

# 侦测分神司机

(机器学习纳米学位毕业项目)

方无迪 2017 年 10 月 26 日

## 1 问题的定义

### 1.1 项目概述

本项目名为侦测分神司机，来源于数据分析竞赛网站 Kaggle 在 2016 年 8 月的比赛项目[1]。我们知道，司机的分神行为（比如打电话、发短信）对行车安全和交通效率有很大影响。该比赛的愿景是通过车内摄像机来自动检测司机驾驶行为，来据此更合理地为顾客车险投保，有助于改善所述现状。



插图 1：分神司机样照

比赛发起方为 State Farm，美国最大的车险公司。State Farm 公司在 2011 年就开始运用 Usage-Based Insurance (UBI) 车险商业模式，与车联网厂商 Hughes Telematics 合作开展基于驾驶行为数据的保费模式[2]。可以看出，该项目的发起也体现了 State Farm 正在对未来进行探索性布局。



插图 2：UBI 保险模式

该项目是从汽车保险这一大规模消费市场的实际需求出发，着眼于未来车联网大数据环境下的驾驶风险评估系统，利用机器视觉和机器学习技术，搭建驾驶行为自动检测模型，其商业价值和应用前景值得期待。

本项目基于深度学习技术，使用了多个深度卷积神经网络模型，并借助迁移学习，对本项目的数据集进行了训练，最终取得了较理想的驾驶行为检测效果。

## 1.2 问题陈述

比赛举办方提供给我们已经分为 10 个类别的照片，示例如下图所示，作为机器学习模型训练所用。要求我们的模型能够对测试照片进行类别预测，以判断司机当前是处于哪种状态。

表格 1 10 类别及示例照片

		
C0 安全驾驶	C1 右手发短信	C2 右手打电话
		
C3 左手发短信	C4 左手打电话	C5 调收音机
		
C6 喝水	C7 伸手到后面	
		
C8 整理头发和化妆	C9 和乘客说话	

该问题实质上属于有监督机器学习(supervised learning)的分类方向(classification)，并且是计算器视觉(computer vision)范畴。模型的目标是利用大量的分类图片进行训练，来学到不同驾驶行为的特征，然后可以正确识别未见过的驾驶照片。

输入数据主要包括 22424 张训练用照片和 79726 张测试用照片，以及训练照片的列表（包含所属类别和司机 ID）。其中，照片尺寸为  $640 \times 480$ ，并经过 metadata 元数据去除处理（以保证问题是纯粹的机器视觉）。测试集还包含了一些扩充数据（不参与分数计算），以抵御手动标注测试集。此外，训练集和测试集是按照不同司机分割的。这提示我们，在后续训练时也应对训练集按照不同司机划分出验证集，来判断模型是否具备应用于未知司机的泛化能力。

## 1.3 评价指标

为量化基准模型和解决方案，采用 Kaggle 比赛 Private Leaderboard 的得分作为评估指标。该得分是，利用服务器上测试集的真实标签，对 69%比例的测试集计算得到的多类对数损失。其中，多类对数损失公式如下。

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

式中， $N$  为样本数量， $M$  为类别数量， $y_{ij}$  为表征样本  $i$  是否实际属于类别  $j$  的 0-1 指示函数， $p_{ij}$  为样本  $i$  在类别  $j$  上的预测概率。

这个评估标准对于问题本身、数据集以及解决方案来说都是合适：①问题本身是属于多分类问题；② 69%比例测试集多达 55000，足够用来验证模型泛化能力；③解决方案中还可以将该指标应用于划分出的验证集，便于超参数选取等。

## 2 分析

### 2.1 数据探索及可视化

#### (1) 类别平衡性分析

先对训练数据作一个初步的类别平衡性分析，如下图所示。可以看出，10 个类别分布比较均匀（每类数量主要在 2000~2500），这让我们可以省去重采样或类权值法等步骤。

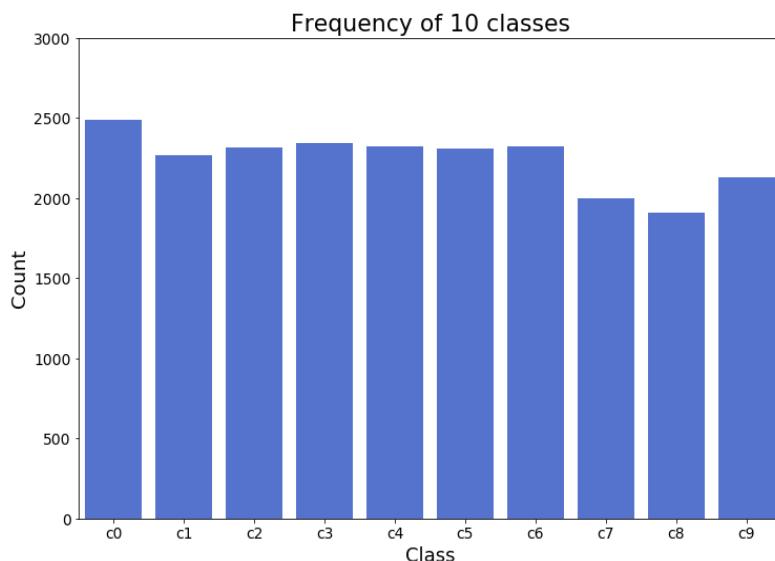


插图 3：类别平衡性分析

再针对训练数据中的 26 个司机，分别做类别分析，如下图所示。可以看出，大多数司机的类别分布比较均匀（类间数目差异在 25%内）。但是个别的如 p072、p051、p050，类别分布差异较大，这意味着后续作验证集划分时，如果只划分一次进行训练，有可能验证指标不够稳定、也不够充分。代码见 1a\_ClassDistribution.ipynb。

