16预训练文件下载地址：

<ftp://mi.eng.cam.ac.uk/pub/mttt2/models/vgg16.npy>

下载代码、数据集、预训练模型，根据数据集、预训练模型名称、路径修改代码中相应部分，运行Generate\_txt.py，util\_data.py，DataGenerator.py生成数据集相应的文件列表txt、csv等文件，然后运行run.py分别进行训练、评估、测试过程

### 二、项目背景

此项目主要是对深度学习网络模型的实践，主要是用Alexnet网络训练kaggle上的经典的分类项目“猫狗识别”，来了解Alexnet的实现，以及对深度学习的了解，能够加强我们对计算机视觉方面图像分类的认识，此外还应用了vgg16模型，比较Alexnet和vgg16两个模型在训练过程中的区别。

### 三、设计目标

实现构建Alexnet、VGGNet网络模型，可用于猫狗图片识别，解决kaggle上较为经典的猫狗分类问题。通过对数据集处理，借助训练验证过程及kaggle评分，比较Alexnet模型以及vgg16模型性能，对网络深度以及卷积核对训练精度以及时间的影响有所认识。使用迁移学习fine-tuning技术思想，学会使用预训练好的网络模型来训练数据。

### 三、技术路线

环境配置：tensorflow1.14

Python3.6

原码参考：

<https://blog.csdn.net/qq_16137569/article/details/72802387>

<https://github.com/steelOneself/kaggle/tree/master/cat_vs_dog/AlexNet>

1.数据的准备

数据包含两部分，训练集和测试集，训练集有25000张图片，测试集有12500张图片，在训练集的图片名称中包含了图片的标签信息，而测试集的图片名称代表的图片的ID。

将训练集中的猫狗数据做可视化处理，查看猫狗数据集的分布情况。观察数据集可以看到在训练集中猫狗各为12500张，并且前12500张为猫，后12500张为狗的图片，因为要从训练集中选取20%作为验证集，所以要对猫狗的图片做分层抽样，以保证验证集中的图片分布和训练集一致。

将图片信息保存为txt文件，保存格式为：图片ID，图片路径，图片标签。

将数据分为训练集、验证集、测试集。

加载数据，在读取图片的时候需要将图片转为227x227大小，因为Alexnet要求输入图片的大小是227x227。

2.其次，构建Alexnet模型，使用训练集训练。

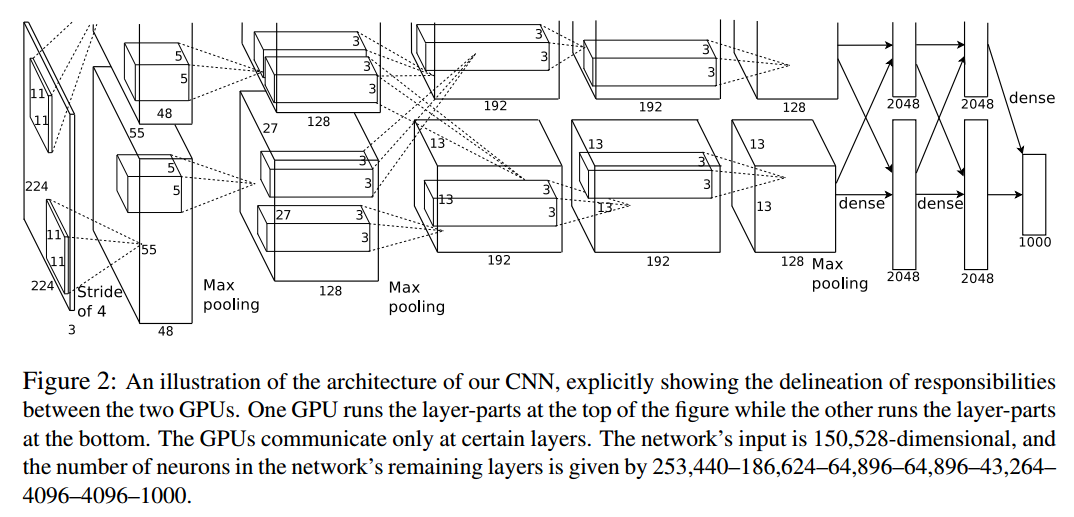
3.最后，模型评估，查看训练好模型的准确率，比较模型性能。

### 四、关键原理

此项目主要是使用Alexnet网络和vgg16网络模型进行猫狗识别的训练。主要为Alexnet网络和vgg16网络的认识和实践。

Alexnet：

网络模型：



从网络模型中可以看出，模型分为上下两部分，这两部分分别对应两个GPU，只有到了特定的网络层后才需要两块GPU进行交互，设置的目的主要是利用两块GPU来提高运算的效率。

