机器学习工程师纳米学位毕业项目

Kaggle 比赛之走神司机侦测

王闻宇

2017 年 9 月 7 日

目录

1. 定义

1.1. 项目概述

我们经常遇到这样的场景：一盏灯变成绿色，你面前的车不走。 另外，在没有任何意外发生的情况下，前面的车辆突然减速，或者转弯变道。等等这些现象，给道路安全带来了很大的影响。

那么造成这样现象的原因是什么，主要有因为司机疲劳驾驶，或者走神去做其他事情，想象身边的例子，开车时候犯困，开始时候打电话，发短信，喝水，拿后面东西，整理化妆的都有。这对道路安全和行车效率形成了极大的影响。

据中国安全部门介绍，五分之一的车祸是由司机分心引起的。令人遗憾的是，这样一来，每年有 42.5 万人受伤，3000 人因分心驾驶而死亡。

我们希望通过车内摄像机来自动检测驾驶员走神的行为，来改善这一现象，并更好地保证客户的安全。

1.2. 问题描述

我们要做的事情，就是根据车内摄像机的画面自动检测驾驶员走神的行为。如果是安全驾驶则一切正常，如果有走神行为，给予警报提醒。

驾驶员可能存在的走神的行为，大概有如下几种，左右手用手机打字，左右手用手持方式打电话， 调收音机（玩车机），喝饮料，拿后面的东西，整理头发和化妆，和其他乘客说话。
侦测的准确率 accuracy 就是衡量解决这个问题好坏的重要指标

1.3. 输入数据集

输入数据集来自 Kaggle 下载地址如下：
https://www.kaggle.com/c/state-farm-distracted-driver-detection/data
下载下来解压后有 3 个文件
- driver_imgs_list.csv.zip (92.89K)
- imgs.zip (4G) 所有的图片数据，解压后
    - train (训练集数据)
        - c0 ~ c9 分别表示不同状态的训练集

- test (测试集数据，用于提交 Kaggle 比赛的测试集)
- sample_submission.csv.zip (206.25K) Kaggle 比赛需要提交的样本

其中 driver_imgs_list.csv.zip 的是对分类标号和人分类编号的 csv 文件。这个 csv 表格有三列
subject：人的 ID，不同的人，这个值不同
classname：状态，c0 ~ c9
img：图片名称

数据有这些特点
训练集的图片数量是 22424，测试集的图片是 79726，测试集的数量远大于训练集。而且训练集司机完全不是测试集司机。
由于数据有这些特点，那么过拟合可能是要遇到的最大的问题。

不过，对于每个司机而言，其数据是连续的，是从一个视频里面一帧一帧截取出来的，这个特点可能对最后的分析有用。

1.4. 评估指标

这是典型的分类问题，评估指标采用
精度 accuracy 来评估结果好坏。
Logloss 的评估方式，这也是 kaggle 比赛的评估方式

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

对比这两种方案。Accuracy 对于判断正确和错误的比重是一样的，也就是对了就多一个，错了就少一个，最终看正确的百分比，也就是判断正确的占总体测试数据的比例。

而 logloss 的评估方式对判断是不是是有明显的方法，如果正确了，Pij=1 => log(Pij)=0, 而 Pij=0.999 => log(Pij)=-0.001。最后增加的 log 差不多。但如果判断错误，如 Pij=0 => log(Pij) = -无穷。Pih=0.001 => log(Pij)=-6.9 也就是判断错误一个，对等分影响会非常大。

我认为，在 accuray>0.9 的情况下，看 logloss 更有意义

2. 分析

2.1. 数据可视化

下面是 10 种状态下每个状态的示例图片：图片大小 640x480

c0 安全驾驶 	c1 右手手机打字 
c2 右手打电话 	c3 左手手机打字 
c4 左手手机打字 	c5 调收音机（玩车机） 
c6 喝水	c7 拿后面东西

	
c8 整理头发和化妆	c9 和其他乘客说话
	

2.2. 算法与技术

这是一个分类器分为，预测的时候是将图片进行归类 C0~C9
首先，将 train 数据，划分为训练和验证集。由于训练集的司机和测试集的司机不是一个人，所有在划分训练和验证集的时候，要区分司机，把一部分司机的数据做为训练集，把另外部分司机的数据作为验证集。

第二步，建模调参，首先采用迁移学习(transfer learning) 的方式，对 imagenet 上的已经训练好的模型拿过来，只对以已经预测过的数据做全连接层的训练，也就是保留全连接层之外所有层次的 weights。我这里会采用 Keras 自带的迁移学习模型，尝试使用 VGG16，VGG19，ResNet50，InceptionV3，Xception 看看结果。

第三步，根据这些迁移学习模型的结果，尝试改造模型和自己建模，并且参数调优，在验证集上优化精度 accuracy。

第四步，选择最高精度 accuracy 的模型和参数，手动拿出 16 个图片来看看结果是否正确，生成 Kaggle 的 pred 文件并提交文件，看看世界排名的情况，争取做到世界排名的 1/3。