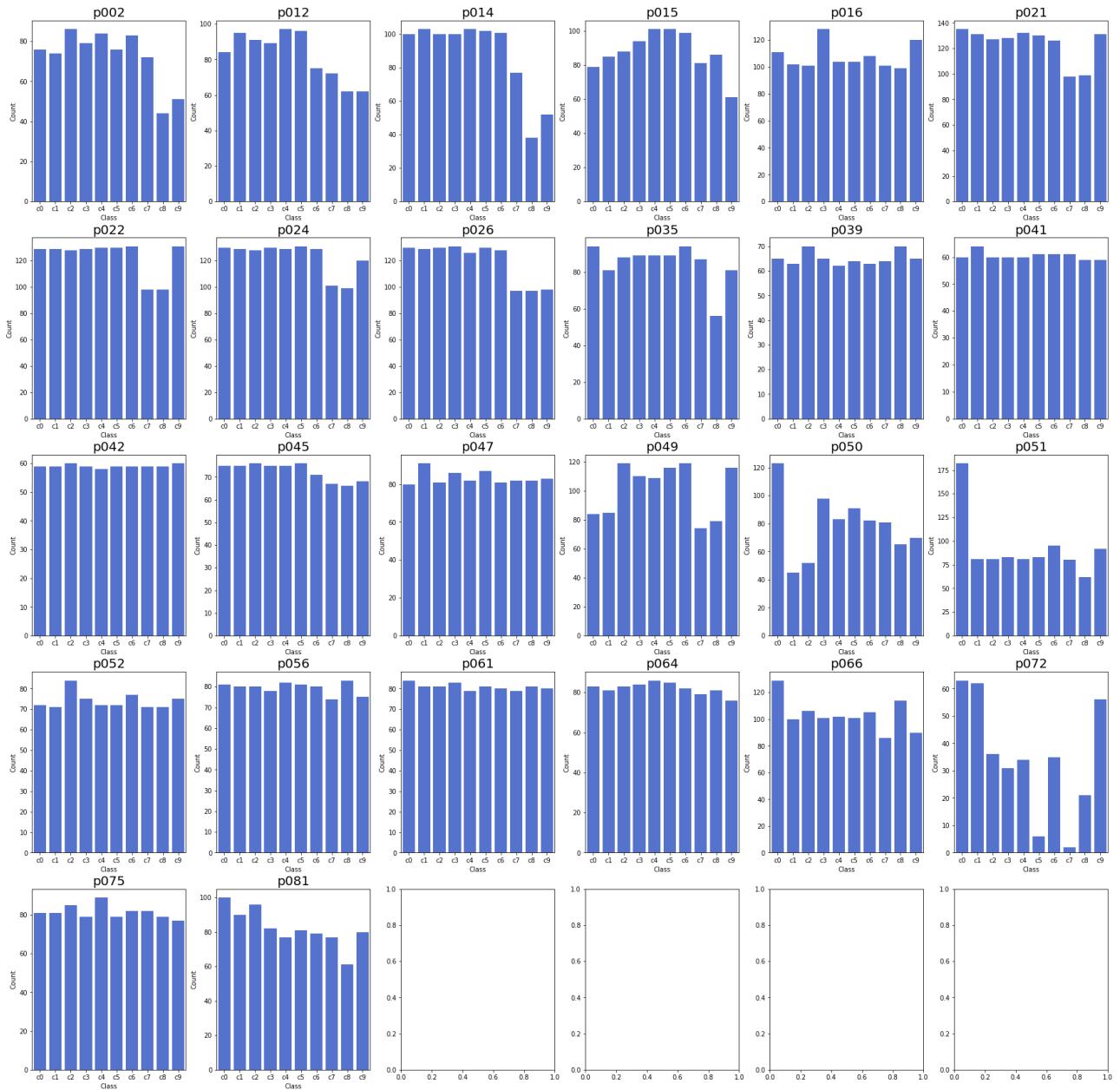


插图 4: 26 个司机类别分布

## (2) 数据集误差分析

该项目数据集的标签过程大致是这样的，被测司机按照指令做出一系列动作，然后从录制的一段段视频中分别提取出图像并标记为一类（同一个动作视频里的图像均为同一类）。这样使得数据集有可能出现一些图像是标定错误，或者是人眼也无法分类。

为了对模型最理想的结果有一个估计，有必要对数据集先进行人工的误差分析。本文随机抽取了 100 张图像，并进行人眼判别分类，再和标签对比。人工分类错误统计结果见表格 2，表中无法分类表示的是，认为不属于任何一类或者无法确定属于哪一类。具体示例见表格 3 所示。代码见 1b\_HumanError.ipynb。

表格 2 人工分类错误统计

标签项	数量	误判项
Talking (C9)	3	2 项为 Safe driving (C0)
		1 项为无法分类
Safe driving (C0)	2	1 项为 Talking (C9)
		1 项为无法分类
Makeup (C8)	2	2 项为无法分类
其他 93 项人工识别正确		

表格 3 错误具体示例



因此，预估数据集（包括训练集和测试集）标签的准确率在 93% 左右。进一步的，将 7 个误判项假定为 10% 正确率的话，根据前述公式，可以预估理想模型的多分类 loss 值在 0.16 左右。

## 2.2 算法和技术

项目的技术领域为机器视觉，更具体地说是交通工具内人体姿势与行为的图像机器学习。有必要提及的是，近年来深度学习的突破进展，为此类复杂问题的解决奠定了基础。自 1958 年 Frank Rosenblatt 提出感知机，人工神经网络的研究开始兴起，后又因多层感知机的训练瓶颈而陷于低潮，再到 1986 年 Rumelhart 提出反向传播算法迎来复苏，然而再次因为深层神经网络梯度消失或爆炸导致的训练困难，进入又一个寒冬[3]。直到 2006 年，Geoffrey Hinton 提出逐层训练法[4]，正式揭开了深度学习的序幕，伴随着计算能力的大幅提高（包括 GPU 的使用）和互联网大数据的趋势，深度学习开始在多个领域迅猛发展：2011 年微软和谷歌将神经网络引入语音识别、2012 年 Hinton 的学生在 ImageNet 图像识别竞赛夺冠、2014 年 Facebook 和百度将深度学习应用于翻译和图片搜索[5]。在计算机视觉的诸多领域，例如人脸识别、图像问答、物体检测、物体跟踪等，深度学习取得的效果远超传统机器学习算法。

而针对当前项目所涉及的图像分类问题，由于其状态空间庞大等因素，最适合地便是借助深度学习技术。目前在图片分类、检测及分割等方面应用最广泛的算法是，卷积神经网络 (CNN)。CNN 突出的特点在于，层与层之间通过感受野滤波器实现局部连接和权值共享，使

得网络结构简单、减少网络参数，也有利于通过多层卷积实现深层网络模型。近几年来涌现很多非常有效的卷积神经网络模型：自从 LSVRC-12 大赛上 Krizhevsky 提出 AlexNet 深度卷积神经网络模型，每年的 LSVRC 大赛都被卷积神经网络统治，例如 LSVRC-14 提出的 VGG16 模型和 GoogLeNet 模型、LSVRC-15 的冠军 ResNet 模型等。由于这些模型都已经在 LSVRC 数据集上进行了充分训练，并得到了很好的分类和检测的效果，很多研究者发现利用这些训练好的模型在其他图片分类的任务上也能得到很好的效果，这也就是迁移学习的利用。

表格 4 预训练模型

序号	模型	描述	训练权重	本项目用途
1	Vgg16[7]	ILSVRC2014 冠军	ImageNet	基准模型
2	ResNet[8]	ILSVRC2015 冠军	ImageNet	采用了 ResNet-50 迁移学习模型之一
3	InceptionV3[9]	GoogLeNet 的改进模型	ImageNet	迁移学习模型之一
4	Xception[10]	Inception 极致版本	ImageNet	迁移学习模型之一