网络主要分为8层，5层卷积，3层全连接层。

第一层：卷积层1，输入为 224×224×3的图像，卷积核的数量为96，两片GPU分别计算48个核; 卷积核的大小为 11×11×3; stride = 4, stride表示的是步长， pad = 0, 表示不扩充边缘;

第二层：卷积层2, 输入为上一层卷积的feature map， 卷积的个数为256个，两个GPU分别有128个卷积核。卷积核的大小为：5×5×48; pad = 2, stride = 1; 然后做 LRN， 最后 max\_pooling, pool\_size = (3, 3), stride = 2;

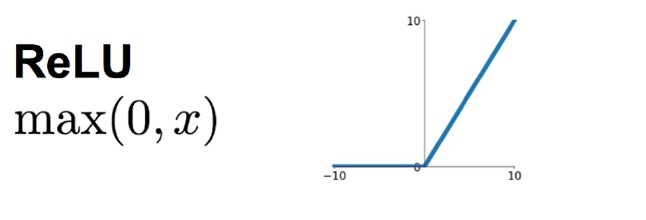
第三层：卷积3, 输入为第二层的输出，卷积核个数为384, kernel\_size = (3×3×256)， padding = 1, 第三层没有做LRN和Pool

第四层：卷积4, 输入为第三层的输出，卷积核个数为384, kernel\_size = (3×3), padding = 1, 和第三层一样，没有LRN和Pool

第五层：卷积5, 输入为第四层的输出，卷积核个数为256, kernel\_size = (3×3), padding = 1。然后直接进行max\_pooling, pool\_size = (3, 3), stride = 2;

第6,7,8层是全连接层，每一层的神经元的个数为4096，最终输出softmax为1000,因为上面介绍过，ImageNet这个比赛的分类个数为1000。全连接层中使用了RELU和Dropout。

Alexnet第一次使用了relu作为激活函数。ReLU是一个分段线性函数，小于等于0则输出为0；大于0的则恒等输出。相比于sigmoid，ReLU有以下特点：

* 计算开销下。sigmoid的正向传播有指数运算，倒数运算，而ReLu是线性输出；反向传播中，sigmoid有指数运算，而ReLU有输出的部分，导数始终为1.
* 梯度饱和问题
* 稀疏性。Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生。

**局部相应归一化**

ReLU具有让人满意的特性，它不需要通过输入归一化来防止饱和。如果至少一些训练样本对ReLU产生了正输入，那么那个神经元上将发生学习。然而，我们仍然发现接下来的局部响应归一化有助于泛化。表示神经元激活，通过在(x,y)位置应用核i，然后应用ReLU非线性来计算，响应归一化激活通过下式给定：

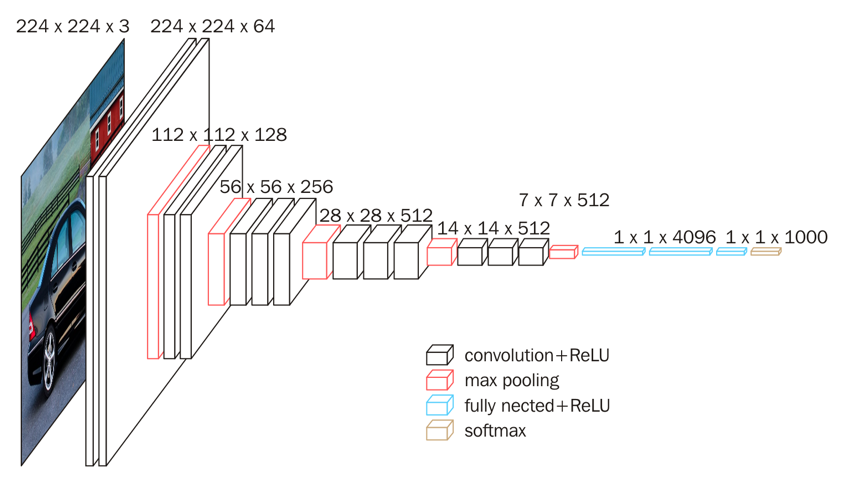
其中，N是卷积核的个数，也就是生成的FeatureMap的个数；k,α,β,n是超参数，论文中使用的值是k=2,n=5,α=10−4,β=0.75,输出和输入的上标表示的是当前值所在的通道，也即是叠加的方向是沿着通道进行。将要归一化的值所在附近通道相同位置的值的平方累加起来