2.3. 基准指标

我用 ImageNet 上已经成熟的模型来做基准模型来和我的计算结果做对比。

我选择选择 ResNet50 的去掉原有全连接层之后，自己训练全连接层来做为基准模型。我基于这个基准模型再做改进。

我的目标是： accuracy > 0.93 并且 logloss < 1.0 并且 世界排名在前 1/3。

3. 实现

3.1. 在训练数据中区分训练集和验证集

由于数据有这个特点：训练集司机完全不是测试集司机，所以在训练数据中区分训练集和验证集的时候，不能简单地随机分离，也要区分司机，不然很容易出现过拟合。【另外，我试过，如果训练集和测试集，如果不区分司机的化，直接简单随机拆分的化，最后看似精度能做到很高，但是提交到 kaggle 后，分数非常低，也就是验证集得出的验证结果会严重失真】

于是我，抽取了司机编号为 p081 和 p075 这两个司机的数据，作为验证集使用，其他司机都作为训练集使用。

这样，训练集的数据有 20787 条，验证集的数据有 1637 条，这样训练集和验证集的比例为大约 12.6：1。

这部分的代码在 splite_valid.py 中。

3.2. 数据预处理

于是我，将所有图片生成用于迁移学习的数据集 h5 文件，h5 文件是 h5py 这个 python 库的保存文件，使用 h5py 这个库可以轻松地写入和读取批量数据结构。

我的数据预处理，主要分两步：

第一步对图片做 resize，其中 ResNet50, VGG16 和 VGG19，我将图片 resize 到 224x224，因为这是这三个模型的默认输入值；另外，InceptionV3 和 Xception，我将图片 resize 到 299x299，因为这是这两个模型的默认输入值。

第二步是使用 Keras 自带的模型（VGG16，VGG19，ResNey50，InceptionV3，

Xception)，使用 ImageNet 的权重，去掉全连接层，对所有的训练集，验证集，测试集数据，做预测。由于去掉了全连接层，得到的结果是向量，其中，ResNet50，InceptionV3，Xception 模型得到的向量长度是 2048；而 VGG16，VGG19 模型得到的向量长度是 512。然后多个图片预测后得到的向量写入到 h5 文件中，便于后面最终的训练和预测。

这部分的代码在 write_bottleneck.py 中。

3.3. 基准模型评估

以下部分的代码在 main.ipynb。
为了最终的优化和对比，我采用一个统一的全连接层，我采用的全连接层如下。

我选择的优化器是 adadelta，因为 adadelta 有个特点，就是自适应调整学习率。

我选择的损失函数是 binary_crossentropy，因为 binary_crossentropy 采用的就是 logloss 对数损失，和我开题的评估方式是一致的，也是 Kaggle 比赛的评估方法。

我采用以下模型来进行训练和获得结果

3.3.1. VGG16 模型

VGG16 的输入模型时，图放缩到大小为 (224, 224, 3)
其去掉全连接层之后的输出是 长度为 512 的向量。

在第 10 次迭代后的结果是：loss: 0.1733 - acc: 0.9351 - val_loss: 0.3505 - val_acc: 0.8914

3.3.2. VGG19 模型

VGG16 的输入模型时，图放缩到大小为 (224, 224, 3)
其去掉全连接层之后的输出是 长度为 512 的向量。

在第 10 次迭代后的结果是：loss: 0.1697 - acc: 0.9357 - val_loss: 0.3293 - val_acc: 0.8917

3.3.3. ResNet50 模型

VGG16 的输入模型时，图放缩到大小为 (224, 224, 3)