本项目实现所使用的预训练模型，如表格 4 所示。这里简要对上述几个模型结构作一下简要介绍。（1）在 2014 年，VGG 模型由 Simonyan 和 Zisserman 提出。VGG 模型结构简单有效，前几层仅使用  $3 \times 3$  卷积核来增加网络深度，通过 max pooling（最大池化）依次减少每层的神经元数量，最后三层分别是 2 个有 4096 个神经元的全连接层和一个 softmax 层。（2）在 2015 年，ResNet 模型由 He 等人首先提出。与传统的顺序网络架构不同的是，其加入了恒等映射层，可以让网络在深度增加情况下却不退化，因此可以通过使用残差模块和常规 SGD（需要合理的初始化 weight）来训练非常深的网络。（3）Inception V3 模型来源于 Szegedy 等人在 2014 年提出的 Inception 模块，Inception 模块的目的是充当“多级特征提取器”，使用  $1 \times 1$ 、 $3 \times 3$  和  $5 \times 5$  的卷积核，最后把这些卷积输出连接起来，当做下一层的输入。Inception V3 模型提出了对 Inception 模块的更新，进一步提高了分类效果。（4）Xception 是由 François Chollet 提出的。Xception 是 Inception 架构的扩展，它用深度可分离的卷积代替了标准的 Inception 模块。

表格 5 硬件环境

序号	项目	描述
1	GPU	NVIDIA K80, 12GB 显存
2	CPU	Intel Xeon E5
3	内存	60GB
4	硬盘	30GB SSD

该项目实现使用的硬件环境，使用了 AWS EC2 p2.xlarge 实例[6]，具体如表格 5 所示。该项目实现使用的软件环境，是自行配置的 AMI 映像，具体如表格 6 所示。

表格 6 软件环境

序号	软件及版本	本项目用途
1	Ubuntu 16.04	操作系统
2	Anaconda 4.2	包与环境管理
3	Python 3.5	编程语言
4	Keras 2.0	深度学习框架前端
5	Tensorflow 1.3	深度学习框架后端
6	OpenCv 3.0	计算机视觉库
7	Cuda 8.0	GPU 架构
8	CuDNN 6.0	GPU 加速库
9	其他，如 Scikit-learn, Numpy, Pandas 等，不在此一一概述	

## 2.3 基准模型

为了和解决方案做对比，拟采用经过 ImageNet 预训练的 Vgg16 模型，并全新训练末端全连接层，作为基准模型。Vgg16 模型简洁和直观，在考虑使用卷积神经网络解决图像问题时，通常可以先使用 Vgg16 作为基准模型，来对数据集进行初步验证，然后再考虑对模型进行优化或使用层数更多的模型，并用于后续的客观对比。

基准模型采用了经过 ImageNet 预训练的 Vgg16 模型（不包含末端全连接层，全部冻结），并添加全局池化层、dropout 层、10 节点的全连接层（用于输出分类结果）。训练时，未进行图像增强，batch 采用 16，训练优化器采用 RMSprop（参数保持默认），验证集使用司机 ID 为 p014、p041、p042。

训练 2 轮后，验证集达到 loss 为 0.9457，acc 为 0.6916。第三轮验证集结果没有改善，故未采用。将测试集结果提交 Kaggle，得到 Private Score 为 1.69636。可以预期的是，下文所述的 fine-tune 模型的结果应该会比这个基准模型提高很多。

### 3 方法

#### 3.1 数据预处理

##### (1) 验证集划分

利用司机照片列表，对原始训练集所包含的 26 个司机照片，划分出 3 个司机照片作为验证集，剩余 23 个作为训练集。下文描述的各个模型都是每次重新划分。

另需注意的是，实现代码是采用了新建指向图片软链接的方法，不仅不占用内存，执行速度也非常快。代码见 2\_TrainValidSpilit.ipynb。

##### (2) 图片预处理

训练集图片尺寸为 480（高）×640（宽），在将其送入模型训练前，为减轻计算压力，有必要先进行缩放处理。针对用作基准模型的 Vgg16，训练图片缩放至 224×224；针对用作 fine-tune 的 ResNet 模型，训练图片缩放至 240×320；针对用作 fine-tune 的 Inception 和 Xception 模型，训练图片缩放至 330×440。后三个模型，将图片短边缩小到比原预训练尺寸（ResNet 为 224，Inception 和 Xception 为 229）略大，并保持了比例不变。

此外，Vgg16 和 ResNet 还要按照其在 Imagenet 预训练时所作的那样，进行像素归一化等预处理操作。我这里是对四个模型，调用 Keras 为其配置好的 preprocess\_input 函数处理。

### 3.2 执行过程

#### (1) 多个模型的模型选择

选择不同的预训练模型、数据增强、fine-tune 起始层、验证集，形成了如下表所示的多个训练模型。

表格 7 训练模型

模型名称	预训练模型	数据增强	Fine-tune 起始层/总层数	验证集
Model 1	ResNet50	DG1	77 /171	p021, p052, p081
Model 2	ResNet50	DG1	109 /171	p035, p045, p050
Model 3	ResNet50	DG3	99 /171	p016, p026, p061
Model 4	InceptionV3	DG1	133 /311	p014, p041, p042
Model 5	InceptionV3	DG2	133 /311	p014, p041, p042
Model 6	InceptionV3	DG3	165 /311	p015, p052, p066
Model 7	Xception	DG1	86 /132	p002, p075, p081
Model 8	Xception	DG2	76 /132	p026, p052, p081
Model 9	Xception	DG3	76 /132	p026, p052, p081

表中，数据增强按照强度分为多个等级，具体含义如下表所示。其中 DG1.5 为后文改进章节涉及。

表格 8 数据增强

项目	含义	DG1	DG1.5	DG2	DG3
rotation_range	图片随机转动的角度	10	12.5	15	15
width_shift_range	图片水平偏移的幅度	0.05	0.08	0.1	0.15
height_shift_range	图片竖直偏移的幅度	0.05	0.08	0.1	0.15
shear_range	图片剪切变换的角度	0.1	0.1	0.1	0.1
zoom_range	图片随机缩放的幅度	0.1	(0.85, 1.1)	(0.85, 1.1)	(0.85, 1.1)
channel_shift_range	随机通道偏移的幅度	/	5	10	10

模型选择中，fine-tune 起始层的选择，是参考了预训练模型的网络结构，如下图所示意。高亮数字为模型中选择作为 fine-tune 的起始层，在此之前的模型层参数全部冻结，在此之后模型层参数参与训练迭代更新。