**Dropout**

这个是比较常用的抑制过拟合的方法了。  
 引入Dropout主要是为了防止过拟合。在神经网络中Dropout通过修改神经网络本身结构来实现，对于某一层的神经元，通过定义的概率将神经元置为0，这个神经元就不参与前向和后向传播，就如同在网络中被删除了一样，同时保持输入层与输出层神经元的个数不变，然后按照神经网络的学习方法进行参数更新。在下一次迭代中，又重新随机删除一些神经元（置为0），直至训练结束。

vgg16:

网络模型：



VGG16共包含：

* 13个卷积层（Convolutional Layer），分别用conv3-XXX表示
* 3个全连接层（Fully connected Layer）,分别用FC-XXXX表示
* 5个池化层（Pool layer）,分别用maxpool表示

其中，卷积层和全连接层具有权重系数，因此也被称为权重层，总数目为13+3=16，这即是VGG16中16的来源。(池化层不涉及权重，因此不属于权重层，不被计数)。

VGG16的突出特点是简单，体现在：

1. 卷积层均采用相同的卷积核参数

卷积层均表示为conv3-XXX，其中conv3说明该卷积层采用的卷积核的尺寸(kernel size)是3，即宽（width）和高（height）均为3，3\*3是很小的卷积核尺寸，结合其它参数（步幅stride=1，填充方式padding=same），这样就能够使得每一个卷积层(张量)与前一层（张量）保持相同的宽和高。XXX代表卷积层的通道数。

1. 池化层均采用相同的池化核参数

池化层的参数均为2×2，步幅stride=2，max的池化方式，这样就能够使得每一个池化层（张量）的宽和高是前一层（张量）的1/2。

1. 模型是由若干卷积层和池化层堆叠（stack）的方式构成，比较容易形成较深的网络结构（在2014年，16层已经被认为很深了）。

### 五、具体实现

1、搭建网络

Alexnet：

①定义网络层函数：

'''

卷积函数

'''

def conv(x,filter\_height,filter\_width,num\_filters,stride\_y,stride\_x,

name,padding="SAME",groups=1):

#获取输入tensor的channel

input\_channels = int(x.get\_shape()[-1])

#创建一个lambda函数

convolve=lambda i,k:tf.nn.conv2d(i,k,

strides=[1,stride\_y,stride\_x,1],padding=padding)

with tf.variable\_scope(name) as scope:

#定义权重

weights= tf.get\_variable("weights",

shape=[filter\_height,filter\_width,

input\_channels/groups,num\_filters])

#定义偏置

biases = tf.get\_variable("biases",shape=[num\_filters])

if groups == 1:

conv = convolve(x,weights)

else:

input\_groups= tf.split(axis=3,

num\_or\_size\_splits=groups,value=x)

weight\_groups= tf.split(axis=3,

num\_or\_size\_splits=groups,value=weights)

output\_groups=[convolve(i,k) for i,k in zip(input\_groups,weight\_groups)]

#连接卷积层

conv = tf.concat(axis=3,values=output\_groups)

bias = tf.reshape(tf.nn.bias\_add(conv,biases),tf.shape(conv))

#relu激活函数

relu = tf.nn.relu(bias,name=scope.name)

return relu

'''

'''

全连接层函数

'''

def fc(x,num\_in,num\_out,name,relu=True):

with tf.variable\_scope(name) as scope:

#定义权重和偏置

weights= tf.get\_variable("weights",shape=[num\_in,num\_out],

trainable=True)

biases = tf.get\_variable("biases",[num\_out],trainable=True)

fc\_out = tf.nn.xw\_plus\_b(x,weights,biases,name=scope.name)

if relu:

fc\_out = tf.nn.relu(fc\_out)

return fc\_out

'''

最大池化层函数

'''

def max\_pool(x,filter\_height,filter\_width,stride\_y,stride\_x,name,padding="SAME"):

return tf.nn.max\_pool(x,ksize=[1,filter\_height,filter\_width,1],

strides=[1,stride\_y,stride\_x,1],padding=padding,name=name)

'''

lrn层

'''

def lrn(x,radius,alpha,beta,name,bias=1.0):

return tf.nn.local\_response\_normalization(x,depth\_radius=radius,

alpha=alpha,beta=beta,bias=bias,name=name)