其去掉全连接层之后的输出是 长度为 2048 的向量。

在第 10 次迭代后的结果是：0.0732 - acc: 0.9729 - val_loss: 0.2139 - val_acc: 0.9155

3.3.4. InceptionV3 模型

VGG16 的输入模型时，图放缩到大小为 (299, 299, 3)
其去掉全连接层之后的输出是 长度为 2048 的向量。

在第 10 次迭代后的结果是：loss: 0.0945 - acc: 0.9649 - val_loss: 0.2074 - val_acc: 0.9249

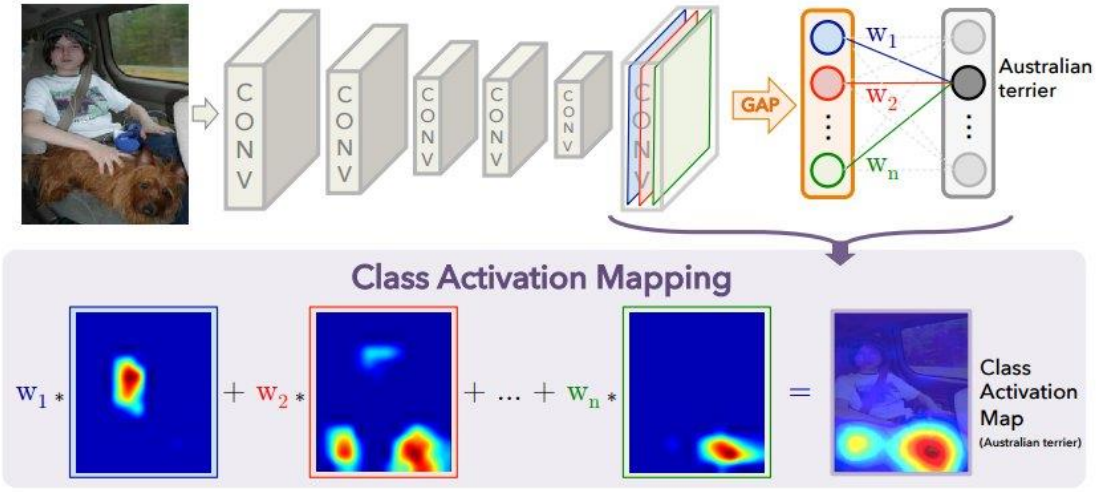
3.3.5. Xception 模型

VGG16 的输入模型时，图放缩到大小为 (299, 299, 3) 。
其去掉全连接层之后的输出是 长度为 2048 的向量。

在第 10 次迭代后的结果是：loss: 0.0792 - acc: 0.9729 - val_loss: 1.7878 - val_acc: 0.8184

3.4. 可视化分析

采用 CAM 可视化的方案分析各个模型的情况，CAM 可视化方案参考 [h http://cnnlocalization.csail.mit.edu/](http://cnnlocalization.csail.mit.edu/)。

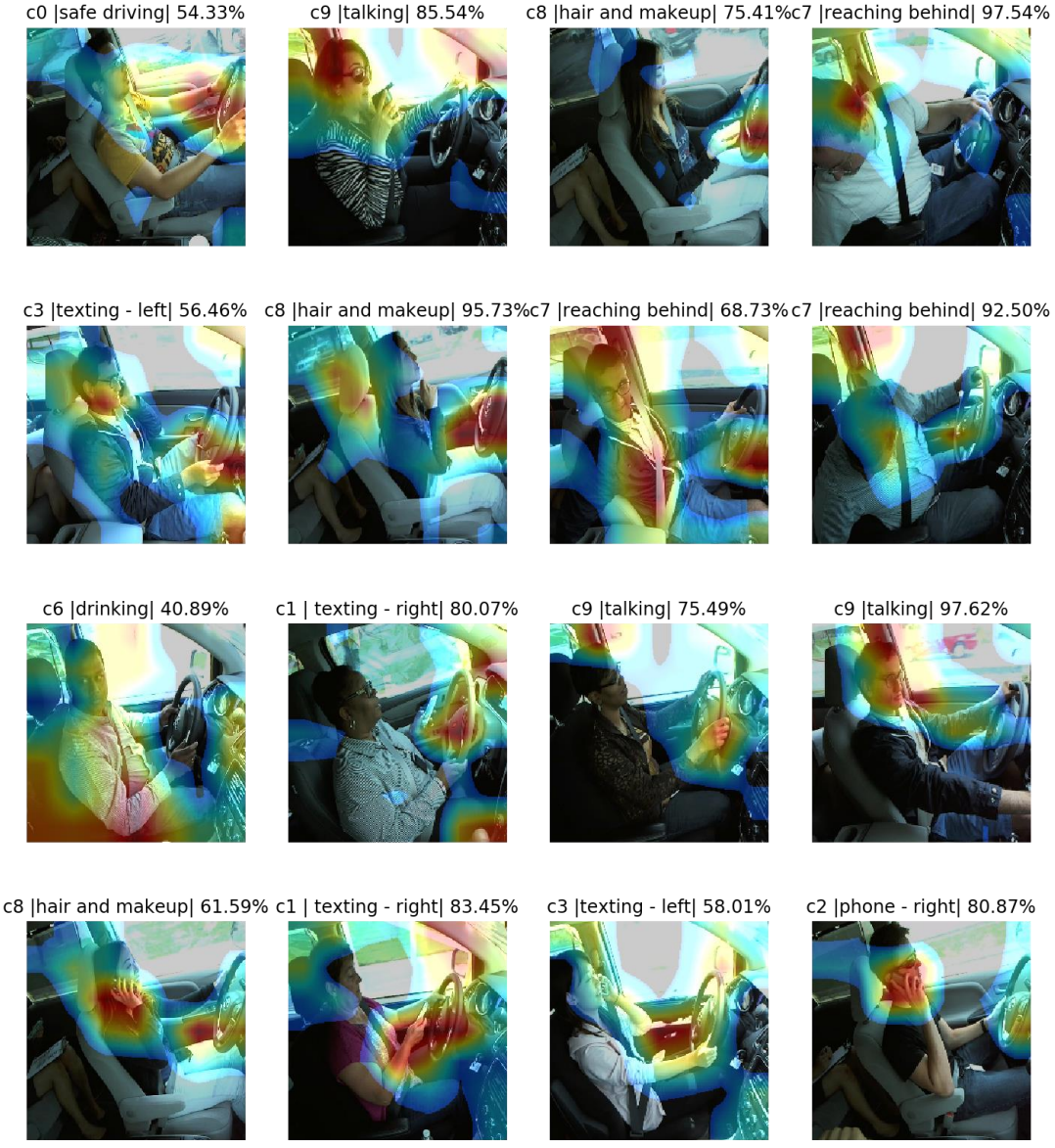


下面我将列出每个迁移学习模型的可视化图。我一次分析了，每种模型的分析图。我在分析每种模型的时候，遵循了以下几点，保留了原有模型的非全连接层的 ImageNet 权重，将全连接层换成了比较简单的非全连接层，并且在训练的时候，只拟合了非全连接层的权重。由于为了提升性能，在分析 CAM 热力图的时候，用的 batch_size=16,

