# 驾驶员状态检测

## 一、 项目概述

本人选择的项目将训练出一个可以识别司机当前状态的分类器，可以从输入的图片中判断出司机当前处于一个什么样的状态，是否是安全的驾驶行为。

这个项目主要为计算机视觉相关，主要涉及到了图像预处理、图像检测以及图像识别方面的理论与应用技巧，计算机视觉研究如何让计算机从图像和视频中获取高级、抽象的信息。前两年随着深度学习的兴起，此类相关技术更是得到了极大发展，各种学术研究比较深入，工业应用模型发展得较为快速和成熟，在比如自动驾驶，医疗图像，安全监控等多个方面都有着广泛的应用前景。

计算机的视觉主要可以简单分为三大类：分类（classification）、检测（detection）、分割（segmentation）。针对于本项目来说，为一个在简单场景下的图像分类问题，本人打算采用深度神经网络来完成此项目，在实践中巩固之前学习的关于深度学习和机器学习的知识。

我打算使用的卷积神经网络（Convolutional Neural Network,CNN）是一种近年发展起来，并引起广泛重视的一种高效图像识别方法。20 世纪 60 年代，Hubel 和 Wiesel 在研究猫脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网络（Convolutional Neural Networks-简称 CNN）。现在，CNN 已经成为众多科学领域的研究热点之一，业界已经有了众多优秀的成熟架构的卷积网络模型，我打算利用深度学习中的迁移学习的方法，充分利用优秀模型架构上的优点和特征提取能力，来完成此项目

## 二、 问题陈述

前文已经说过，此项目将使用深度学习的方法检测驾驶员的状态，即训练深度神经网络模型作为分类器进行给定图像的分类，具体的实际问题为给出了一个含有若干分类的驾驶员的不同的状态的彩色图片，驾驶员的状态即分类数量有 10 种，需要通过给定训练集的训练，生成一个图片分类器，来对测试集的

图片进行分类，即输入一个测试集的驾驶员的当前操作图片，预测出驾驶员此刻的驾驶状态，也就是这张图片属于每种分类的具体概率数字。

这个项目曾经是 Kaggle 上的一个竞赛，在比赛中，官方给出的评价方法为 multi-class logarithmic loss（多分类对数损失函数），此函数计算的结果为 Kaggle 比赛的最终排名依据，根据优达学城的要求，要求此项目必须排名在前 10%之内。

要解决这个问题，需要使用正确的方式处理训练数据集，建立一个分类器学习模型，利用训练数据集对这个分类器进行训练，同时对参数进行调试，来使得这个分类器的准确率和泛化能力达到一个好的效果，之后，就可以使用这个分类器对测试数据进行预测，来判断测试数据中驾驶员的行为。

前文已经说明，我打算使用的是深度学习里的卷积神经网络，利用卷积神经网络对于静态图片的特征提取能力，来达到对给定图片分类的效果。

大致的操作方式为建立一个卷积神经网络，将训练集图片读取为数字向量，同时按照分类生成训练集的独热编码标签，将这二者送入模型，设置好各种参数让其进行学习，控制拟合程度，提高预测精确度，来达到最终的目的。

因为现在有很多业界优秀的卷积网络模型，所以，我不打算自己从头搭建卷积网络模型，而是采用迁移学习的办法，直接使用别人搭建完毕的优秀模型结构，在 ImageNet 上已经训练好的权重的基础上，通过 fine-tuning 方法，开放一定的层数，输入我自己的训练数据，进行再次训练，让本来就已经具有优秀特征提取能力的模型来更加拟合自己的实际问题。

因为此次项目侧重于考察实现效果，所以，我不打算使用 tensorflow 来进行编写，而是使用 keras 这种更高级的库，这样，更利于我快速的调整模型和项目方案。

## 三、 评价指标

在训练中，自己的检查标准，参考值为 keras 模型在训练时的训练集 loss 值与 val_loss，应该将这两个值通过模型的训练，以及参数的调整，尽量的越小越好，并且下降的曲线应该是平滑的，并且逐渐降低至完全收敛。这个是在训练过程中，评估模型效果的标准之一，因为我觉得这个项目的训练集比较有限，自己

无法组织比较有效果或者有代表性的验证集，所在验证集上的最后得分不一定能够完全衡量测试集的有效得分，在加上本人选择的模型为 InceptionResNetV2，是一个层数非常深的模型，因此，在硬件设备的限制下，只能大致的判断模型是否达到了一个理想的收敛条件，所以，综上，最好的判断模型效果的方式是通过将结果上传至 Kaggle，通过最终测试集的得分来判断自己的模型处于一个什么水平。

所以，我没有给自己设定具体的评价得分指标，而是参考训练中的 loss 值与 val_loss，依靠最终 Kaggle 上的得分为准，我觉得这样更能够衡量模型的具体效果。

所以，我评估效果时，直接将模型生成的结果上传至 Kaggle 中，查看自己的 private sorce 和 public sorce 的分数值，查看自己的排名情况，以此来确定自己的参数已经方案选择是否恰当。

此项目 Kaggle 官方给出的评价函数为 multi-class logarithmic loss（多分类对数损失函数），具体计算公式如下：

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

此函数计算结果分为用于 Kaggle 上的最终排名，根据优达学城的要求，此项目必须排名在 10%之内，即名次最少要为 144 名，Private Leaderboard 的得分需要小于 0.25634。

## 四、 数据的探索

经过我的考察，此项目的训练数据为 Kaggle 所提供的一个包含 22424 张图片的文件夹，图片分辨率一样，都为 JPG 格式，内容基本都为一个司机在开车时某一时刻的状态。并且将这 20000 多张图片，已经按照定义好的十个动作，分散在十个文件夹内，每个文件夹中的图片都是同一个动作状态。还有一个 csv 文件，包含的是图片与司机的对应关系，还有一个提交结果的示例 csv 文件。

这个训练数据集一个含有 26 名司机，在对数据集的分析中，我发现每张图

片都是从一段短视频中截取出来的，可以按照司机把他们拼凑成一个完整的短视频。

此项目的测试数据是一个 79726 张的图片，也是通过视频抽帧出来的，但不是连续的，也没有按照动作状态分类，但测试数据和训练数据的背景非常相似。

- **所提供数据集相关统计信息如下：**

```
train_pic_category = len(os.listdir(trainPath))
train_pic_nums = [len(os.listdir(trainPath+path))for path in os.listdir(trainPath)]
img = cv2.imread(trainPath+'c0/img_375.jpg')

print('所提供数据集分为训练数据集和测试数据集,一共包含有26名不同的司机，其中:')
print('每张图片的格式为：width: %d height: %d channel: %d'%(img.shape[1], img.shape[0], img.shape[2]))

print('  1、训练数据集中共%d张图片，有%d个类别，数量分别为:'%(sum(train_pic_nums),train_pic_category))
for path in os.listdir(trainPath):
    print('     类别%s,数量为：%d'%(path,len(os.listdir(trainPath+path))))

print('  2、测试数据集中共%d张图片'%(len(os.listdir(testPath))))
```

```
所提供数据集分为训练数据集和测试数据集,一共包含有26名不同的司机，其中:
每张图片的格式为：width: 640 height: 480 channel: 3
  1、训练数据集中共22424张图片，有10个类别，数量分别为:
     类别c0,数量为：2489
     类别c1,数量为：2267
     类别c2,数量为：2317
     类别c3,数量为：2346
     类别c4,数量为：2326
     类别c5,数量为：2312
     类别c6,数量为：2325
     类别c7,数量为：2002
     类别c8,数量为：1911
     类别c9,数量为：2129
  2、测试数据集中共79726张图片
```

要获取这些数据，我初步想法是利用 opencv 将这些图片读取为数字向量，然后才可以变成输入模型的数据，并且根据我所选择的模型不同，我可能还需要对这些数据向量进行维度、归一化、调整标准正态分布等一系列的预处理，具体的情况还要看我所使用的模型。

我可能还需要在训练集中按照司机（不按照司机来进行抽取的话，有可能训练集和抽取的验证集的样本是基本一致的，模型本身就是高度拟合的，就失去了验证的意义），随机的抽取出若干验证集出来，用于验证和反馈我在训练时的效果，便于我调整模型。等到分类器模型建立完毕之后，我利用训练好的模型，对测试集的数据进行预测，生成测试集数据属于每一个类别的概率数据，按照提交结果要求，将结果上传至 Kaggle, 查看我的得分和排名。