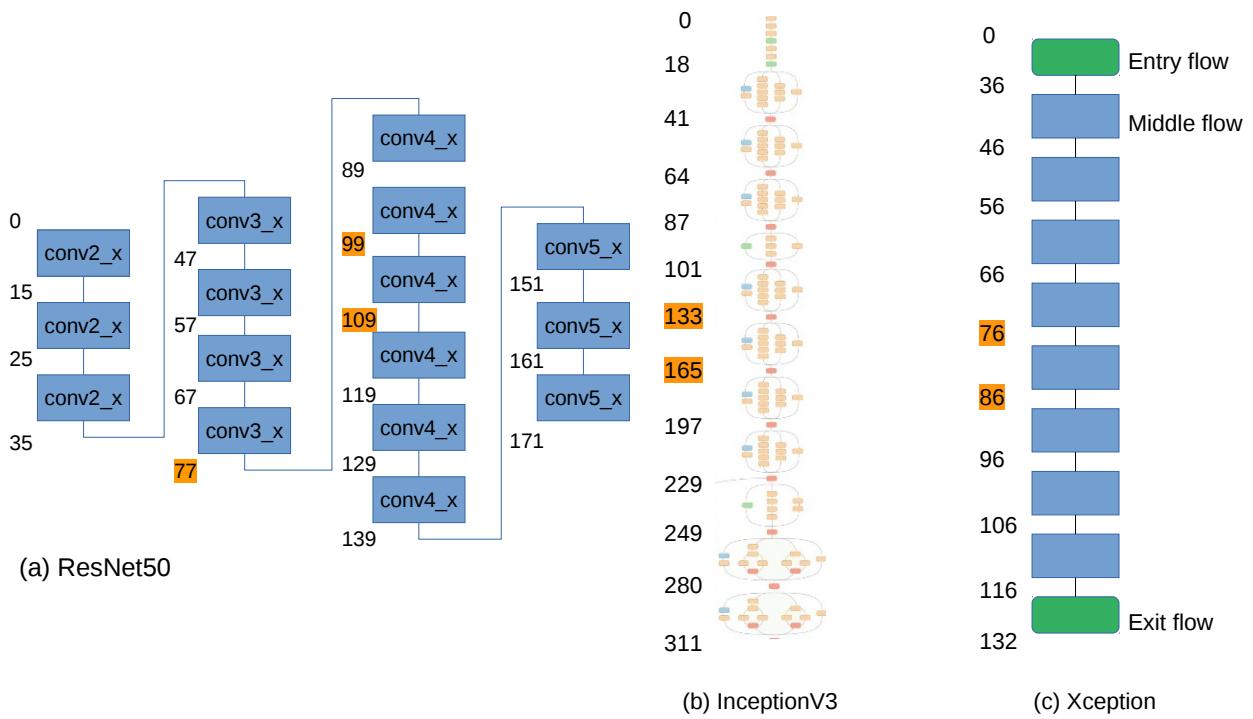


插图 5: 预训练模型层数示意

上述 9 个模型，模型结构的搭建，都统一按照插图 6 示意方式进行。引入预训练模型，并去除末端全连接层之后，依次添加全局平均池化层、Dropout 层、Dense 层。其中 Dropout 层 [11] 设置为丢弃概率 0.5，可以一定程度上缓解过拟合。其中 Dense 层为 10 节点，并设置激活函数为 softmax，以此作为 10 分类概率输出结果。

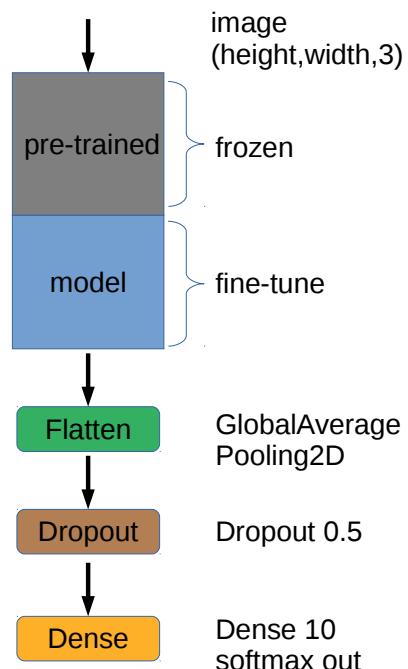


插图 6: 模型结构搭建

## (2) 多个模型的训练参数

在模型训练时，采用了每个 epoch 手工调节学习率的方法。

这是因为，经过前期的试验，多个 epoch 只设定一次学习率的情况下，很容易出现①学习率过高，训练集过拟合而验证集 loss 不降反升②学习率过低，训练需要 10 个或更多轮次才收敛③验证集 loss 上下振荡。具体分析来看，首先是因为为了模型性能所选择 fine-tune 层数较深，本身比较难训练，其次是数据量从司机数量角度看其实偏少（参与训练的是 23 个司机样本，要求泛化到未见司机）。从训练的实际表现可以看出，前期需要较高学习率，后期需要较低学习率。

为简化起见，将学习率的选择限定在下列数值：3e-4、1e-4、3e-5、1e-5、3e-6、1e-6、3e-7。将优化器的选择限定为 Adam 和 RMSprop。各个模型多个 epoch 的优化器参数如表格 9 所示。参数的选定依赖人工调试，这里作下简要说明。①首轮学习率的选取在此处一般选择 1e-4，我们知道的是模型如果全新训练一般默认选取 1e-3，而如果是 fine-tune 所有层一般选择 1e-5，我这里是为了尽量快地训练，故选择 1e-4，（Model6 和 7 选择了 3e-4，是为了加快收敛速度，这里 fine-tune 的层数也相对浅）。②第二轮和第三轮学习率的选择一般为前一轮的十分之一，如果发生验证集 loss 不降反升，则重新选择更低的学习率，而如果一直低至 3e-7 也无法使验证集 loss 下降，则停止训练，并将模型权重回滚至上轮。③ epochs rest 一列所示的 Model5 是一个特例，此处是因为第三轮的 1e-6 学习率训练过后效果明显，判断是仍有改善空间，故尝试了继续进行两轮 1e-6 的训练，该特例并未进行对比分析。

表格 9 优化器参数

模型名称	预训练模型	epoch 1	epoch 2	epoch3	epochs rest
Model 1	ResNet50	1e-4, Adam	1e-5, RMSprop	1e-6, RMSprop	\
Model 2	ResNet50	1e-4, Adam	1e-6, RMSprop	\	\
Model 3	ResNet50	1e-4, Adam	3e-7, RMSprop	\	\
Model 4	InceptionV3	1e-4, Adam	1e-5, RMSprop	1e-6, RMSprop	\
Model 5	InceptionV3	1e-4, Adam	1e-5, RMSprop	1e-6, RMSprop	(1e-6, RMSprop)×2
Model 6	InceptionV3	3e-4, Adam	3e-5, RMSprop	\	\
Model 7	Xception	3e-4, Adam	3e-5, RMSprop	3e-6, RMSprop	\
Model 8	Xception	1e-4, Adam	1e-5, Adam	\	\
Model 9	Xception	1e-4, Adam	1e-5, Adam	\	\

优化器的选择是，首轮均采用 Adam，之后倾向于采用 RMSprop，不过未作对比分析。从实践经验看，两个优化器本身是相似的，故性能表现未见明显差异。Adam(Adaptive Moment Estimation)，结合了 Adagrad 善于处理稀疏梯度和 RMSprop 善于处理非平稳目标的优点，比较适合大数据集和高维空间；RMSprop 其实依然依赖于全局学习率，适合处理非平稳目标 [12]。在本项目中，是属于高维空间、稀疏数据，模型网络较深和复杂，那么为了更快地收敛，比较适宜地是采用学习率自适应优化方法，因此所选择的 Adam 和 RMSprop 是比较合适的。