epochs=10。如果 batch_size 如果太大了，分析会非常非常慢，而使用这几个模型主要的目的是为了分析，所以 batch_size 参数设得非常小

3.4.1. VGG16 模型

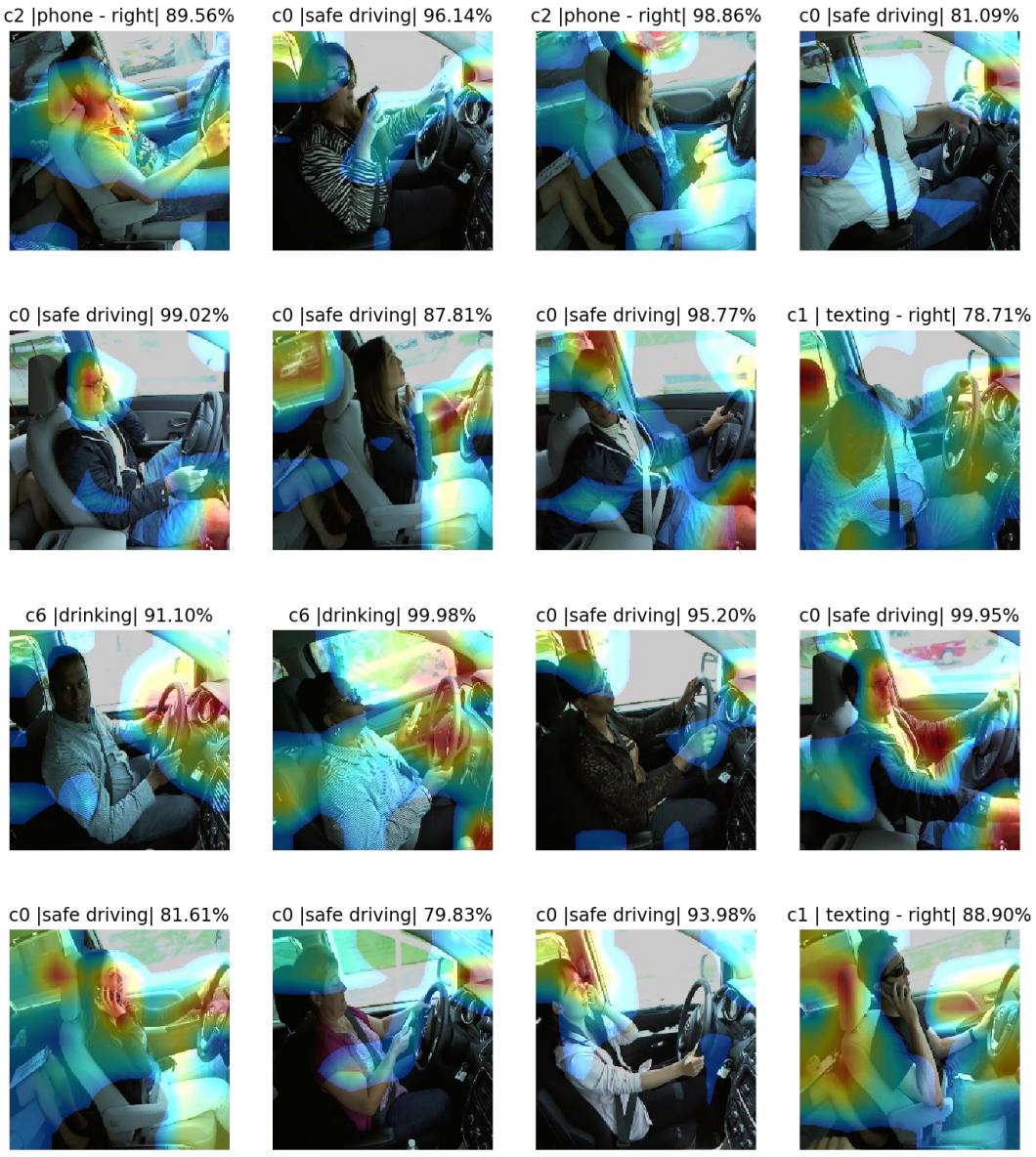
可视化 Notebook 见 keras-vgg16-visual.ipynb



分析： 直观感受，CAM 热力图收敛得并不是非常好，神经网络看了很多不该看的地方，该看的地方没有看。