- **训练样本概览**

我随机选取了一些示例的图片，将他们抽取出来进行了查看，以下为代码和显示结果：

```
#训练集的每种类别的图片示例为
pathList = []
for (root, dirs, files) in os.walk(trainPath):
    for dirc in dirs:
        file = os.listdir(os.path.join(root,dirc))[0]
        pathList.append((dirc,root+dirc+'/'+file))
plt.figure(figsize=(20,5)) #设置窗口大小
plt.suptitle('train data pics') # 图片名称
for i,path in enumerate(pathList):
    plt.subplot(2,5,i+1), plt.title(path[0])
    img = plt.imread(path[1])
    plt.imshow(img), plt.axis('off')
plt.show()
```


## ➢ 训练样本组成视频

同时，我在查看这些训练数据集中发现，训练集中一共有 26 位司机，他们的这些图片好像都是从一个连续的短视频中逐帧截取下来的，于是，我做了一下简单的探索，尝试将这些短视频进行了恢复，随机选取了两位司机，恢复的结果如下：

```
Writing GIF: p035
Writing GIF: p039
 动态图片已经生成!!!
<matplotlib.figure.Figure at 0x232834f5c0>
```

**生成的视频动态图：**

因 word 不能实时显示动态图，所以动态的图片可以在 HTML 文件中看到。所以，不同的学习图片是一段短视频的连续截图，我觉得这样对于模型的学习效果来说有一定影响，同一位司机特征差别不大的图片对于模型的泛化能力来说，没有很好的帮助，需要采取一定的手段来消除这种影响，后续的模型实现过程中，训练集的这种特点也得到了很明显的体现。同时，对训练集和测试集进行比较，训练集的数量较之测试集要小得多，所以，这个项目，在提高模型的准确度的同

时，我认为对于模型的泛化能力的提升是最重要的。

# 五、 算法和技术

## 1. 图像分类算法

本项目简单来说为一个静态图像的分类问题，属于计算机视觉技术中的一种，是计算机视觉、模式识别与机器学习领域非常活跃的研究方向，计算机的视觉技术大致有分类（classification）、检测（detection）、分割（segmentation）等。图像分类在很多领域都有着广泛的应用，比如安防领域的人脸识别、行人检测、交通领域的交通场景物体识别、、逆行检测，以及互联网领域的基于内容的图像检索、相册自动归类等。

图像的分类与人眼进行图像分类不同的是，他不是直接对于图像本身内容进行分类，而是通过一定的手段提取图像属性的向量化特征之后，然后再通过传统的机器学习方法（SVM、决策树等），或者是近年来新兴的深度学习方法（物体检测网络、CNN 等）来理解图片内容，做出分类决策。

## 2. 深度神经网络

收到人类大脑结构的启发，神经网络的计算模型与 1943 年提出，之后感知机的发明使得人工神经网络成为可以从数据中“学习的模型”，随着近年来，云计算和海量数据的普及，神经网络以“深度学习”的名字在很多领域都突破了传统机器学习的瓶颈，在计算机视觉、语音、自然语言等多个领域大放异彩。

深度神经网络的基本组成单位为神经元节点，他们逐层互相连接，每一个神经元节点获取其他节点的输入，神经元拥有自己的权重参数，获取输入然后进行非线性计算，通过激活函数得到输出，作为下一个神经元的输入。

神经元以不同的组成方式组成完整的网络，而他们的结构则有两个总体原则：非线性和多层结构。从总体上看，神经网络的解决问题过程就是一个优化过程，而损失函数则刻画了网络需要优化的目标，在优化神经网络时，最常用的就是梯度下降算法和反向传播算法。而在优化的过程中，需要通过调整网络结构、调整学习率、使用正则化手段等一系列方法来调整整个模型训练的速度、拟合度等等。

利用神经网络来解决分类问题一般分为如下 4 个步骤：1、提取问题中实体

的特征向量作为神经网络的输入 2、定义神经网络的结构，并且定义如何从神经网络的输入得到输出。3、通过训练数据来调整神经网络中参数的取值，这个就是训练神经网络的过程。4、使用训练好的神经网络来预测未知的数据。

## 3. 卷积神经网络

我打算使用的卷积神经网络（Convolutional Neural Network,CNN），他是目前在图像处理方面比较流行的深度学习模型方法之一，它主要由卷积层、池化层、全连接层组成，通过卷积逐层提取特征，最终由若干个全连接层完成图像的识别或者分类，个人觉得 CNN 网络的主要特点有权重共享和局部连接等，减少了网络中大量无用的参数和计算量，保留了重要的特征，来达到更好的训练效果

## 4. Tensorflow

Tensorflow 是由 Jeff Dean 领头的谷歌大脑团队基于谷歌第一代深度学习系统 DistBelief 改进而来的通用计算框架，如今，在谷歌的语音搜索、广告、电商、图片、街景图、翻译、YouYube 等众多的团队都在使用基于 Tensorflow 的系统，DeepMind 团队也将他们所有的研究都转移至 tensorflow 之上，如今，他是世界上最流行的深度学习框架，得到了广泛的应用。

Tensorflow 是一种采用数据流图，进行数值计算的框架系统，计算图（tf.Graph）、张量（tf.Tensor）和会话（tf.Session）是 Tensorflow 最基本的概念，计算图是 Tensorflow 的计算模型，所有程序都会以计算图的形式表示，每一个节点都是一个运算，计算图还标识了数据传递关系、每个设备的运行信息、运算之间的依赖挂你，还能管理不同的集合。而张量则是 Tensorflow 的数据模型，所有运算的输入输出都是张量。张量本身不存储任何数据，他只是对运算结果的引用。会话则是 Tensorflow 的运算模型，他管理了一个 Tensorflow 程序拥有的系统资源，所有的运算都要通过会话进行。

Tensorflow 程序一般可以分为两个阶段，在第一个阶段需要定义计算图的所有计算，第二个阶段为执行阶段。

## 5. Keras

Keras 是一个高层神经网络 API，Keras 由纯 Python 编写而成并且集成 Tensorflow、Theano 以及 CNTK 后端。Keras 可以满足快速实验的需求，能够把

想法迅速转换为结果，他提供一致而简洁的 API， 能够极大减少一般应用下用户的工作量。

Keras 可以理解为一个层的序列或数据的运算图，完全可配置的模块可以用最少的代价自由组合在一起。具体而言，网络层、损失函数、优化器、初始化策略、激活函数、正则化方法都是独立的模块，可以使用它们很简单的构建自己的深度学习模型，并且都是由 python 代码描述，使其更紧凑和更易 debug，并提供了扩展的便利性

## 6. 模型融合

模型融合顾名思义就是对于同一个问题，采用不同的方式或者不同的数据样本训练多个不同的模型，然后采用某种方式将他们的结果融合到一起，如果方法得当的化，将能够提升问题的解决效果，当用于组合的每个模型显著不同，没有很强的线性相关性的话，融合的效果是比较理想的，我了解到的常见的模型融合方法大致有：

1. Voting，对于多个模型对于同一待预测对象的结果，那么就采取投票制的方法，投票多者确定为最终的分类。
2. Averaging，简单直接的思路是对于多个模型预测结果取平均。稍稍改进的方法是进行加权平均。权值可以用排序的方法确定，举个例子，比如 A、B 三种基本模型，模型效果进行排名，假设排名分别是 1，2 那么给这三个模型赋予的权值分别是 2/3、1/3。
3. Bagging， Bagging 是机器学习定义的集成学习方法中的一种，就是对训练样本采用有放回的方式进行抽样，每次抽取不同的随机样本，每次训练出一个弱分类器模型（相对于最后的融合模型效果而言），将这个过程重复多次，最后再将这些弱分类器的结果进行融合。
4. Boosting，Boosting 与 Bagging 不同，Boosting 先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多的关注，然后基于调整后的样本分布来训练下一个基学习器；如此重复进行，直至基学习器数目达到事先指定的值，最终将这若干个基学习器进行加权结合。
5. 其余的模型融合方法还有 Stacking 和 Blending 等等，碍于篇幅，不在详