需说明的是，采用每个 epoch 手工调节学习率的方法，虽然可以让模型以较少轮次就收敛到合理水平，但是相较以固定的较小学习率并训练较多轮次的方法而言，往往不能获得更好的结果。另外，每个轮次依照验证集结果来进行手动调节学习率，也在一定程度上对验证集产生过度拟合，对泛化性能有些不利影响。当然了，这两点不足，可以通过下文的多模型结果集成来弥补。

此外，根据模型的不同、fine-tune 层数的深浅，会调整 batch size：一般尽可能定为 128，若内存不满足则降至 64 或 48。根据 batch size 的大小，也会相应调整一个轮次的迭代数：batch size 为 128 时，迭代数选为正常值，即训练总图片数量除以批大小；若为 64，迭代数选择 75% 比例，若为 48，迭代数选择 50% 比例。此处迭代数调整为经验选择，因为 batch 较小时，还按照常规方法会使得单个轮次的总迭代数偏大，容易造成过拟合，不利于前述的人工调整学习率。

各个模型的训练，详细信息见代码，4\_Model(1-9)\_xxx.ipynb。

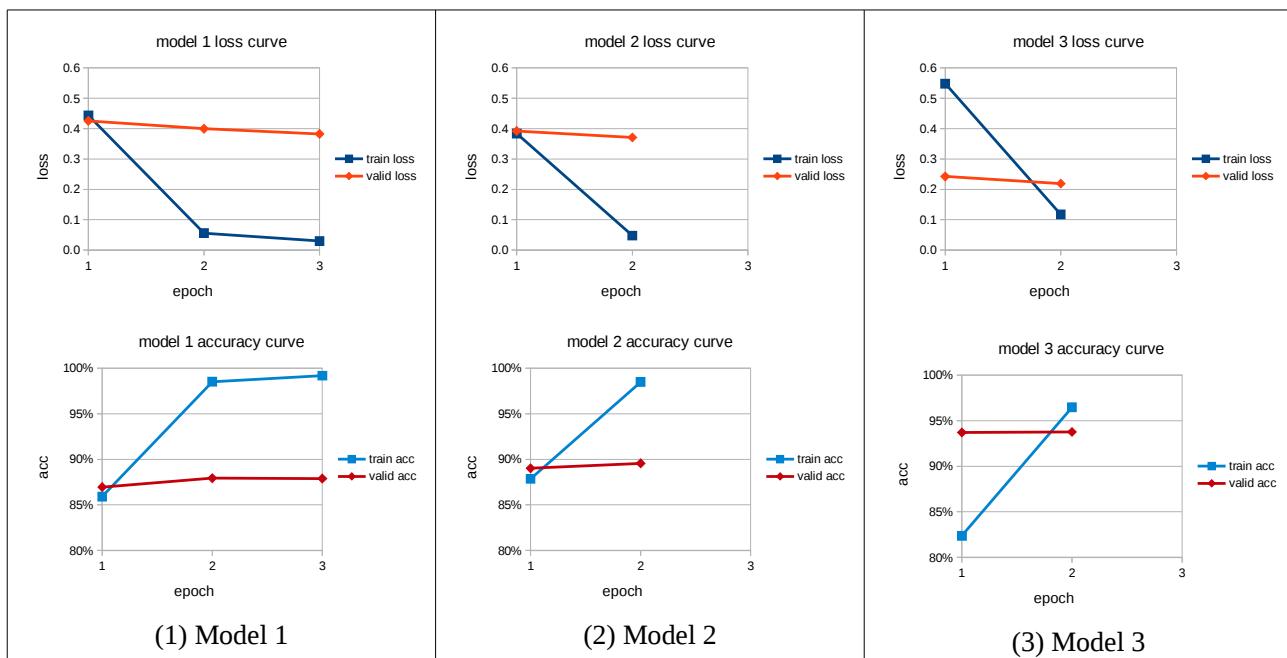
### (3) 多个模型的训练结果

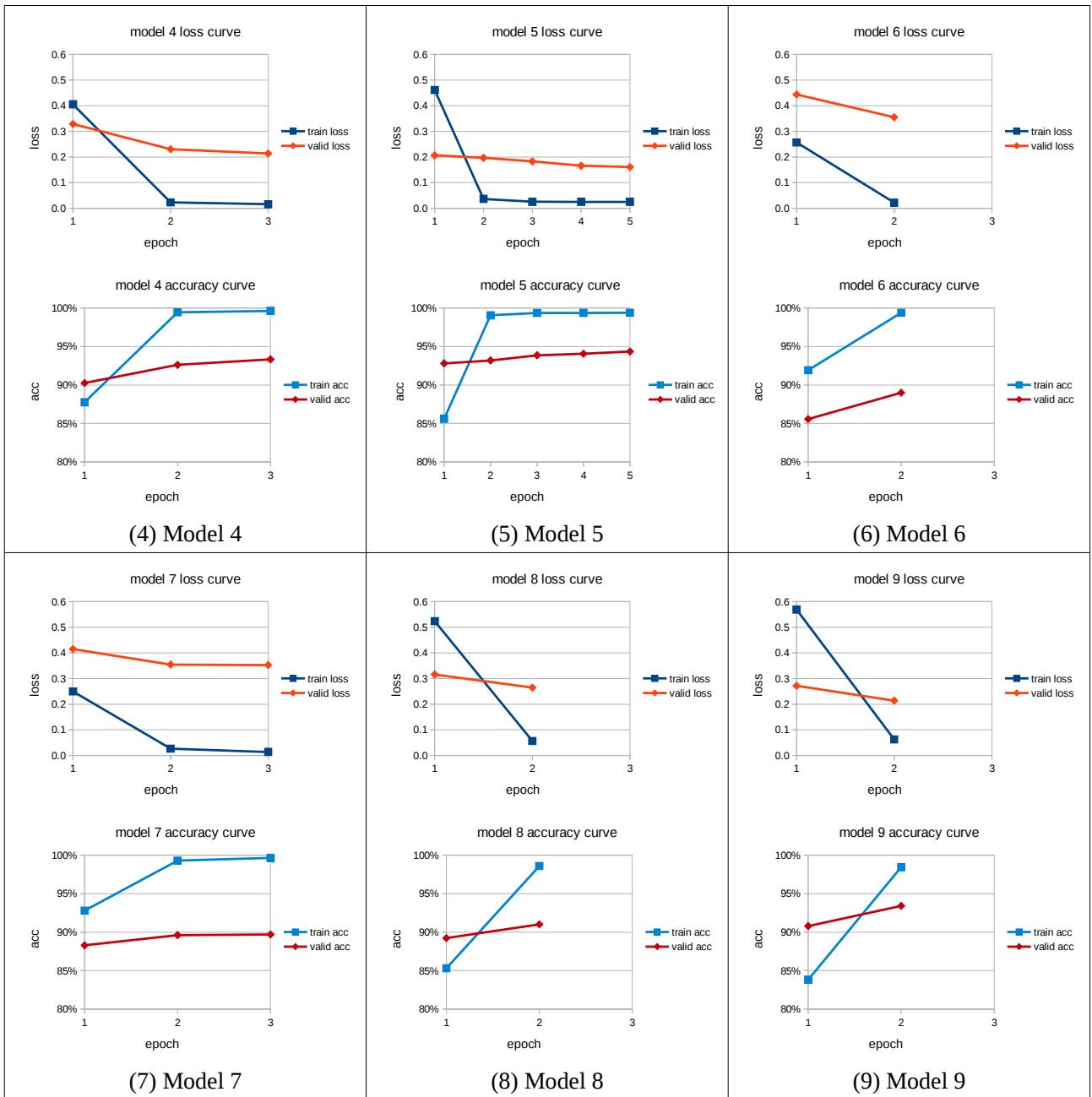
多个模型的训练结果，原始数据及 loss 和 acc 曲线图如下两表所示。

表格 10 训练结果原始数据

Model	Metric	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
1	train loss	0.4437	0.0560	0.0298		
	valid loss	0.4257	0.3996	0.3826		
	train acc	0.8591	0.9852	0.9918		
	valid acc	0.8696	0.8793	0.8789		
2	train loss	0.3838	0.0476			
	valid loss	0.3923	0.3710			
	train acc	0.8785	0.9849			
	valid acc	0.8901	0.8956			
3	train loss	0.5478	0.1174			
	valid loss	0.2421	0.2187			
	train acc	0.8237	0.9647			
	valid acc	0.9371	0.9377			
4	train loss	0.4053	0.0231	0.0158		
	valid loss	0.3286	0.2303	0.2138		
	train acc	0.8775	0.9945	0.9963		
	valid acc	0.9025	0.9262	0.9334		
5	train loss	0.4613	0.0367	0.0254	0.0251	0.0251
	valid loss	0.2063	0.1965	0.1824	0.1660	0.1608
	train acc	0.8561	0.9905	0.9935	0.9937	0.9938
	valid acc	0.9281	0.9319	0.9387	0.9406	0.9435
6	train loss	0.2567	0.0219			
	valid loss	0.4439	0.3548			
	train acc	0.9193	0.9939			
	valid acc	0.8557	0.8899			
7	train loss	0.2496	0.0265	0.0137		
	valid loss	0.4146	0.3546	0.3528		
	train acc	0.9280	0.9930	0.9964		
	valid acc	0.8828	0.8962	0.8970		
8	train loss	0.5236	0.0565			
	valid loss	0.3158	0.2648			
	train acc	0.8530	0.9859			
	valid acc	0.8923	0.9102			
9	train loss	0.5687	0.0625			
	valid loss	0.2717	0.2140			
	train acc	0.8382	0.9844			
	valid acc	0.9079	0.9342			

表格 11 训练结果曲线图





