

# 司机驾驶行为检测

机器学习工程师纳米学位毕业项目

丁南

2017 年 2 月 19 日

## 1. 定义

### 项目概述

该项目是 Kaggle 上 State Farm 公司举办的一个竞赛项目，其目的是得到一个能检测司机驾驶行为的模型。

随着移动互联网的发展，司机在驾驶过程中收到的干扰越来越多，在汽车上安置「驾驶提醒」报警的需求越来越迫切。为此，State Farm 在很多汽车上的仪表盘上安放了摄像头，截取了很多司机驾驶过程中的 2D 图片，State Farm 希望 Kaggle 的参赛者能基于这些驾驶图片，训练一个能检测驾驶行为的模型。

本项目使用了近几年在 LSVRC 大赛上取得突破性进展的深度卷积神经网络模型，在这些预训练模型的基础上，利用迁移学习的能力对本项目的数据集进行了训练，最终取得了不错的检测效果。

### 问题陈述

该项目本质上是一个监督分类学习的问题。训练数据集已经做好了分类标注。模型的目标是学习不同驾驶行为的特征，以致能正确识别一个未见过驾驶图片。

本项目首先按不同的驾驶行为对训练集进行分类，随后将图片和分类转换成 numpy array 和 categorical variables，并按 Tensorflow 的 ordering 进行重排。数据准备好之后，构造一系列 CNN 模型，利用交叉验证对数据进行训练，记录每次迭代的 Loss 和

Accuracy。训练数据处理完后，模型被用于预测测试数据，随后将预测结果保存成 csv 文件，csv 每一行记录一个被测图片的 10 种行为分类的概率。最后将该 csv 文件上传到 Kaggle，Kaggle 根据该项目的判断指标来对预测结果进行评分。

## 评价指标

该项目是多分类的问题，该问题的评价指标主要有两个，预测结果损失以及模型的训练和预测时长。

- Log Loss: Kaggle 对预测结果的评分使用 multi-class logarithmic loss，也称 cross-entropy，计算公式为：

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

- Time: 一个预测算法能应用到实际产品中，训练和预测时长也很关键。

## 2. 分析

### 数据集分析及可视化

State Farm 为参赛者提供了训练和测试数据集，所有数据集均是从车载摄像头录像中截取下来的静态图片。其中训练集已经做了基于 label class 的分类，分类为 [ c0, ..., c9 ]，其含义依次是：

- c0: 安全驾驶
- c1: 右手打字
- c2: 右手打电话
- c3: 左手打字
- c4: 左手打电话
- c5: 调收音机
- c6: 喝饮料
- c7: 拿后面的东西

- c8: 整理头发和化妆
- c9: 和其他乘客说话

训练集中相同分类的图片在同一个文件夹中，各个分类的图片数量分布见图 1。测试集则没有做分类，一共包含 79721 张静态图片。数据集的图片尺寸均是 640x480。

除了图片数据集，该项目还提供了司机 id 列表，该列表包含了数据集中图片对应的司机 id 以及对应的分类，从图 2 可以简单看出不同司机的图片分类的分布。

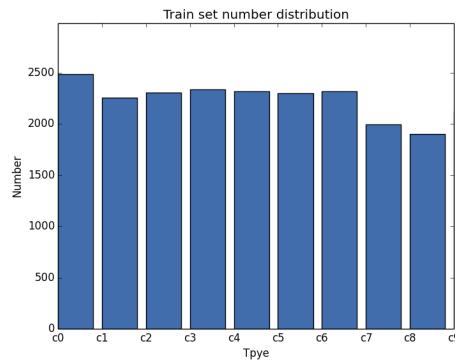


图1：训练数据集的分类分布

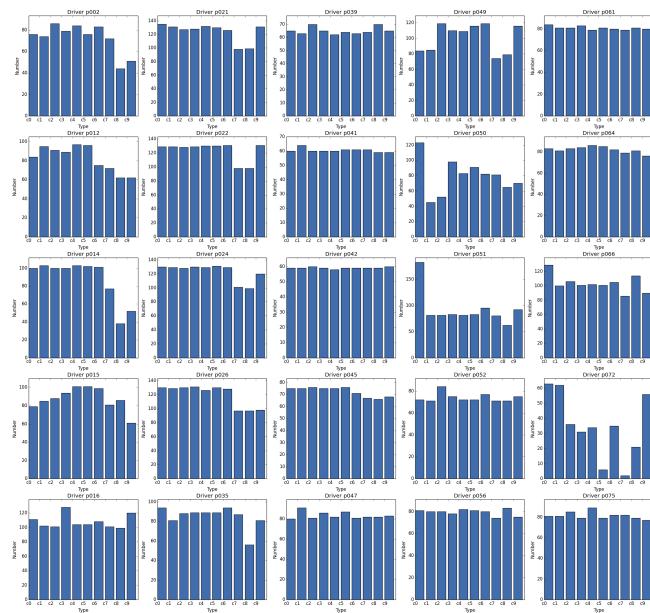


图2：不同司机的图片分类分布

## 算法和技术

该项目的最终目的是根据训练集，对测试集的图片进行分类，也就是说是一个基于监督学习的分类识别问题。虽然监督学习分类算法有很多，但图片分类的许多复杂的特性使得许多传统的监督分类学习算法的并不适用，例如状态空间巨大、图像含义在不同位置的平移不变性（translation invariance）等。