'''

dropout层

'''

def dropout(x,keep\_prob):

return tf.nn.dropout(x,keep\_prob)

②创建Alexnet网络：

def create(self):

#第一层卷积

conv1 = conv(self.X,11,11,96,4,4,padding="VALID",name="conv1")

norm1 = lrn(conv1,2,2e-05,0.75,name="norm1")

pool1 = max\_pool(norm1,3,3,2,2,padding="VALID",name="pool1")

#第二层卷积

conv2 = conv(pool1,5,5,256,1,1,groups=2,name="conv2")

norm2 = lrn(conv2,2,2e-05,0.75,name="norm2")

pool2 = max\_pool(norm2,3,3,2,2,padding="VALID",name="pool2")

#第三层卷积

conv3 = conv(pool2,3,3,384,1,1,name="conv3")

#第四层卷积

conv4 = conv(conv3,3,3,384,1,1,groups=2,name="conv4")

#第五层卷积

conv5 = conv(conv4,3,3,256,1,1,groups=2,name="conv5")

pool5 = max\_pool(conv5,3,3,2,2,padding="VALID",name="pool5")

#第六层,全连接层

flattened = tf.reshape(pool5,[-1,6\*6\*256])

fc6 = fc(flattened,6\*6\*256,4096,name="fc6")

dropout6 = dropout(fc6,self.KEEP\_PROB)

#第七层，全连接层

fc7 = fc(dropout6,4096,4096,name="fc7")

dropout7 = dropout(fc7,self.KEEP\_PROB)

#第八层,全连接层

self.fc8 = fc(dropout7,4096,self.NUM\_CLASSES,

relu=False,name="fc8")

③VGG16：

#第一层

conv1\_1 = conv(self.X,3,3,64,1,1,padding="VALID",name="conv1\_1")

conv1\_2 = conv(conv1\_1, 3, 3, 64, 1, 1, padding="VALID", name="conv1\_2")

pool1 = max\_pool(conv1\_2,2,2,2,2,padding="VALID",name="pool1")

#第二层

conv2\_1 = conv(pool1,3,3,128,1,1,groups=2,name="conv2\_1")

conv2\_2 = conv(conv2\_1, 3, 3, 128, 1, 1, groups=2, name="conv2\_2")

pool2 = max\_pool(conv2\_2,2,2,2,2,padding="VALID",name="pool2")

#第三层

conv3\_1 = conv(pool2,3,3,256,1,1,name="conv3\_1")

conv3\_2 = conv(conv3\_1, 3, 3, 256, 1, 1, name="conv3\_2")

conv3\_3 = conv(conv3\_2, 3, 3, 256, 1, 1, name="conv3\_3")

pool3 = max\_pool(conv3\_3, 2, 2, 2, 2, padding="VALID", name="pool3")

#第四层

conv4\_1 = conv(pool3,3,3,512,1,1,name="conv4\_1")

conv4\_2 = conv(conv4\_1, 3, 3, 512, 1, 1, name="conv4\_2")

conv4\_3 = conv(conv4\_2, 3, 3, 512, 1, 1, name="conv4\_3")

pool4 = max\_pool(conv4\_3, 2, 2, 2, 2, padding="VALID", name="pool4")

#第五层

conv5\_1 = conv(pool4,3,3,512,1,1,name="conv5\_1")

conv5\_2 = conv(conv5\_1, 3, 3, 512, 1, 1, name="conv5\_2")

conv5\_3 = conv(conv5\_2, 3, 3, 512, 1, 1, name="conv5\_3")

pool5 = max\_pool(conv5\_3, 2, 2, 2, 2, padding="VALID", name="pool5")

#第六层,全连接层

flattened = tf.reshape(pool5,[-1,7\*7\*512])

fc6 = fc(flattened,7\*7\*512,4096,name="fc6")

dropout6 = dropout(fc6,self.KEEP\_PROB)

#第七层，全连接层

fc7 = fc(dropout6,4096,4096,name="fc7")

dropout7 = dropout(fc7,self.KEEP\_PROB)

#第八层,全连接层

self.fc8 = fc(dropout7,4096,self.NUM\_CLASSES,relu=False,name="fc8")

2.数据处理

将数据集图片信息写入txt文件中，制作文件列表

with open(save\_txt\_path,mode="w",encoding="utf-8") as f\_wirter:

for img\_name in os.listdir(train\_img\_dir):

img\_path = os.path.join(train\_img\_dir,img\_name).replace("\\","/")

if img\_path.endswith("jpg"):

if save\_mode == "train":

#获取图片的标签名cat或者dog

label\_name,img\_id,\_ = img\_name.split(".")