从结果可见，训练完成后，验证集 loss 可下降至 0.16~0.38，精度可达 87%~94%。从实践中发现，验证集的选择不同，直接决定了验证集 loss 和精度能达到的最好水平，而训练集的 loss 和精度总是能够很快收敛。同时，预训练模型、数据增强的不同和 fine-tune 层数的深浅，共同影响了手工调节的学习率。另外，此处发现 ResNet-50 模型相比较另两个模型，显得更难训练，应该是因为模型结构本质不同造成的，而且最初手动调节学习率的经验是来自于对 InceptionV3 的试验。

### 3.3 模型集成

上述 9 个模型，从单个模型角度来讲，普遍存在下列问题：①还有 3 个司机样本是作为验证集，未参与训练；②模型的训练一般仅仅 2 到 3 轮，可能训练不充分而不能达到最优；③模型的每个轮次学习率是依照验证集手工调节，有可能对验证集过度拟合。

因此，为完善结果，提出的最终模型是上述 9 个模型的集成。采用集成方法，就可以弥补了上述缺陷，因为 9 个模型的验证集一般都是不相同的，并且合计轮次也达到了 22 轮。此外，还具备了下述优势：选用了 3 种预训练模型、3 种数据增强模式、每种预训练模型至少 2 个 fine-tune 层选择，大大提高了多样性，有利于集成后模型的泛化性能。

这里采用的是最简单的集成方法，即将 9 个模型的测试结果取平均值，代码见 5b\_Merge Submission.ipynb。此外，9 个模型的测试结果也在此之前提交到 Kaggle 上，代码见 5a\_Clip Submission.ipynb。需说明的是，代码中会对结果会进行下限值截断处理，也就是每个类的预测概率限制在[0.001, 1]，以避免多类对数损失对小概率的过大惩罚，这个技巧可以微弱地提高得分。集成模型的结果，连同 9 个模型的提交得分，见后文所示。

## 4 结果与讨论

### 4.1 模型的评价与验证

提交至 Kaggle 的 Private Score 得分见下表所示。

表格 12 Kaggle Private Score

	Valid loss	Kaggle Private Score
Model 1	0.3826	0.30217
Model 2	0.3710	0.31493
Model 3	0.2187	0.30075
Model 4	0.2138	0.25513
Model 5	0.1608	0.27546
Model 6	0.3548	0.30807
Model 7	0.3528	0.30464
Model 8	0.2648	0.28098
Model 9	0.2140	0.25383
<b>Model Ensemble</b>	/	<b>0.20366</b>

从结果可见，前述 9 个模型的 Private Score 得分分布在 0.253~0.315，平均值为 0.288。而集成后模型得分，直接降至 0.20366。结果表明，直接取平均值的集成方法，方法虽简单，效果足够明显。

## 4.2 合理性分析

相比较基准模型（Vgg16，Private Score 为 1.69636），最终模型的结果达到了 0.2 的 loss，可以说是较好地发挥了预训练模型的潜力，借助了迁移学习的优势，很好地解决这个项目司机驾驶行为分类的问题。

当然了，最终模型的结果距离前文所说的理想模型（loss 值 0.16）仍有一定差距。初步分析，主要原因在于，①没有对预训练模型微调全部层，未能完全体现预训练模型深层网络的表现能力；②单个模型训练轮次偏少，未能使单个模型性能达到最优。更进一步的分析，有待下文的可视化进行讨论。

## 4.3 结果可视化

上文的模型 9 是在验证集和 Kaggle 上表现最好的模型之一，故使用该模型来进行结果可视化，并进一步讨论。代码见 6a\_ResultAnalysis.ipynb。

使用训练好的模型 9，在验证集上评估，得到下表所示。可以看到，针对总样本为 2336 的验证集，模型 9 获得了平均为 0.93 的准确率和 0.93 的召回率，模型表现较好。不过，仍然可以看出，个别类别仍然有改进空间，比如 c0、c8、c9 等。

表格 13 验证集分类报告

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
c0	0.87	0.79	0.83	274
c1	1.00	0.99	1.00	271
c2	1.00	0.99	1.00	246
c3	0.92	0.96	0.94	240
c4	0.99	0.94	0.96	242
c5	0.99	1.00	1.00	217
c6	1.00	0.93	0.96	243
c7	0.86	0.99	0.92	173
c8	0.79	0.85	0.82	201
c9	0.84	0.86	0.85	229
<b>avg/total</b>	<b>0.93</b>	<b>0.93</b>	<b>0.93</b>	<b>2336</b>