3.4.2. VGG19 模型

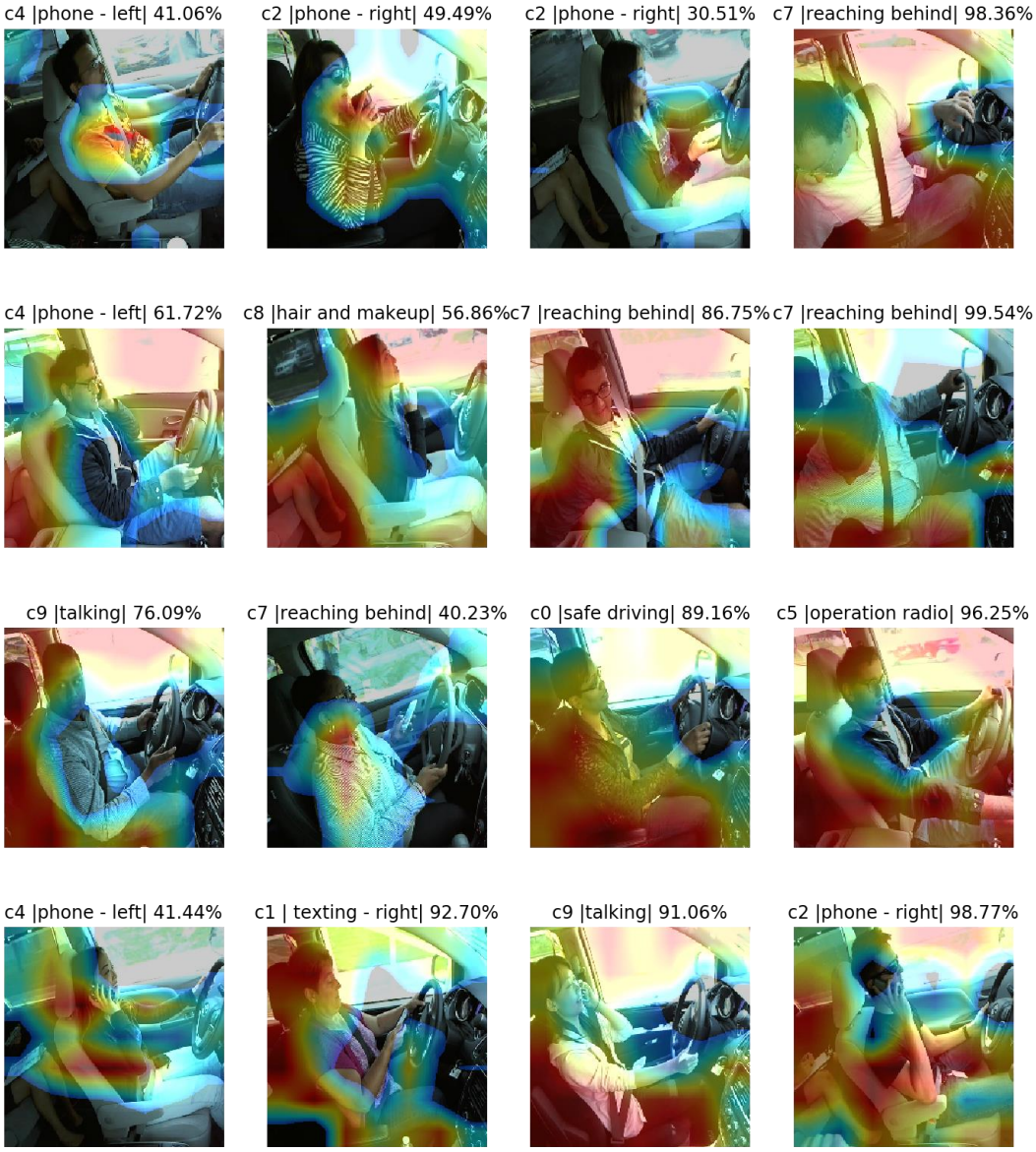
可视化 Notebook 见 keras-vgg19-visual.ipynb



分析： 直观感受，CAM 热力图收敛得并不是非常好，神经网络看了很多不该看的地方，该看的地方没有看。

3.4.3. ResNet50 模型