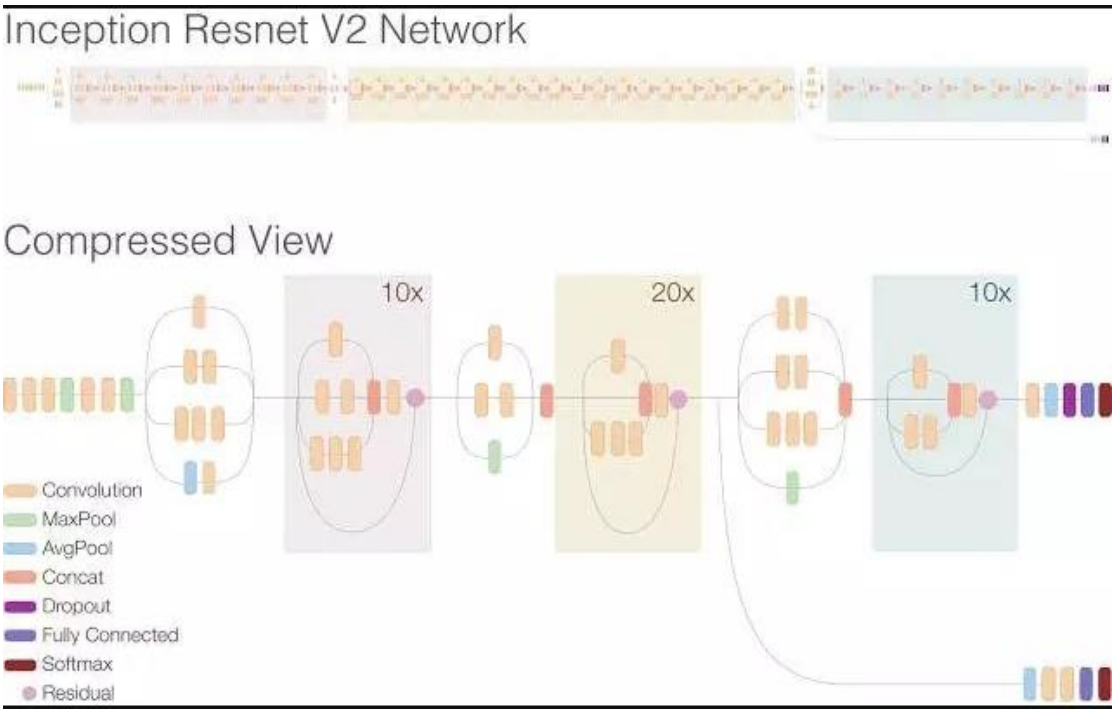
细的论述。

## 7. 数据增强

数据增强指的是对样本量比较小的训练数据通过某种变换操作，从而生成新数据的过程，特别是在图像的深度学习中，为了丰富图像训练集，更好的提取图像特征，泛化模型（防止模型过拟合），一般都会对原数据图像进行数据增强，对于图像的数据增强，常用的方式，就是旋转图像，剪切图像，改变图像色差，扭曲图像特征，改变图像尺寸大小，增强图像噪音（一般使用高斯噪音，盐椒噪音）等。在 tensorflow 与 keras 中，均提供了可方便进行的图像数据增强 API。

## 8. InceptionResNetV2 模型

我在后面的实际操作中，使用了 InceptionResNetV2 模型，它在 ILSVRC 图像分类基准测试中实现了当下最好的成绩。Inception-ResNet-v2 是早期 Inception V3 模型变化而来，同时借鉴了不少微软的残差网络（ResNet）论文中的观点，如下为具体的结构模型图：




InceptionResNetV2 的深度为 572 层，有 55,873,736 个参数，Top1 准确率为 0.804，Top5 准确率为 0.953，是一种相当复杂的网络，同时准确率也非常的理想。作为本人对于此网络的理解还非常的粗浅，还在努力的学习过程中。

9. 几个有代表性的卷积网络：

1）第一个真正意义上的卷积神经网络由 LeCun 在 1989 年提出，它被用于手写字符的识别，提出了权重共享（weight sharing）和特征图像（feature map）的概念，这些概念被沿用至今，就是卷积层的原型

2）深度卷积网络的大发展起步于 2012 年的 AlexNet 网络，它是深度卷积神经网络的鼻祖。这个网络相比之前的卷积网络最显著的特点是层次加深，参数规模变大，除此之外，还有两个主要的创新点：激活函数 ReLU 和 dropout 机制。

3) GoogLeNet 由 Google 在 2014 年提出，其主要创新是 Inception 机制代替了单纯的卷积+激活的传统操作，即对图像进行多尺度处理。其做法是在网络中将多个不同尺度的卷积核，池化层进行整合，形成一个 Inception 模块，共同接受来自前一层的输入图像，并行的进行处理。除此之外，它采用例如全局平均池化等各种手段，大量减少了网络中的参数数量，是优秀且非常实用的模型。

4) VGG 网络由著名的牛津大学视觉组 2014 年提出，VGG 网络的拓展性很强，迁移到其他图片数据上的泛化性非常好。结构非常简洁，整个网络都使用了同样大小的卷积核尺寸（3x3）和池化尺寸（2x2）。到目前为止，依然经常被用来提取图像特征，被广泛应用于视觉领域的各类任务。

5) 残差网络(Residual Network)由微软提出，它用跨层连接（Shortcut Connections）拟合残差项（Residual Representations）的手段来解决深层网络难以训练的问题，将网络的层数推广到了前所未有的规模，比较好的解决了因为网络层次的增加带来的网络退化问题，作者在 ImageNet 数据集上使用了一个 152 层的残差网络，深度是 VGG 网络的 8 倍但复杂度却更低，这个结果赢得了 ILSVRC2015 分类任务的第一名。目前比较有代表性的有 ResNet152 等

6) GoogleNet-Inception-Like 网络系列：随着时间的发展出现了不少基于 Inception 的改进系列，例如 Inception-V2 加入了 BN 层，并且进一步降低参数数量；Inception-V3 则在 V2 的基础上通过卷积核分解使得网络深度进一步增加，增加了网络的非线性，还加入了 RMSProp 优化器和 Label

Smoothing 的正则化方法；Inception-v4 相较于 v3 版本增加了 Inception 模块的数量，整个网络变得更深了；Xception 则是 Google 针对 Inception v3 的另一种改进，主要是采用 Depthwise Separable Convolution 来替换原来 Inception v3 中的卷积操作，在基本不增加网络复杂度的前提下提高了模型的效果；Inception-ResNet v1/v2 则是基于 Inception-v3 和 Inception-v4 将残差网络的思想进行融合，分别得到了 Inception-ResNet-v1 和 Inception-ResNet-v2 两个模型。本人在此次项目中使用的就是 Inception-ResNet-v2 模型。

7) DenseNet 是一种具有密集连接的卷积神经网络。在该网络中，任何两层之间都有直接的连接，也就是说，网络每一层的输入都是前面所有层输出的并集，而该层所学习的特征图也会被直接传给其后面所有层作为输入。DenseNet 的一个优点是网络更窄，参数更少，独特的连接方式使得特征和梯度的传递更加有效，网络也就更加容易训练

从列举各种有代表性的深度卷积网络可以看出，在 AlexNet 之后，很明显的可以看到卷积神经网络的发展分为两类，一类是网络结构上的改进调整，另一类是网络深度的增加，在两个方向的发展中，又不断提出各种加速训练和提升网络性能的方法，并且有的还将各种方向的模型融合，集各种之所长，卷积网络不断发展，适应更多的应用场景，取得了更好的应用效果。

## 六、 基准模型

基准模型我引入了 Xception，在 keras 的介绍中，Xception 模型，权重由 ImageNet 训练而来，深度为 126 层，有 22,910,480 个参数，Top1 准确率为 0.790，Top5 准确率为 0.945。

Xception 作为我的基准模型，和最后使用的模型一样，我使用了 fine-tuning 方法（后文会有详细介绍），在 fine-tuning 方法中，如果将迁移模型全部放开，

参与训练，最后的得分为：

| Submission and Description | Private Score | Public Score | Use for Final Score |
| --- | --- | --- | --- |
| 2018-07-12-17-00-17_all.csv<br>a few seconds to go by liu pan<br>111 | 0.33532 | 0.34276 | ☐ |

# 七、 数据预处理

在开始数据预处理之前，我需要说明的是，我运用了迁移学习，在众多的卷积网络模型中，经过我的实验，我发现 InceptionResNetV2 模型对于我们要解决的问题效果最好，所以，我选用的迁移模型为 InceptionResNetV2.