更进一步的，作出混淆矩阵，如插图 7 所示。可以看出，容易发生混淆的是：c0 误判为 c9、c3、c8，c9 误判为 c0，c8 误判为 c7，c6 误判为 c8，c4 误判为 c8。再回头看下前文所述的人工分类错误的类型，较多的是在 c0、c8、c9 三个发生混淆，可见还比较一致。那么，可能的改进空间或许在 c6 误判为 c8、c0 误判为 c3、c4 误判为 c8 这三种。

Confusion matrix, without normalization										
	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	216	0	0	12	1	0	0	0	10	35
c1	1	269	0	1	0	0	0	0	0	0
c2	0	0	244	0	0	0	0	0	2	0
c3	2	0	0	230	1	1	0	0	4	2
c4	0	0	0	5	227	0	0	0	10	0
c5	0	0	0	0	0	217	0	0	0	0
c6	0	0	0	0	0	0	225	0	18	0
c7	0	0	0	0	1	0	0	172	0	0
c8	1	0	0	0	0	0	1	27	171	1
c9	28	0	0	2	0	1	0	0	2	196

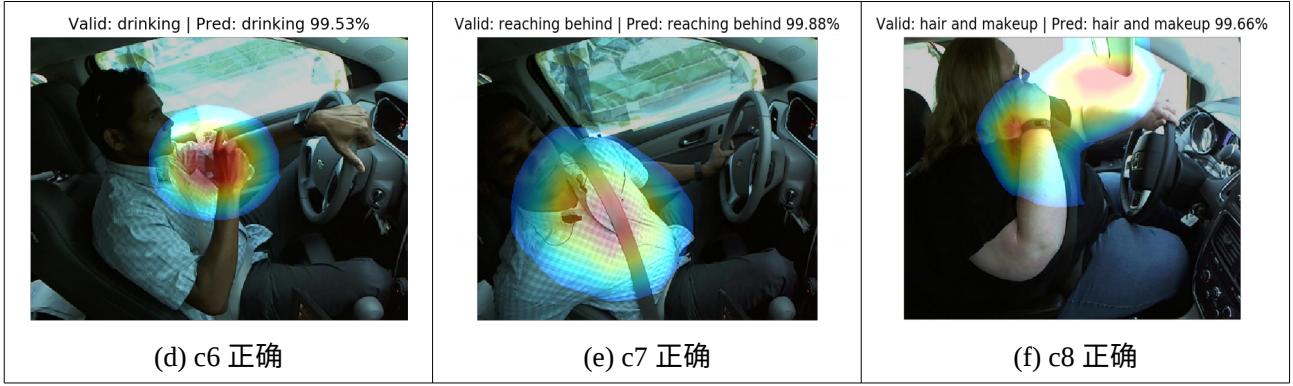
插图 7: 混淆矩阵

为了可以更直观地进行可视化，引入 Class Activation Mapping[13]的方法。CAM 对物体定位的核心思想是，CNNs 的高层卷积层往往记忆了目标分类的位置信息。CAM 具体实现是，将某深层卷积层的输出，乘以这一层对应分类的权重，再映射回图像空间，这样就得到目标分类的可视化图像。下文针对上述分类结果，给出多个 CAM 示例，以便讨论。

表格 14 所示的是预测正确的类激活图。可见，模型对于人工也容易判断的几类，比如发短信、打电话等，特征的定位都比较准确。额外需要注意的是，c7（伸手到后面）的定位主要在胸膛和两肩部位，应该是据此判断出了侧身姿态，不过却不够注重后伸的手臂（另一方面也可能因为照片中看不到后伸的手）；c8（化妆和整理头发）的定位还包括了后视镜，这一点并没有错，但是该类中还有些是没有放下后视镜而在整理头发的，有可能会因此影响模型泛化能力。

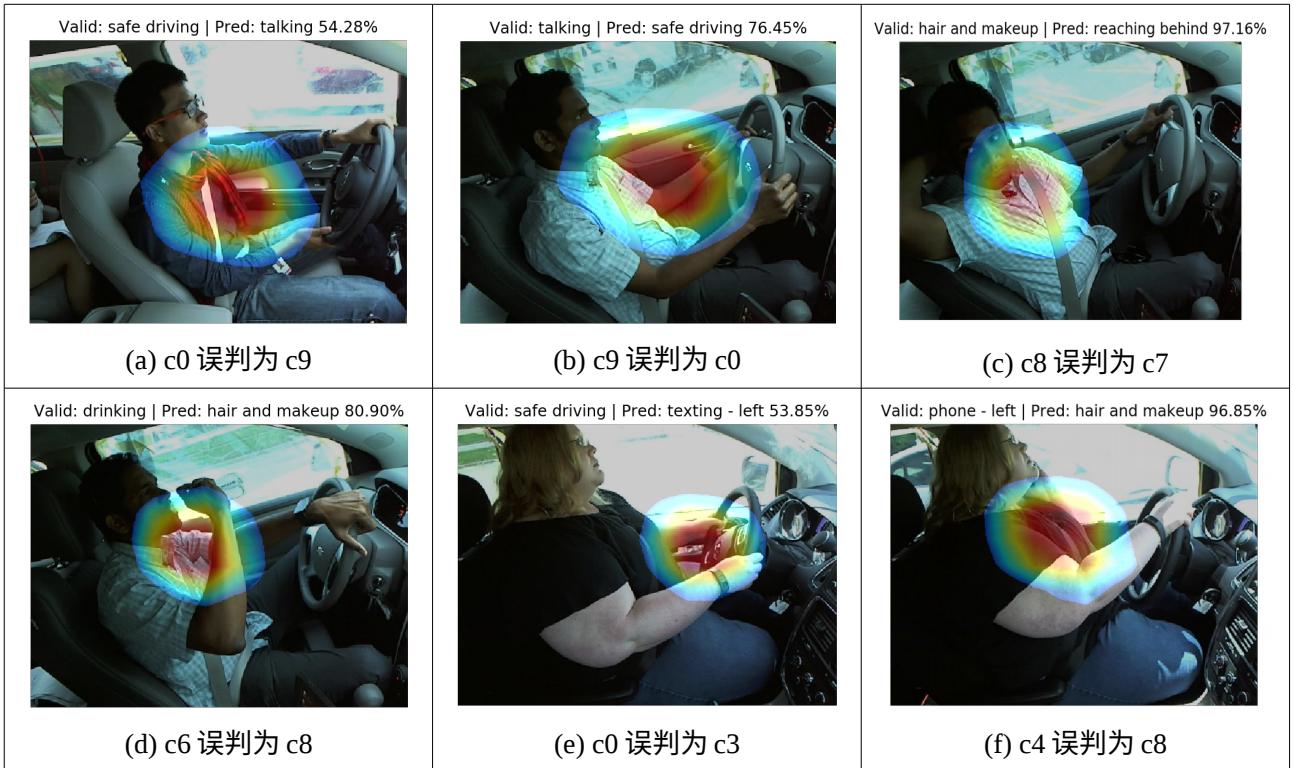
表格 14 预测正确的 CAM 示例





表格 15 所示的是预测错误的激活图。其中 c0、c8、c9 主要是因为数据集自身的缘故。(d) 图所示的，应该是由于饮料瓶颜色透明而没有被识别；(e) 图的定位错误比较难解释，有可能黑色手机和方向盘产生混淆；(f) 图是因为左手的手机被遮挡了，而产生误分类。