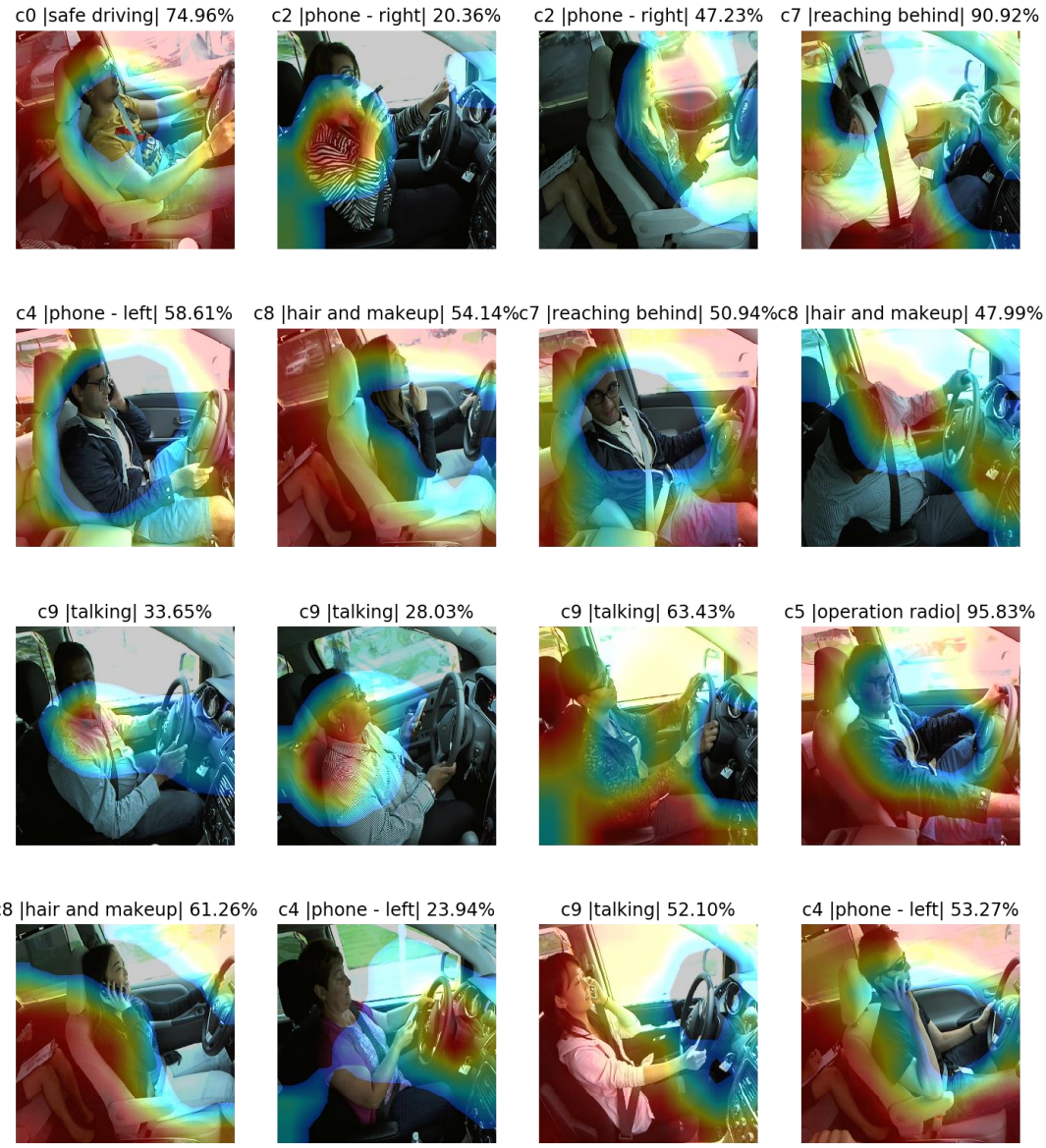
可视化 Notebook 见 keras-resnet50-visual.ipynb



分析：直观感受，CAM 热力图收敛得比 VGG 模型要好些。

3.4.4. InceptionV3 模型

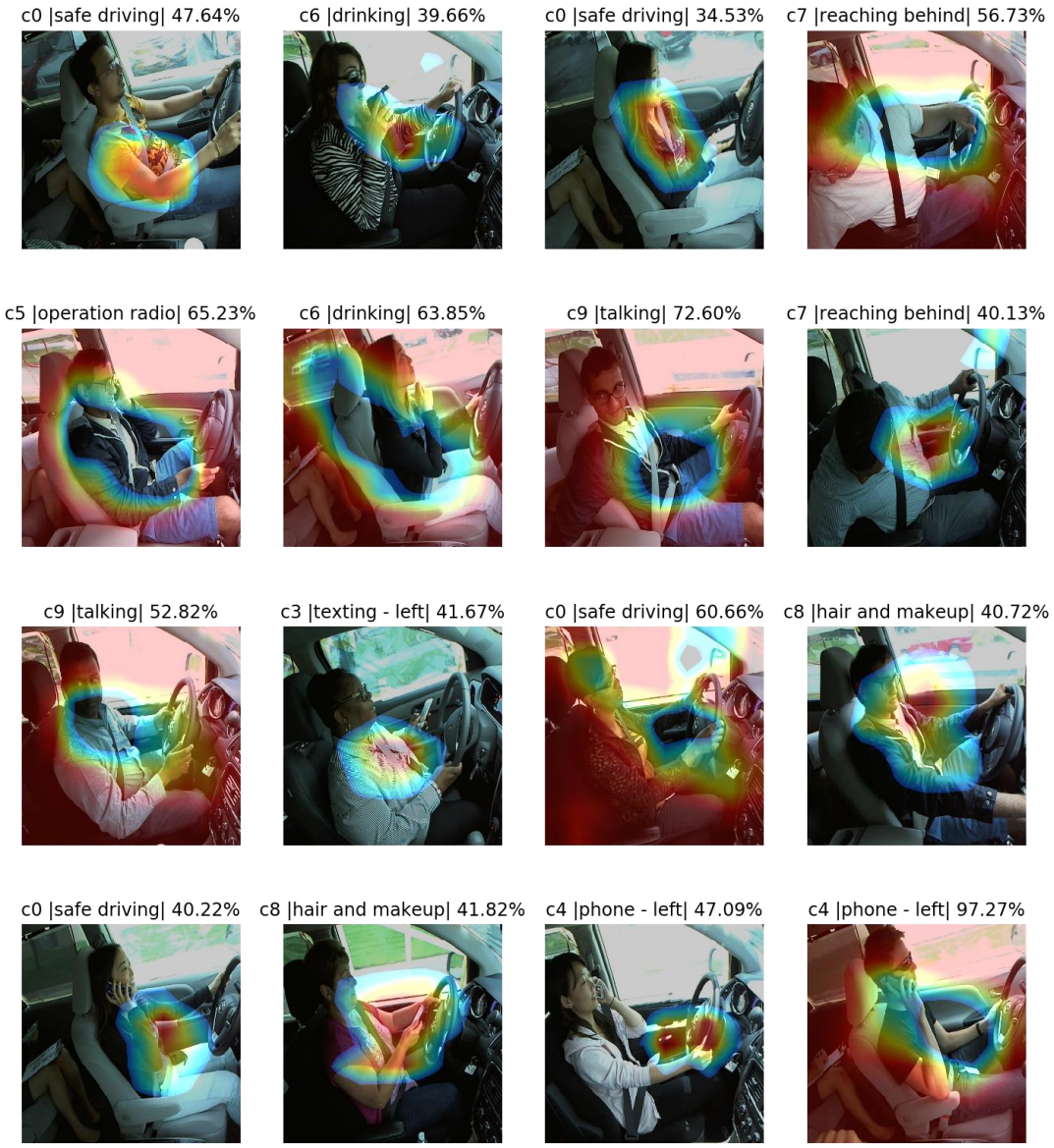
可视化 Notebook 见 keras-inceptionV3-visual.ipynb