并且在解决问题的过程中，我使用了类似于 bagging 的集成学习方法，即我最终的结果使用了多个 InceptionResNetV2 模型，融合他们的不同结果作为最终的提交物（在后面会详细论述这个过程）。

所以，我需要在每一轮的模型训练中，随机抽取不同的司机作为训练集训练多个模型，所以，对于数据的预处理来说，我主要做了以下两个事情：

1. N 轮训练数据集的整合、训练、恢复、再次挑选

由于我使用 keras 作为实现框架，而他可以针对训练集自动的生成带标签的数据，但是这些带标签的数据要放在不同的文件夹中，所以，在这个 N 轮的训练中，每一轮数据都是 随机抽取一部分作为验证集-进行训练-恢复数据为初始状态-进行下一轮抽取-进行下一轮训-重复之前过程，直到 N 轮的训练完毕，主要使用的代码如下：

- **建立基本的训练、验证用文件夹，以及存放模型，H5 文件的文件夹**

```
def checkFiles():
    for rootPath in rootPathList:
        if not os.path.exists(rootPath + '/valid'):
            os.mkdir(rootPath + '/valid')
        for i in range(10):
            if not os.path.exists(rootPath + '/valid' + '/c%d' % i):
                os.mkdir(rootPath + '/valid' + '/c%d' % i)
            if not os.path.exists(rootPath + '/drop' + '/c%d' % i):
                os.mkdir(rootPath + '/drop' + '/c%d' % i)
        if not os.path.exists(rootPath+'/saved_models'):
            os.mkdir(rootPath+'/saved_models')
        if not os.path.exists(rootPath+'/saved_models/last'):
            os.mkdir(rootPath+'/saved_models/last')
        if not os.path.exists(rootPath+'/saved_models/best'):
            os.mkdir(rootPath+'/saved_models/best')
        if not os.path.exists(rootPath+'/saved_h5'):
            os.mkdir(rootPath+'/saved_h5')
```

- **下面这段代码就是每次训练时随机挑选出司机，作为新的一轮的训练集和验证集**

```
def pick_train_data(num,rootPathList,CHECK_DRIVERS_NUMBER):
    print('开始挑选本次训练的数据集')
    for i,rootPath in enumerate(rootPathList):
        check_drivers = drivers_to_files.drop_duplicates(['subject'])['subject'].sample(n=CHECK_DRIVERS_NUMBER, random_state=(num+217))
        print('对于第 %d次训练,抽取的验证集司机编号为：'% num)
        print(check_drivers)
        for driver in check_drivers:
            check_drivers_dataFrame = drivers_to_files.loc[drivers_to_files['subject'] == driver]
            for indexs in check_drivers_dataFrame.index:
                filePath = check_drivers_dataFrame.loc[indexs]
                sourceFile = (rootPath + '/train' + '/' + filePath['classname'] + '/' + filePath['img'])
                targetFile = (rootPath + '/valid' + '/' + filePath['classname'] + '/' + filePath['img'])
                if (not os.path.exists(targetFile)) and os.path.exists(sourceFile):
                    os.rename(sourceFile, targetFile)
    print('数据集挑选完毕，可以开始训练过程')
```

- **下面这段代码则为将数据集环境恢复为初始状态，然后开始一下轮司机挑选、训练**

```
def recover_data(rootPathList):
    print('开始重新恢复初始数据集。。。。。。')
    for i,rootPath in enumerate(rootPathList):
        for  filePath in (os.listdir(rootPath + '/valid')):
            files = os.listdir(rootPath + '/valid/'+filePath)
            i = filePath[-1]
            for file in files:
                targetFile = (rootPath + '/train' + '/c%s/' + file) % (i )
                sourceFile = (rootPath + '/valid' + '/c%s/' + file) % (i )
                if (not os.path.exists(targetFile)) and os.path.exists(sourceFile):
                    os.rename(sourceFile, targetFile)
    print('数据已经恢复完毕，可以开始重新挑选数据集')
```

上述三段代码，就是在为融合模型过程中的多个模型挑选出不同的司机作为训练集和验证集，其实，通过多轮测试我发现，这个竞赛只有 20000 多张图片，而需要预测的则有 70000 多张图片，如果对不进行多次挑选司机训练集和测试集的话，单个模型本身很容易产生出严重的过拟合和泛化性能不好现象，最后的预测结果还带有一定的随机性。

我还发现，训练集和测试集的场景和需要提取的特征又是基本一致的，所以，在多轮训练中，如果能够让验证集的司机尽可能的多样的话，去选择每一次挑选之后的对于验证集正确率最高的模型来进行融合的话，会减轻不少过拟合和泛化性能不足的现象。

2. 对于数据本身的处理

我主要使用了 keras 自带的图片生成器 ImageDataGenerator，它主要用来将训练集和测试集的图片读取为图片向量数据，并且还可以自动生成这些向量的标签信息，如下图：

```
datagen = ImageDataGenerator(
    preprocessing_function=inception_resnet_v2.preprocess_input)
```

在创建 ImageDataGenerator 时，还可以定义 keras 自带的图片预处理工具，

如上图，调用了 InceptionResNetV2 的数据预处理工具，将图片缩放为 299*299、归一化数据为-1 到 1 之间（InceptionResNetV2 模型的本身要求）。

同时为了节约内存，我主要使用了图片生成器的 flow_from_directory 方法，它提供的是一个迭代生成器，用来批量的返回训练时所用的图片向量信息

```
train_generator = datagen.flow_from_directory(
    rootFilePath+'/train',  # this is the target directory
    target_size=imageSize,  # all images will be resized to imageSize
    batch_size=TRAIN_GENERATOR_BATCH_SIZE,
    class_mode='categorical')  # since we use binary_crossentropy loss, we need binary labels
```

上图就展示了一个图片生成器的创建过程，它可以定义生成图片的大小，批处理的大小，同时还能打乱数据，避免一次性送入大量相同的训练样本，以免影响效果。最终，图片生成器返回的数据为一个 4 维的向量，分别为：数据批量大小、宽度、高度和通道数，这些数据，都被送入模型中进行训练。

其实，图片生成器还可以实时的进行数据增强，对于图片可以进行缩放、拉升、裁切等等很多数据变化手段，来解决训练数据的缺失，增加泛化能力，但是在本项目中，经过我的实践，漫无目的，没有逻辑的增加数据增强参数，对最后的结果没有起到效果，所以，我暂时没有使用 keras 自带的图片生成器的数据增强方法。以后有机会可以仔细研究。

# 八、 执行过程

执行的主要工作及为在迁移模型的基础上搭建卷积网络模型，主要的步骤如下：

## 1. 多个模型挑选前的预处理

因为我已经决定使用迁移学习来完成项目，有很多的迁移模型我需要进行测试，每个模型的图片格式和预处理方式不一样，所以，我先利用下面这个一个函数，来根据我选择的模型，返回不同的图片生成器以及图片尺寸，方便之后的调用，代码如下：

```
def ImageDataGenerator_select(funtion_name):
    datagen = None
    imgesize = None
    if funtion_name == 'DenseNet201':
        imgesize = (224,224)
        datagen = ImageDataGenerator(
            preprocessing_function=densenet.preprocess_input)
    if funtion_name == 'InceptionResNetV2':
        imgesize = (299,299)
        datagen = ImageDataGenerator(
            preprocessing_function=inception_resnet_v2.preprocess_input)
    if funtion_name == 'InceptionV3':
        imgesize = (299,299)
        datagen = ImageDataGenerator(
            preprocessing_function=inception_v3.preprocess_input)
    if funtion_name == 'Xception':
        imgesize = (299,299)
        datagen = ImageDataGenerator(
            preprocessing_function=xception.preprocess_input)
    if funtion_name == 'ResNet50':
        imgesize = (224,224)
        datagen = ImageDataGenerator(
            preprocessing_function=resnet50.preprocess_input)
    if funtion_name == 'VGG19':
        imgesize = (224,224)
        datagen = ImageDataGenerator(
            preprocessing_function=vgg19.preprocess_input)
    if funtion_name == 'VGG16':
        imgesize = (224,224)
        datagen = ImageDataGenerator(
            preprocessing_function=vgg16.preprocess_input)
    if funtion_name == 'NASNetLarge':
        imgesize = (331,331)
        datagen = ImageDataGenerator(
            preprocessing_function=nasnet.preprocess_input)
    if funtion_name == 'NASNetMobile':
        imgesize = (224,224)
        datagen = ImageDataGenerator(
            preprocessing_function=nasnet.preprocess_input)
    return datagen,imgesize
```

从上图可以看到，在 keras 支持的所有迁移模型，我都罗列了进来，在后续的实验进行过程中，我测试了大部分的迁移网络。

## 2. 搭建迁移网络

```
print("*********************************************************")
print('建立模型！')
# 建立模型
base_model = eval(funtion_name)(weights='imagenet', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(10, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
  paralleled_model = multi_gpu_model(model, gpus=2)

# 先训练全连接层，不用完全随机化的全连接层去连接迁移的模型
for layer in base_model.layers:
    layer.trainable = False

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# 建立图片生成器，利用keras自带的，可以节约机器资源，提高效率,要根据调用的模型，采用不同的书里函数
```