#将图片信息写入到txt文件中,使用逗号进行分割

f\_wirter.write("%s,%s,%s\n"%(img\_id,img\_path,label\_name))

else:

#获取图像的id

img\_id,\_ = img\_name.split(".")

#写入文件，图像的id和路径

f\_wirter.write("%s,%s\n"%(img\_id,img\_path))

f\_wirter.close()

②根据文件列表，划分数据集，首先将训练集按猫狗划分，打乱顺序后按设置比例抽取相同数量的猫狗图片构成验证集，将训练集和验证集文件信息保存到csv文件中

group\_data = data.groupby("img\_label")

print('aaa')

print(group\_data)

for label\_num,data in group\_data:

if label\_num == 0:

cat\_dataset = data

print(label\_num)

if label\_num == 1:

dog\_dataset = data

#打乱顺序

print(img\_labels)

print(cat\_dataset)

cat\_dataset = shuffle(cat\_dataset)

dog\_dataset = shuffle(dog\_dataset)

val\_num = int(len(img\_ids) \* val\_size)

val\_dataset = shuffle(pd.concat([cat\_dataset.iloc[:val\_num,:],dog\_dataset.iloc[:val\_num,:]],axis=0))

train\_dataset = shuffle(pd.concat([cat\_dataset.iloc[val\_num:,:],dog\_dataset.iloc[val\_num:,:]],axis=0))

#将训练集文件和测试集文件保存为csv文件

train\_dataset.to\_csv("txt/train.csv")

val\_dataset.to\_csv("txt/val.csv")

return train\_dataset,val\_dataset

③加载图片预处理，标签编码，图片resize，加载图片

#标签转为one-hot编码

one\_hot = tf.one\_hot(label,self.num\_classes)

#加载图片的预处理

img\_string = tf.read\_file(filename)

img\_decode = tf.image.decode\_jpeg(img\_string,channels=3)

img\_resized = tf.image.resize\_images(img\_decode,[227,227])

return img\_resized,one\_hot

④训练过程主要代码

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

writer.add\_graph(sess.graph)

model.load\_initial\_weights(sess)

#记录最好的验证准确率

best\_val\_acc = 0.9

print('开始训练')

#迭代训练

for epoch in range(num\_epochs):

sess.run(training\_init\_op)

for step in range(train\_batches\_per\_epoch):

# print('0batch')

img\_batch,label\_batch = sess.run(next\_batch)

sess.run(train\_op,feed\_dict={x:img\_batch,y:label\_batch,keep\_prob:dropout\_rate})

if step % display\_step == 0:

s,train\_acc,train\_loss = sess.run([merged\_summary,accuracy,loss],

feed\_dict={x:img\_batch,y:label\_batch,keep\_prob:1.0})

writer.add\_summary(s,epoch\*train\_batches\_per\_epoch+step)

sess.run(val\_init\_op)

#统计验证集的准确率

val\_acc = 0

#统计验证集的损失值

val\_loss = 0

test\_count = 0

for \_ in range(val\_batches\_per\_epoch):

img\_batch,label\_batch = sess.run(next\_batch)

acc,val\_batch\_loss = sess.run([accuracy,loss],feed\_dict={x:img\_batch,y:label\_batch,keep\_prob:1.0})

val\_acc += acc

val\_loss += val\_batch\_loss

test\_count += 1

val\_acc /= test\_count

val\_loss /= test\_count

print("%s epoch:%d,train acc:%.4f,train loss:%.4f,val acc:%.4f,val loss:%.4f"

%(datetime.now(),epoch+1,train\_acc,train\_loss,val\_acc,val\_loss))

if val\_acc > best\_val\_acc:

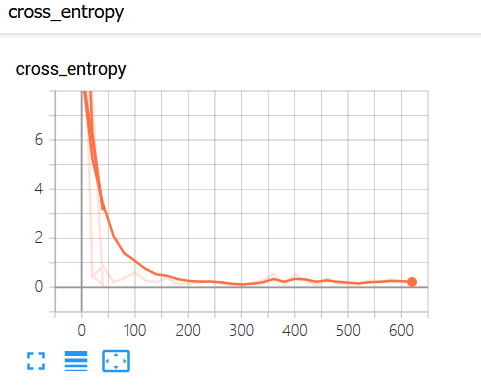
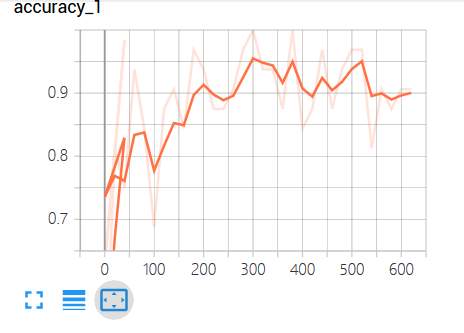
checkpoint\_name = os.path.join(checkpoint\_path,"model\_epoch%s\_%.4f.ckpt"%(str(epoch+1),val\_acc))

saver.save(sess,checkpoint\_name)

best\_val\_acc = val\_acc

### 五、结果展示

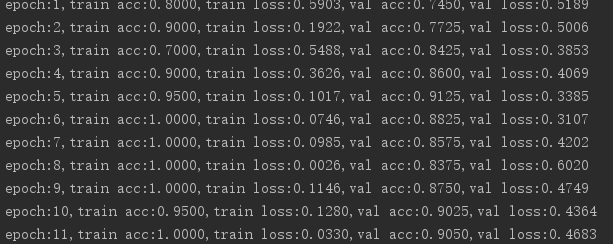
1.使用25000张图片的完整数据集进行训练，在AlexNet上完成训练、评估、测试，得到结果：准确率及损失曲线如下：

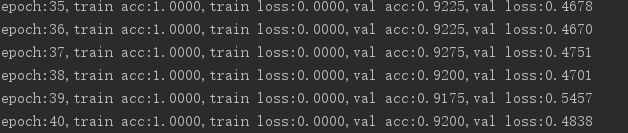


采用完整训练集，共25000张图片，batchsize设为32，这样一个epoch即为625步，一个epoch后验证集上准确率就可以达到0.9187,5轮epoch后准确率可以达到0.97以上

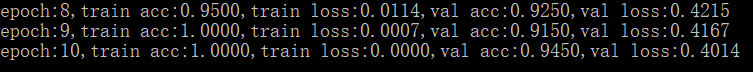
由于VGG16网络较深，全连接层节点数多达4096，受算力限制，没有完成在完整数据集上VGG16网络的训练。

考虑减小数据集规模，再次进行训练，训练集中选取1000张猫图片1000张狗图片构成2000张图片的数据集，再次进行训练，得到AlexNet网络结果：





可以观察到，在训练较少轮数后训练集上准确率就达到100%，而验证集准确率最高达到0.9275，与使用完整训练集的结果有差距，这也是由于减少了训练数据集规模而产生的。



使用VGG16网络，经过训练后训练集上准确率也到达100%，验证集上最高可得到0.945的准确率，可以观察到在充分训练后，VGG16网络性能是优于AlexNet网络的。

但需要注意的是，在上面的结果中，虽然VGG16优于AlexNet，但两者性能差距并不大，两个网络都没有训练到最优性能，这是由于实际上两个网络都是存在一定的过拟合现象的，在较小验证集上准确率还可以接受，但当网络扩展使用完整12500张测试集测试时，网络的表现并不好，出现了过拟合现象。下面使用kaggle评分用以参考，







其中，第三个是使用完整数据集训练的AlexNet，可见减小数据集规模后两个网络在大规模的测试集上表现都并不好，但VGG16还是优于AlexNet的

### 六、存在的问题

首先，减小数据集规模后，网络出现了过拟合现象。标准的AlexNet应该使用两块GPU并行分别训练，这样可能会有更好的性能，而本项目中使用的是较为广泛的替代方法，只用一个GPU训练。 VGGNet由于较深的网络结构，决定了必须有大量的训练数据，虽然在项目中使用fine-tuning技术，但主要是加快模型训练速度，无法避免小数据集造成的过拟合。为了验证两个网络的性能差距，实际上应该采用完整的大规模数据集，训练标准的AlexNet和VGG16，但由于算力限制，最终没能实现完整数据集上的训练

### 七、贡献率

SA19225318 宋根鹏完成网络编写、系统构建等部分，贡献率30%

SA19225301 乔明超完成训练评估编写、文档编写等部分贡献率20%

SA19225298 钱皓雪

SA19225397 吴丛明

SA19225378 王炜康 三人完成数据处理、系统完善等部分，贡献率均分50%

### 完成时间：2019.11.23