分析：直观感受，CAM 热力图收敛得不算好，看了很多不该看的地方，但是对比起来比 VGG 系列要好。

3.4.5. Xception 模型

可视化 Notebook 见 keras-xception-visual.ipynb



分析：CAM 热力图收敛得不算好，也看了很多不该看的地方。

3.5. 模型改进

3.5.1. 之前模型的总结

从计算结果上看，大致如下：

首先对比各个模型的运行结果如下，参数都是

模型	训练 Accuracy	验证 Accuracy	训练 Loss	验证 Loss	Kaggle Logloss Public Score	Kaggle Logloss Private Score
VGG16	0.8351	0.8914	0.1733	0.3505	1.64561	1.92176
VGG19	0.9357	0.8917	0.1697	0.3293	3.56845	3.47565

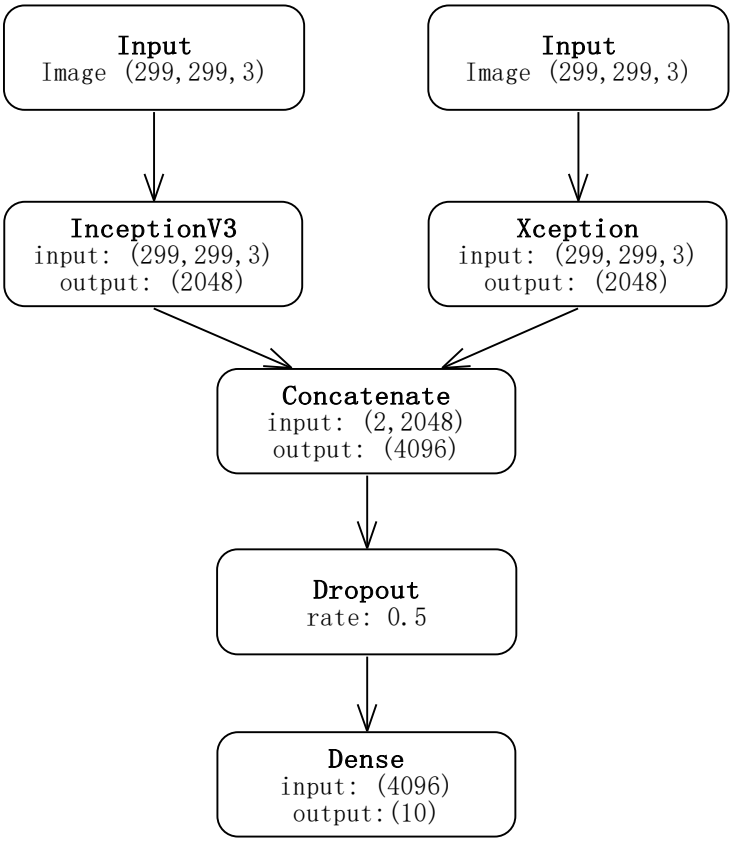
ResNet50	0.9729	0.9224	0.0732	0.2139	1.37847	1.28424
InceptionV3	0.9649	0.9249	0.0945	0.2074	1.32846	1.03619
Xception	0.9729	0.8184	0.0792	1.7878	0.98077	0.98665

根据以上基准模型的数据，已经从 CAM 看神经网络最后关注的位置，可以考虑将 Xception 的数据和 InceptionV3 的数据结合起来，续我采用的最终方案是将 Xception 和 InceptionV3 得到的全连接层之前的向量串起来，再进行全连接层的训练，从而得到最后的结果。