上图代码中的“function_name”确定我需要调用的迁移模型种类，导入的迁移模型去掉了自身的顶层全连接输出层，然后整个迁移模型的输出进入一个全局的平均池化层，经过全局的平均池化之后，然后连接一个 1024 个节点的全连接层，并且使用了 relu 激活函数，最后，按照项目要求，分为 10 类，最后的输出为一个 10 个节点的全连接层，并且使用了 softmax 激活函数，换算成分类预测的概率值，这样，就组成了我的神经网络模型的全结构。

### 3. 整个网络的可视化

整个网络的模型结构图以我使用如下代码，即可实现结构的可视化，

```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```

需要说明的是，我使用的迁移模型 InceptionResNetV2 非常的复杂，用此方式生成的网络图非常的巨大，根本无法在 Word 文档里面进行展示，所以，我将生成的图片也一并放入了文件夹中，可以点击后面的链接进行观看。model.png（此图片非常巨大，点击开了也需要放大很多倍数才能正常观看）

### 4. 准备训练和测试数据

在模型搭建完毕之后，需要开始准备好训练数据和验证数据的输入，前文已

经说明，在本项目中，我使用了 keras 自带的图片数据迭代器来帮我在网络训练时批量的输入数据，使用它自带的 API，我觉得有如下好处：

- 直接帮我按照指定的图片分辨率将图片文件读取成了数字向量，省去了我自己利用 CV2 等等库来自己读取的大量工作。
- 可以方便的设定各种参数，可以让我很方便的调整网络的输入，并且还可以直接生成样本的标签，甚至还自带数据增强功能（本项目中并没有使用）
- 最重要的是，提供的数据迭代器大大的节省了内存，不用一次性的将图片数据全部读入内存，而是通过迭代的方式，分批量的将我的训练数据送入网络，大大节省了我的工作量。

具体实现如下图：

```
datagen = None
print("************************************************************")
datagen,imageSize = ImageDataGenerator_select(funtion_name)
print('设置' + funtion_name + '模型图片处理方式，图片尺寸为:('+str(imageSize[0])+', '+str(imageSize[1])+')')

#设置图片生成器的路径已经其他相关信息
print("************************************************************")
print('读取训练与验证图片！')

train_generator = datagen.flow_from_directory(
    rootFilePath+'/train',  # this is the target directory
    target_size=imageSize,  # all images will be resized to 150x150
    batch_size=TRAIN_GENERATOR_BATCH_SIZE,
    class_mode='categorical')  # since we use binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation data
validation_generator = datagen.flow_from_directory(
    rootFilePath+'/valid',
    target_size=imageSize,
    batch_size=VALID_GENERATOR_BATCH_SIZE,
    class_mode='categorical')
```

在上图中：

1) 我先利用自己所定义的函数，获取了 InceptionResNetV2 的图片读取对象和相关的图片尺寸
2) 定义了两个图片读取迭代器，一个为’train_generator’为生成批量的训练数据，一个‘validtaion_generator’，来批量的生成验证数据，在参数的设置上，我自己中总结的需要注意的要点有：需要将数据的原有顺序打乱，并且分类模式设置为‘categorical’，这样，它会自动的为数据集生成独热编码标签。

## 5. 训练模型详细过程

### 5.1. 迁移学习方式的选择

根据本项目的特点，我觉得迁移方式可以选择如下两种方式：

1） 直接使用在 ImageNet 上训练好的模型权重，直接拿过来使用，让训练数据通过迁移模型提取特征之后送入自己所加的顶层网络结构中，直接得出分类概率结果，因为 ImageNet 上通过良好的网络结构以及充足的算力以及将 1000 余中物体的特征表现得足够良好，在某些前提下可以直接使用在网络当中。

2） 另外一种就是采用 fine-tuning 方式，迁移模型也要部分参与训练，即对迁移模型的某些层的权重开放训练模式，让训练数据流过时也进行权重调整，对迁移模型本身进行微调，我查询到的资料一般是开放迁移模型的高层，让其底层抓取物体一般的共性特征，而让高层模型根据具体问题去抓取更加精细化，更有表征性的特征。即让模型更加适应特定问题，来提供更加理想的结果。

### 5.2. fine-tuning 的具体使用

#### 1） 锁住迁移模型

按照 keras 官方文档的建议，在进行迁移模型 fine tuning 之前，最好先锁住迁移模型，不让其参与训练，而是将自己所加的顶层网络训练的足够充分之后，然后在开放迁移模型进行训练，因为如果自己所加的顶层网络没有训练就开放迁移模型的话，自己所加的顶层网络的随机权重会大幅度的破坏迁移模型本来训练得有一定表征能力的权重，影响整体模型的效果。

```
#     paralleled_model = multi_gpu_model(model, gpus=2)

    # 先训练全连接层，不用完全随机化的全连接层去连接迁移的模型

    for layer in base_model.layers:
        layer.trainable = False

    model.compile(loss='categorical_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])
```

从上图可以看到，我将迁移模型全部锁住了，模型优化器采用‘rmsprop’，损失函数采用多分类交叉熵，而评价指标则使用分类准确度来度量，先训练我自己

加的全连接和全局池化以及最后的输出层，代码如下图：

```
print("*************************************************************")
print('预先训练全连接层！')
model.fit_generator(
    train_generator,
    steps_per_epoch=FC_FIT_PRE_SAMPLES,
    epochs=FC_FIT_EPOCHS,
    validation_data=validation_generator,
    validation_steps=FC_VALID_STEPS)
```

利用 keras 自带的训练 API，设置好训练和验证数据生成器，每轮训练样本数，训练轮数，测试样本数、测试轮数之后，让模型进入训练状态。

**2） 放开迁移模型**

```
# 设置放开训练层数:
if MODEL_OPENED_LAYERS:
    print("*************************************************************")
    print('放开训练层数，本次训练从 %d 层之后开始放开！' % MODEL_OPENED_LAYERS)
else:
    print("*************************************************************")
    print('本次训练迁移模型所有层均已经放开')

for layer in model.layers[:MODEL_OPENED_LAYERS]:
    layer.trainable = False
for layer in model.layers[MODEL_OPENED_LAYERS:]:
    layer.trainable = True

from keras.optimizers import Adam
model.compile(
    optimizer=Adam(
        lr=learning_rate,
    ),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
# 设置权重存储条件
```

上图代码为在将自己所加的全连接层训练得差不多之后，然后进行迁移模型本身的训练，可以根据设置，开放迁移模型需要再训练的指定层数。

在我自己的测试中，我发现，将所有的层数放开，训练的结果是比较理想的，我推测可能本次的项目需要解决的问题是一个固定场景下的问题，背景环境比较单一，且训练集样本差异性又不是很大，且样本量又比较小，容易出现过拟合，在 ImageNet 训练的普遍特征表达没有从问题环境下的提取的实际区别作用大，我觉得充分利用迁移模型结构上的优势可能比迁移模型的初始训练权重更加重

要。所以，在最后的模型训练时，我放开了迁移模型的所有层数，来进行训练。

**3） 进行整个模型的训练**

下面两张图为模型整个参与训练时的代码

```
from keras.optimizers import Adam
model.compile(
    optimizer=Adam(
        lr=learning_rate,
    ),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
# 设置权重存储条件
```

```
# 可以驯练放开之后的迁移模型，保存准确率最高的一个模型
print("*****************************************************************")
print('开始fine tuning!!!')
print('\n')
model.fit_generator(
    train_generator,
    steps_per_epoch=FINETUNING_FIT_PRE_SAMPLES,
    epochs=FINETUNING_FIT_EPOCHS,
    validation_data=validation_generator,
    validation_steps=FINETUNING_VALID_STEPS,
    callbacks=[checkpointer],
    verbose=1)
model.save(fileTime+'_'+'p3'+'_'+str(train_epoch+1)+'_last_model.hdf5')
print(" %s开始的训练在数据集%s上的第%d轮训练结束，最优模型已经存储！！！"%(fileTime,rootFilePath,train_epoch+1))
```

前文已经说过，我放开了迁移模型的所有层，来进行训练，在训练中，我使用了的优化器为 Adam, 学习率 learning rate =0.0001， 损失函数采用多分类交叉熵，而评价指标则使用分类准确度来度量，以上是表现得较好的参数配置。

在训练中，我设置保存了两个模型，一个是在验证数据集中度量最好的模型，一个是最后训练结束时、loss 充分收敛的模型。

**4） 模型参数的调整**

关于模型参数，对于一个模型的调优来说，作为一个初学者，我觉得有数据预处理的方法、网络结构的选择、训练时的学习率、优化器、批量大小、训练停止的轮数、激活函数的选择、以及采用动态学习率、丢弃神经元、使用 BN、增加正则化损失等等。

但是我这个模型使用的是迁移学习模型，并且没有使用数据增强，所以，对于网络结构和使用的层数来说，我觉得不是调整的重点，且迁移网络本身的结构已经足够优秀，且使用了大量的优化和特征提取手段，所以，对于我这个水平的初学者来说，相对于模型整体来说，输入模型的参数的调整是重点一些的内容，但是由于条件所限，主要是硬件限制，我并没有详细记录对于参数不同组合的探

索过程，而是直接在结果中使用了 Adam 优化器, 学习率 learning rate =0. 0001。这个参数组合，在我后面的得分来看，表现还不错。所以，请评审老师见谅。

5） 利用训练好的模型在测试数据集生成结果并存储