**表格 15 预测错误的 CAM 示例**



#### 4.4 如何改进

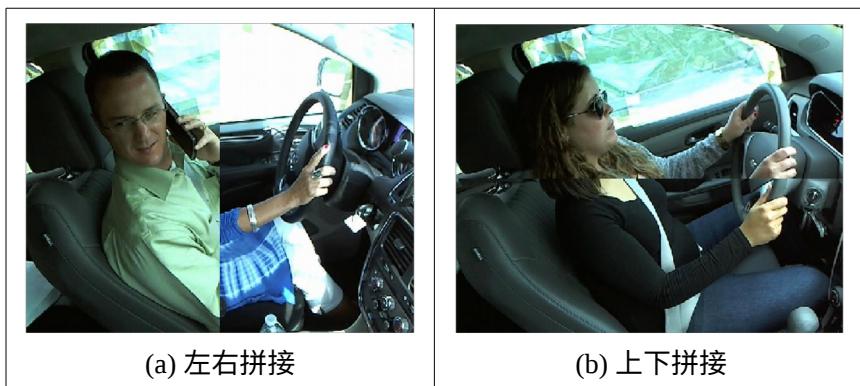
通过对结果的可视化及讨论，我们可以看到模型结果的误分类其实核心问题仍在于训练样本。训练样本的不充分，会使得模型学到了局部的特征、个例的细节，而没办法学到隐藏其中的规律。那么，如何针对此进行改进呢？

这里采用了本次 Kaggle 比赛排名第 5 名 DavidGbodiOdaibo 所提出的一个思路，即同类别的图像拼接形成新样本。他是将宽度 640 像素图像，分成左部分 240，右部分 400，再进行拼

接形成 500 万张新样本。我这里是直接在数据增强前进行实时的图像拼接，并做了改进，即不仅有左右拼接，还有上下拼接，详细代码见 my\_image.py。

拼接效果如下所示。需注意的是，不论是左右拼接，还是上下拼接，拼接前的两张照片都是属于同一类的，拼接后样本类别也是定义为该类。使用这样的拼接图片作为样本，就可以极大地抑制不相关物体的干扰（比如后排的记录员、后视镜、车的内饰、司机性别肤色等），而让模型将注意力更多地放在更为宏观并能体现规律的事物（比如头和两手的相对位置等）。

表格 16 同类拼接样本示例



## 5 改进及最终结果

### 5.1 改进措施

表格 17 训练模型及参数

模型名称	预训练模型	数据增强	训练参数
Model 11	ResNet50	DG1	Epoch1: Adam( $lr=1e-4$ ) Epoch2: RMSprop( $lr=1e-5$ ) Epoch3: RMSprop( $lr=1e-6$ )
Model 12	ResNet50	DG1.5	
Model 13	ResNet50	DG2	
Model 14	InceptionV3	DG1	
Model 15	InceptionV3	DG1.5	
Model 16	InceptionV3	DG2	
Model 17	Xception	DG1	与上同，且新增 Epoch4: RMSprop( $lr=1e-6$ )
Model 18	Xception	DG1.5	
Model 19	Xception	DG2	

相较前述模型，作了以下改进。

- (1) 采用“同类拼接”样本
- (2) fine-tune 所有层；
- (3) 训练所有样本，不留验证集。

相较前述模型，还有余下变化（但未验证是否是改善）。

- (1) 训练轮次的参数做了调整，将同一类预训练模型统一，见表格 17；
- (2) 数据增强中的 DG3，使用 DG1.5 替代（详细参数见前文表中所列，可以理解为数据增强强度介于 DG1 和 DG2 之间）；
- (3) batch 大小都减小至 48，甚至 16，由于显存的限制。

## 5.2 改进结果

9 个改进模型的测试集结果、及其集成结果（仍按前述方法作平均），提交至 Kaggle 的 Private Score 得分见下表所示。改进后，9 个模型的 Private Score 得分分布在 0.199~0.250，平均值为 0.224（同比改进前下降 0.064）。而集成后模型 loss 为 0.16805（同比下降 0.036），排名可列第 25 位，进入前 2%。

**表格 18 Kaggle Private Score**

	<b>Kaggle Private Score</b>
Model 11	0.24993
Model 12	0.25041
Model 13	0.22136
Model 14	0.22956
Model 15	0.21450
Model 16	0.19922
Model 17	0.22757
Model 18	0.20586
Model 19	0.21371
<b>Model Ensemble</b>	<b>0.16805</b>

## 5.3 进一步讨论

在此之后，还进行了增大集成模型数量的尝试，即加入采用了 DG3 数据增强的模型，发现集成结果的得分可以进一步下降。由于时间和精力有限，没有作更进一步试验。这里简单说一下，如果需要进一步完善，可以做的措施如下（根据 Kaggle 已有结果，预计能达到最好效果是 loss 为 0.15 左右）。

- (1) 再做 6 个或更多的不同数据增强样式的模型，加入集成结果；
- (2) 采用其他种类模型结构，比如 Vgg、InceptionResNetV2 等。
- (3) 增加大量训练轮次，并调整学习率，甚至不采用预训练模型，而全新训练（因为利用拼接法，可以生成大量样本）

## 6 结论

本项目针对司机驾驶行为检测问题，经过初期数据探索与分析，采用深度卷积神经网络和迁移学习技术，搭建了9个多样性好的机器学习模型并集成结果，再经结果可视化与讨论、提出并实现了同类图片拼接等等改进措施，最终模型在测试集上取得了较理想的成绩。

## 参考文献

1. [State Farm Distracted Driver Detection \(Kaggle\)](#)
2. [UBI 车险海外案例简析：State Farm（车云网）](#)
3. [Andrey Kurenkov. A 'Brief' History of Neural Nets and Deep Learning.](#)
4. Hinton G E, Osindero S, Teh Y W. A fast learning algorithm for deep belief nets[J]. Neural computation, 2006, 18(7): 1527-1554.
5. [回溯深度学习革命，盘点 16 大历史时刻（搜狐网）](#)
6. <https://aws.amazon.com/cn/ec2/instance-types/>
7. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale imagerecognition. In ICLR, 2015.
8. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.
9. C. Szegedy, W. Liu, Y. Jia, et al. Rabinovich. Going deeper with convolutions. In CVPR, 2015.
10. François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv preprint arXiv:1610.02357, 2016.
11. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929-1958.
12. [深度学习最全优化方法总结比较（知乎网）](#)
13. B. Zhou, A. Khosla, A. Lapedriza, et al. Learning Deep Features for Discriminative Localization. arXiv preprint arXiv:1512.04150, 2015.