注：为什么选择 Xception，Xception 在我自己的数据集上计算并不好，但是在 Kaggle 排名上最好，所以我把 Xception 作为选择之一。

3.5.2. 新的模型

新的模型把 InceptionV3 和 Xception 混合起来做，从而完成结果

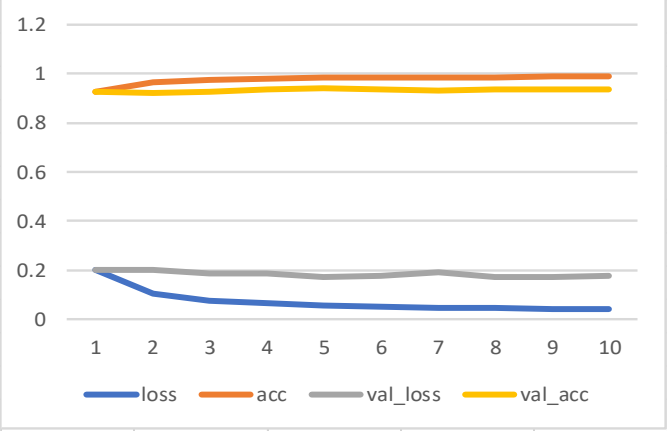


4. 结果

4.1. 模型评估与验证

使用新的模型的运行情况如下表。

Epoch	loss	acc	val_loss	val_acc
1	0.2017	0.9273	0.2018	0.9248
2	0.1031	0.9638	0.2016	0.9225
3	0.0761	0.9751	0.1872	0.9276
4	0.0628	0.9799	0.1836	0.9342
5	0.0557	0.9824	0.1731	0.9388
6	0.0497	0.9844	0.1765	0.9359
7	0.0455	0.9857	0.1886	0.9303
8	0.0433	0.9862	0.1729	0.936
9	0.0407	0.9873	0.1697	0.9382
10	0.0384	0.9879	0.1741	0.9385



模型在训练的过程中，是呈现收敛状态的。

4.2. 结果分析

本地验证结果看：

验证集 Loss：0.1741

验证集 Accuracy：0.9385

将结果提交到 Kaggle，得到的分数是:

Public Score: 0.91478 排名： 473 / 1440

Private Score: 0.78666 排名： 447 / 1440

其结果对比起基准模型

模型	训练 Accuracy	验证 Accuracy	训练 Loss	验证 Loss	Kaggle Logloss Public Score	Kaggle Logloss Private Score
VGG16	0.8351	0.8914	0.1733	0.3505	1.64561	1.92176
VGG19	0.9357	0.8917	0.1697	0.3293	3.56845	3.47565
ResNet50	0.9729	0.9224	0.0732	0.2139	1.37847	1.28424
InceptionV3	0.9649	0.9249	0.0945	0.2074	1.32846	1.03619
Xception	0.9729	0.8184	0.0792	1.7878	0.98077	0.98665
Final	0.9879	0.9385	0.0384	0.1741	0.91478	0.78666

5. 结论

5.1. 总结

以上的结果勉强达到了我之前设定的 Accuracy > 0.93, Loss < 1.0, 世界排名在前 1/3 的目标。

但是由于我自身机器性能问题，和业务学习不多没有足够时间训练数据，没有做得更好。 不过我会继续改进算法，改进思路大概如下。

5.2. 后续改进

关于后续改进有几点想法：

5.2.1. 不锁定 ImageNet 模型的部分 weights

我因为机器性能和工作时间问题，目前使用 Keras 里面的现成模型，使用 Keras 自带下载的 Imagenet weights 文件，只训练了全链接层，由于走神司机侦测要看很多细微的差别，而对 ImageNet 的训练来说，并不一定往本课题中希望观察的部分收敛了，如果可以放开一部分层次的 weights，让这些知名模型学习的化，对最终的结果应该是有帮助的。

5.2.2. 可以参考前后几帧图片来判断

我现有的方式，是把每一帧数据当作完全独立的图片来处理的，但是如果参考前后每帧的画面内容，想办法用起来，对结果应该是有些帮助的。而且如果最终在车上实地部署的化，新数据预测的时候也可以采用关联前后帧的方式来做，这样应该会有更高的精度。

后续我有更多时间，还会继续优化和尝试，把分数做得更高。

参考文献

Very Deep Convolutional Networks for Large-Scale Image Recognition [Karen Simonyan, Andrew Zisserman] (Submitted on 4 Sep 2014 (v1), last revised 10 Apr 2015 (this version, v6)) https://arxiv.org/abs/1409.1556

Deep Residual Learning for Image Recognition [Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun] (Submitted on 10 Dec 2015) https://arxiv.org/abs/1512.03385

Rethinking the Inception Architecture for Computer Vision [Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna] (Submitted on 2 Dec 2015 (v1), last revised 11 Dec 2015 (this version, v3)) https://arxiv.org/abs/1512.00567

Going Deeper with Convolutions [Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich] (Submitted on 17 Sep 2014) https://arxiv.org/abs/1409.4842

ADADELTA: AN ADAPTIVE LEARNING RATE METHOD
[Matthew D. Zeiler1] (Submitted on 22 Dec 2012) https://arxiv.org/abs/1212.5701

手把手教你如何在 Kaggle 猫狗大战冲到 Top2% [杨培文] （发表于 2017-03-18）
https://ypw.io/dogs-vs-cats-2/

使用 Keras 来破解 captcha 验证码 [杨培文] (发表于 2017-03-07)
https://ypw.io/captcha/

Kaggle 求生：亚马逊热带雨林篇 [刘思聪] (发表于 2017 年 7 月 31 日)
https://zhuanlan.zhihu.com/p/28084438?utm_source=itdadao&utm_medium=referral