```
import h5py
def model_predict(
        mode,
        funtion_name,
        modelFilePath
):
        #确定模型路径
        if mode == 'last':
            rootFilePath = modelFilePath
        else:
            rootFilePath = 'data/saved_models/'+modelFilePath
        if not os.path.exists(rootFilePath):
            print('模型文件不存在，请检查！！！')
            return 0
        #加载模型文件
        print("*************************************************************")
        print('加载模型文件.....模型文件名为：' + rootFilePath)

        model = load_model(rootFilePath)
        print('加载模型文件成功！')
        datagen = None
        print("*************************************************************")
        datagen,imageSize = ImageDataGenerator_select(funtion_name)
        print('设置' + funtion_name + '模型图片处理方式，图片尺寸为：('+str(imageSize[0])+','+str(imageSize[1])+')')

        # 设置图片生成器的路径已经其他相关信息
        print("*************************************************************")
        print('读取训练与验证图片！')
        rootFilePath1 = 'data'
        test_generator = datagen.flow_from_directory(
                rootFilePath1+'/test1',
                target_size = imageSize,
                shuffle=False,
                batch_size=1,
                class_mode=None)

        if mode == 'last':
            targetFilePath = 'predict/last_h5_'+modelFilePath
        else:
            targetFilePath = 'predict/best_h5_'+modelFilePath
        print("*************************************************************")
        print('开始存储模型生成向量')
        # 建立训练、测试和验证变量
#         train = model.predict_generator(train_generator, train_generator.samples)
#         valid = model.predict_generator(validation_generator,validation_generator.samples)
        test = model.predict_generator(test_generator,test_generator.samples)
        #存储文件
        with h5py.File(targetFilePath) as h:
#                 h.create_dataset("train", data=train)
#                 h.create_dataset('valid', data=valid)
            h.create_dataset("test", data=test)
#                 h.create_dataset("train_label", data=train_generator.classes)
#                 h.create_dataset("valid_label", data=validation_generator.classes)

        print('H5文件存储完毕，存储文件名为：'+targetFilePath)
```

上面的代码则是我利用训练好的模型，将测试数据集中所有图片转化为最后的 softmax 概率值，这个值就是最后提交 Kaggle 的值，并且我将每个模型生成的值都保存为静态文件，保存到了一个 H5 文件中，这样也方便我后面的模型融合时方便测试和挑选最优化的方案。

实现也就是利用 keras 提供的 API 加载了训练好的模型文件，构建了一个读取测试数据的图片生成器用来迭代输入测试数据，整个最后结果的生成可以根据参数加载测试集得分最高的模型或者是训练到基本收敛的模型，并且将生成的 H5 文件夹保存下来，以后直接读取这个 H5 结果即可。

**6） 不同迁移模型得分的比较**

此项目 Kaggle 官方给出的评价方法为 multi-class logarithmic loss（多分类对数损失函数），具体计算公式如下：

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

在具体的模型计算中，我需要先挑选出比较一个表现良好的模型，用于后续的 bagging 集成学习，所以，我先测试了在尽可能相同条件下单模型训练时的表现，使用了全部训练数据，放开所有层进行训练，他们上传到 Kaggle 上的得分如下：

| 模型名称 | Private Score | Public Score |
|---|---|---|
| DenseNet201 | 0.9168 | 1.04676 |
| InceptionV3 | 0.50108 | 0.55431 |
| InceptionResNetV2 | 0.33233 | 0.32764 |
| ResNet50 | 0.69064 | 0.56677 |
| Xception | 0.51118 | 0.52975 |

从图上看见，几个单模型的结果都不是很好，可能模型的参数配置和调优没有做好，但是可以提供一个横向的对比，在其中，InceptionResNetV2 模型效果等分最好，所以，在综合考量之后，我觉得选用这个模型作为我的主要模型，来进行后续的训练。

**7） 挑选不同训练集，进行多轮的迭代训练**

前文已经说过，在模型融合的方面，我采取了类似于集成学习中的 bagging 的算法模式，按照司机，随机选取训练集和验证集用于网络训练，每轮进行时采用的是放回取样，生成多个预测结果，最后在去这些预测结果的平均值，在实际操作中，融合的结果，比起单模型，有了一个非常大的提升。

```
import random
import time
def bagging_net(num,train_driver_num,rootPathList):
    fileTime= time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())
    model_name_list = ['InceptionResNetV2']
    for i in range(num):
        K.clear_session()
        model_name = random.sample(model_name_list,1)
        print("************************************************")
        recover_data(rootPathList)
        pick_train_data(i,rootPathList,train_driver_num)
        signal_model_own(
            train_epoch=i,
            fileTime = fileTime,
            rootFilePath = rootPathList[0],
            funtion_name = model_name[0],
            train_generator_batch_size = 32,
            valid_generator_batch_size = 32,
            fc_fit_pre_samples = 300,
            fc_fit_epochs = 5,
            fc_valid_steps=300,
            fine_tuning_fit_pre_samples=1700,
            fine_tuning_fit_epochs =20,
            fine_tuning_valid_steps=500,
            learning_rate=0.0001,
            model_opened_layers=None)
        print('训练已经结束，模型已经保存')
```

```
bagging_net(10,7,rootPathList)
```

上图就是我用来进行多轮迭代训练的函数，每次都调用之前定义的函数，恢复数据集、挑选新的实际组成训练数据集合和验证数据集，在上述的例子中，我每次从 26 位司机中，挑选出 7 位来作为验证集不参与本次的模型训练，最后会一共生成 10 个模型，同时，用这 10 个模型，得出 10 份测试集的预测结果。下面几张图是多轮模型训练的输出图：

```
**********************************************
开始重新恢复初始数据集。。。。。。
数据已经恢复完毕，可以开始重新挑选数据集
开始挑选本次训练的数据集
对于第 0次训练,抽取的验证集司机编号为：
11373    p042
6847     p024
1548     p014
5614     p022
0        p002
15324    p051
725      p012
Name: subject, dtype: object
数据集挑选完毕，可以开始训练过程
本次训练，所采用的迁移模型为 ：InceptionResNetV2!!!
************************************************************
建立模型！
************************************************************
设置InceptionResNetV2模型图片处理方式，图片尺寸为：(299, 299)
************************************************************
读取训练与验证图片！
Found 16030 images belonging to 10 classes.
Found 6394 images belonging to 10 classes.
************************************************************
预先训练全连接层！
Epoch 1/5
300/300 [==============================] - 211s 705ms/step - loss: 1.9070 - acc: 0.3377 - val_loss: 3.4519 - val_acc: 0.1491
Epoch 2/5
300/300 [==============================] - 207s 691ms/step - loss: 1.1668 - acc: 0.6003 - val_loss: 5.7500 - val_acc: 0.1550
```

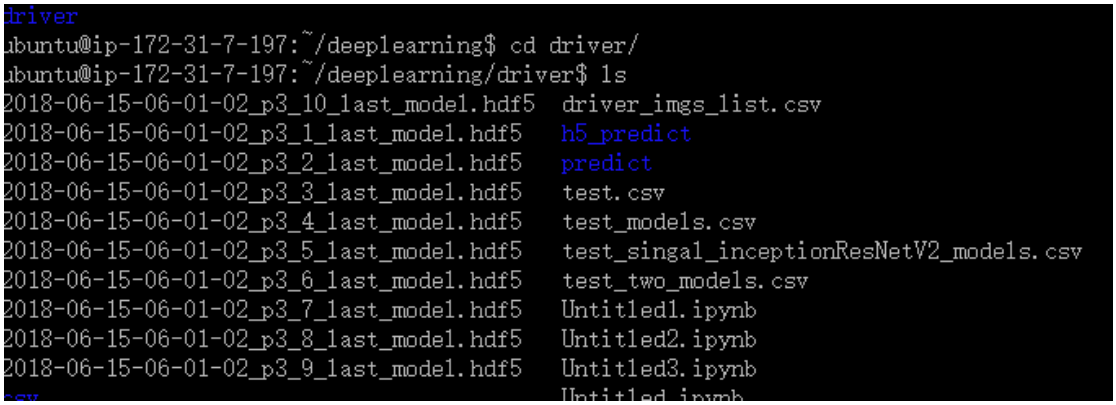
```
************************************************************
本次训练迁移模型所有层均已经放开
************************************************************
开始fine tuning!!!


Epoch 1/20
1699/1700 [============================>.] - ETA: 0s - loss: 0.0365 - acc: 0.9896
Epoch 00001: val_loss improved from inf to 0.57176, saving model to data/saved_models/2018-06-15-06-01-02_p3_1_weights_best_InceptionRe
sNetV2.hdf5
1700/1700 [==============================] - 1135s 668ms/step - loss: 0.0365 - acc: 0.9896 - val_loss: 0.5718 - val_acc: 0.8713
Epoch 2/20
1699/1700 [============================>.] - ETA: 0s - loss: 0.0104 - acc: 0.9970
Epoch 00002: val_loss did not improve
1700/1700 [==============================] - 976s 574ms/step - loss: 0.0104 - acc: 0.9970 - val_loss: 0.5734 - val_acc: 0.8923
Epoch 3/20
1699/1700 [============================>.] - ETA: 0s - loss: 0.0092 - acc: 0.9979
Epoch 00003: val_loss improved from 0.57176 to 0.34638, saving model to data/saved_models/2018-06-15-06-01-02_p3_1_weights_best_Incepti
onResNetV2.hdf5
1700/1700 [==============================] - 998s 587ms/step - loss: 0.0092 - acc: 0.9979 - val_loss: 0.3464 - val_acc: 0.8997
Epoch 4/20
1699/1700 [============================>.] - ETA: 0s - loss: 0.0074 - acc: 0.9982
```

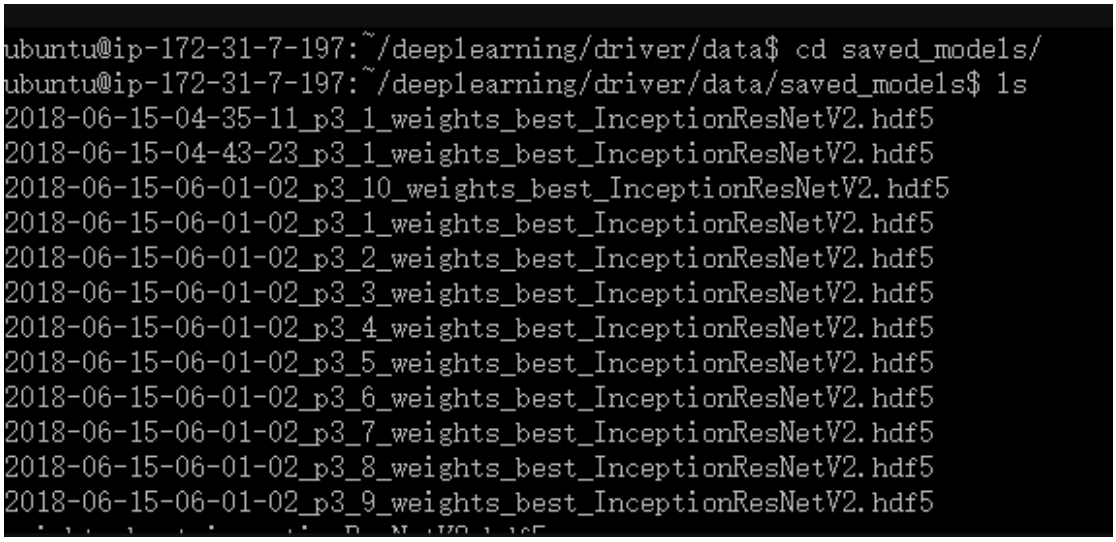
---

# 下图为生成测试集静态 H5 文件

```
************************************************************
加载模型文件.......模型文件名为：data/saved_models/2018-06-15-06-01-02_p3_8_weights_best_InceptionResNetV2.hdf5
加载模型文件成功！
************************************************************
设置InceptionResNetV2模型图片处理方式，图片尺寸为：(299, 299)
************************************************************
读取训练与验证图片！
Found 79726 images belonging to 1 classes.
************************************************************
开始存储模型生成向量
H5文件存储完毕，存储文件名为：predict/best_h5_2018-06-15-06-01-02_p3_8_weights_best_InceptionResNetV2.hdf5
************************************************************
加载模型文件.......模型文件名为：data/saved_models/2018-06-15-06-01-02_p3_7_weights_best_InceptionResNetV2.hdf5
加载模型文件成功！
************************************************************
设置InceptionResNetV2模型图片处理方式，图片尺寸为：(299, 299)
************************************************************
读取训练与验证图片！
Found 79726 images belonging to 1 classes.
************************************************************
开始存储模型生成向量
H5文件存储完毕，存储文件名为：predict/best_h5_2018-06-15-06-01-02_p3_7_weights_best_InceptionResNetV2.hdf5
************************************************************
加载模型文件.......模型文件名为：data/saved_models/2018-06-15-04-35-11_p3_1_weights_best_InceptionResNetV2.hdf5
加载模型文件成功！
************************************************************
设置InceptionResNetV2模型图片处理方式，图片尺寸为：(299, 299)
************************************************************
读取训练与验证图片！
Found 79726 images belonging to 1 classes.
************************************************************
开始存储模型生成向量
```

8） 融合多模型预测结果




从上述两张图可以看出，我一共训练了差不多 20 多个模型（最终并没有全部使用），将这些模型的预测结果，我尝试使用了投票法和平均值方法，在最后的结果提交中，我发现比较符合这个竞赛项目的是采用的简单取平均值的方法，能够取得不错的结果，因为这个模型本身就比较大，操作起来不是非常容易，所以，在取平均值取得不错的成绩之后，我没有再去仔细尝试我在前文提到过的更多的模型融合方法。

```
fileNamePathList = [
                    'last_h5_2018-06-15-06-20-00_p2_2_last_model.hdf5',
                    'last_h5_2018-06-15-06-20-00_p2_1_last_model.hdf5',
                    'InceptionResNetV2_2018-06-13-06-25-50.h5',
                    'InceptionResNetV2_2018-06-13-08-39-33.h5',
                    'last_h5_2018-06-25-09-20-42_p3_4_last_model.hdf5',
                    'last_h5_2018-06-25-09-20-42_p3_2_last_model.hdf5',
                    'best_h5_2018-06-25-09-20-42_p3_3_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-25-09-20-42_p3_1_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-11-46-12_1_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-11-46-12_2_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-11-46-12_3_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-11-46-12_4_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-11-46-12_6_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-06-01-02_p3_1_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-06-01-02_p3_2_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-06-01-02_p3_6_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-06-01-02_p3_7_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-06-01-02_p3_8_weights_best_InceptionResNetV2.hdf5',
                    'best_h5_2018-06-15-06-01-02_p3_10_weights_best_InceptionResNetV2.hdf5',
                    'last_h5_2018-06-15-11-46-12_1_last_model.hdf5'
                    ]
```

```
fileNamePathList = ['h5/'+file for file in fileNamePathList]
X_test = []
for file in fileNamePathList:
    with h5py.File(file, 'r') as h:
        X_test.append(np.array(h['test']))
Y_predict = np.mean([test for test in X_test], axis=0)

# 生成用于提交的CSV文件专用函数
pd_win1 = pd.read_csv('C:/Users\lp\Desktop\csv/last_2018-06-25-15-23-13_win1.csv')
pd_title = (pd_win1['img'])

def creat_csv1(Y_predict,title):
    import time
    fileTime = time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())
    # 写入CSV文件
    df_predict = pd.DataFrame(Y_predict, columns=['c0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9'])
    df_name_predict = title
    df_predict.insert(0, 'img', df_name_predict)
    fileName = 'C:\\Users\lp\Desktop\csv\\' + fileTime + '_win2' + '.csv'
    fileName1 = 'csv/' + fileTime + '_' + 'all.csv'
    df_predict.to_csv(fileName1, index=False, float_format='%.17f')
    print('**************************************************')
    print('CSV已经生成完毕，名称为：%s' % (fileName1))
    return fileName
fileName = creat_csv1(Y_predict, pd_title)
```

上面两张图是我生成最后提交的 CSV 文件的代码，从代码中可以看到，我最后使用了总共 20 个预测模型，作为提交，这 20 个模型，生成的预测结果取简单平均后，提交到 Kaggle 上之后，最后的 private sorce 为 0.16338，是一个我觉得比较满意的名次。

在最终模型的选择上，同一训练集，用最后训练误差充分收敛的模型和在验证集上表现最好的模型，后者的得分明显要更好一些，这充分说明了，这个项目训练起来，很容易产生过拟合。

结合前一段的观点，关于 bagging 方法能够大幅度提高成绩的原因，我是这么分析的：

bagging 方法会在原始训练集的随机子集上构建多个实例，然后把这些估计器的预测结果结合起来形成最终的预测结果。通过在构建模型的过程中引入随机性，来减少基估计器的方差，它泛化能力相对较强，对于降低模型的方差很有作用。当然对于训练集的拟合程度就会差一些，也就是模型的偏差会大一些。

在这个项目中，我觉得这个项目的特点就天然比较适合作为 bagging 的处理对象，首先，通过我的分析发现，这个项目训练样本较少，且场景单一，在训练集上就和容易产生过拟合的现象，所以，我通过多次构建随机测试集和验证集，挑选在在不同验证集上表现最好的模型进行组合，最大程度的让最后的融合模型适应尽可能多的司机，这样实际是提高了它的泛化能力，使其在测试集的表现也会好一些。至于这样导致训练样本减少使得单模型在精确度下降来说，泛化能力的提升，显然，在这个项目来说，更加重要一些，并且，从我的经验来看，使用 InceptionResNetV2 模型，在 2000 多张数据集上迭代一次，就已经在某些测试集上达到了不错的精度了。这二者的平衡点，其实在这个项目上，还可以继续深入的研究。

**9） 处理生成的 CSV 文件，提供最后的得分**

在前文的数据预处理中，我已经提到，测试数据集实际上也是一小段视频的截图，所以，要是两张连续的截图，司机的动作基本一致，分类应该是相同的，他们在模型中的预测值也应该相似，我觉得是不是可以在最后的结果上，进行一些处理，考察一张图片前后的连续视频帧的分类信息，将其加入到单张图片的预测结果中，所以，我的基本步骤是这样的：

A. 利用图像搜索的哈希算法，计算出图像的指纹，我发现，均值哈希算法最适合这个数据集，找出的相似图片最为精确

B. 找出相似图片后，相似图片基本和需要预测的图片基本为同一个司机的连续动作，将这些相似图片的预测结果取其平均值，我在这个项目中，一张图片大概平均了其周围 20 多张图片的预测信息

C. 将处理后的信息重新上传，查看其得分

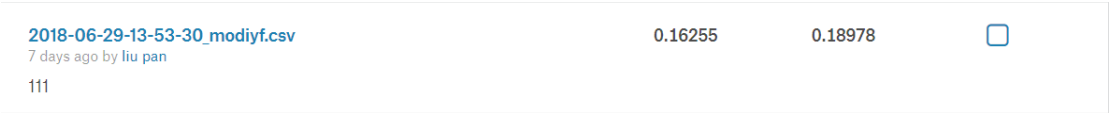
```
columnms=['c0','c1','c2','c3','c4','c5','c6','c7','c8','c9']
csv_path = 'csv/2018-06-29-13-20-09_all.csv'
files = pd.read_csv(csv_path)
modify_files = files.copy()

with open('pkl\\hashList_50.pk', 'rb') as f:
    hashList = pickle.load(f)

for file in tqdm(modify_files['img']):
    temp_list = [temp[0] for temp in  hashList[file] if temp[1] == 1]
    checked_list = []
    if(len(temp_list)!=1):
        old_item_typed = np.array(modify_files[modify_files['img'] == file][columnms]).argmax()
        checked_list    =[item for item in temp_list if old_item_typed == np.array(modify_files[modify_files['img'] == item][columnms]).ar
        if (len(checked_list)!=0):
            s = (modify_files.loc[modify_files['img'].isin(checked_list)][columnms])
            modify_files.loc[modify_files['img'] == file, columnms] = np.array(s.mean())



import time
csv_columns =['img','c0','c1','c2','c3','c4','c5','c6','c7','c8','c9']
fileTime = time.strftime("csv\\%Y-%m-%d-%H-%M-%S", time.localtime())+'_modiyf.csv'
modify_files.to_csv(fileTime,index=False)
```

上面代码就是我利用周围预测当前图片的主要代码片段

上传至 Kaggle 检验其效果：

- **未进行连续图片处理的 kaggle 得分结果：**


- **进行过连续图片处理的 kaggle 得分结果：**




从上图可以看出，经过连续图片处理的结果，有了一个相对较明显的提升。我个人觉得，如果将模型生成的测试集的中间向量拿出来做无监督学习的聚类的话，也可以找到精确相似的图片，并且还属于机器学习的范畴，说不定能够提高成绩，但是因为时间所限，所以，我并没有采用这种方法。

**10）最终结果**

所以，我此项目的最终 private score 为 0.16255，在整个排行榜中，可以排名第 17 位，已经达到了学城的成绩要求。

# 九、 总结与改进

我个人觉得这个项目的特点就是缺少训练集、模型容易产生过拟合，泛化性能不足，导致预测的结果往往还带有一定的随机和运气因素，所以，提高成绩的方式重点就是围绕解决这个问题来进行。

我通过选取适合的模型，按司机多次随机选取训练集的方法，训练多个模型，

最后融合结果，弥补预测模型的泛化能力不足，取得了 private score 在 Kaggle 上排名第 17 名的成绩，满足了学城的项目要求。

后续改进的思路，如果我要对这个项目进行持续的改进的话，我可能会从如下放下入手：

- 针对于单模型来说，应该对模型进行详细的可视化的分析和改进，但是我选取的模型比较复杂，大量精力又浪费在硬件的配置已经环境的搭建上面，其实还可以深入对于单个模型调优后在进行融合，我应该会尝试选择简单、参数稍微少一些的迁移模型，这样比较好进行调整。
- 在模型的融合方式上，我采用的是简单平均预测值的方式，其实我在测试的过程中发现，不同模型对于预测种类的能力是不同的，比如有些模型在 A 类动作的预测值上更准确一些，有些在 B 类动作的预测能力更强，所以，要是有机会的，可以试试加权融合，或者更具不同模型的预测能力来尝试一些别的手段，总之，我觉得这个项目的难点就在于需要解决训练数据不足，泛化能力差的问题。
- 可以围绕训练数据来做文章，我觉得可以尝试的有如下方式：
  - 利用 keras 自带的数据增强 API 来提升，但是我自己尝试下了，发现效果不好，暂时没有找到好的方法。
  - 可以对图片裁剪、拼接、颜色、灰度变化方面的尝试
  - 在这个项目中，我觉得除了司机、方向盘等等之外的背景图像意义不大，所以，可以通过检测司机人体的方式，将司机、方向盘先截取下来，作为训练集让网络去训练，去掉那些无关的景色，后面人物等，我觉得抓取的特征会更加有利于判断。
- 而对于测试数据集，我觉得因为他是也是一小段连续视频的连续帧，所以，要是采用更加准确的无监督的分类算法精确的找到所有连续动作的图片，用一张图片加前后若干张图片的预测结果一起结合，我觉得能够进一步的提高最后的得分，并且我觉得这个是一个在实际使用中也比较合理，比较实用的做法，即在一个小的时间窗口内，用一个司机的连续动作来预测他的行为，显然是更加合理的。
- 我觉得更有挑战的是，可以把 20000 多张训练集看成是有标签的学习数据，

将 70000 多张无标签的测试集看成是没有标签的学习数据，采用半监督学习的方法来训练模型，是不是可以得到更大的提升呢。

参考文献：

[1] Jonathan. Long, Evan. Shelhamer and Trevor. Darrell. Fully Convolutional Networks for semantic Segmentation In ICLR, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[3] Christian.Szegedy,Vincent.Vanhoucke, Sergey.Ioffe, Jonathon.Shlens, Zbigniew.Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567,2015

[4] Christian. Szegedy, Sergey. Ioffe, Vincent.Vanhoucke. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv:1602.07261[cs.CV]

[5] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357 [cs.CV].2017

[6] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger.Densely Connected Convolutional Networks. arXiv:1608.06993 [cs.CV].2018

[7] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.arXiv:1602.07261 [cs.CV]