目前应用于图片分类、检测及分割最广泛的算法即是卷积神经网络，以下用 CNN 来代指卷积神经网络。与普通的前馈神经网络不同，CNN 受到动物的视觉皮层实验的启发，不同层之间的神经元并不是全连接，后一层神经元的刺激（输入）只与前一层部分神经元的输出有关，这一部分起作用的神经元也被称作感受野（receptive field）。感受野作用于神经元的计算方式类似于数学上的卷积操作，这也就是卷积神经网络名字的来源。

CNN 突出的特点有，层与层之间局部连接导致网络 parameters 大大减少，作用于感受野的过滤器在整个图片上的 parameters 共享导致 CNN 有很好的物体平移不变性，网络结构简单清晰有利于对网络内部结构进行可视化分析等等 [1]。

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 [2] 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 [3] 和 GoogeLenet 模型 [4]、LSVRC-15 的冠军 ResNet 模型 [5] 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（pretrained model），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。[6][7]

该项目的实现使用 Numpy 来做数据的提取和转换；使用 Scikit-learn 来做计算结果指标以及交叉验证；使用 Keras 来做深度学习的模型构建和训练，其后端使用的是 Tensorflow。

该项目实现使用的计算资源是 AWS EC2 p2.xlarge 实例，该实例有 4 个 CPU，60GB 内存，1 个 NVIDIA K80 GPU，12GB GPU 内存。机器系统为 Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-105-generic x86\_64)。

## 基准指标

虽然该项目在 Kaggle 上已经结束了，但我们仍可以查看该项目上所有的参赛者的 Leaderboard。该项目的 Leaderboard 一共有 1440 组参赛者，Public Leaderboard 的评价标准使用 19% 的测试集来计算其 log loss，而 Private Leaderboard 的评价标准使用剩下的 81% 测试集来计算最终的 log loss。指数越小的团队排名越高，最终的排名以 Private Leaderboard 为准。本项目的目标是，测试集的分类结果能排在 Private Leaderboard Top 10%，争取能进入前 100 名。

备注，Kaggle 给出的 sample submission benchmark 在 Private Leaderboard 得分为：2.30259。

## 3. 方法

### 数据预处理

对于大部分机器学习项目来说，实现的第一步一般都是对数据进行预处理。

首先先将所有的训练集按照分类来进行数据提取，提取图片的同时也提取了该图片对应的分类，保证图片和分类总是一一对应。

其次需要对图片进行数据转换和处理，为了减少数据量，现将图片进行缩放，为了能使用预训练模型，我们将图片的尺寸缩放成预训练模型能接受的尺寸。使用 VGG、ResNet 模型的话，将图片缩放为 224x224；使用 Inception 模型的话，将图片缩放为 299x299。然后将图片转换为 Numpy array，image ordering 使用 'tf'。最后对图像像素进行零平均（zero-mean）处理。

第三，将图片的分类进行数据转换，由于我们训练使用的指标是 cross entropy，所以我们将图片分类的数值转换成二进制的分类矩阵。

最后，我们需要将提取出的数据进行乱序排列，以免收到原始数据顺序的干扰。

## 实现

该项目的实现尝试了以下几种方法，自己构造的简单的卷积神经网络、VGG16 预训练模型、InceptionV3 预训练模型、ResNet50 预训练模型。

### 简单的卷积神经网络

为了测试一下卷积神经网络的效果，首先构建了一个简单的卷积神经网络，该网络有 2 个卷积层、1 个 max pooling 层，1 个全连接层和最后输出的 softmax 层。2 个卷积层的 filters 依次是为  $3 \times 3 \times 32$ ,  $3 \times 3 \times 32$ ，activation 均为 Relu，并且均使用 same padding。2 个卷积层输出使用 Dropout。Max pooling 层的 stride 和 size 均为 2。全连接层的神经元个数为 128，该层输出使用 Dropout。为了减少该简单模型的参数数量，将图片的尺寸缩放到  $96 \times 96$ 。该卷积层网络结构如图 3 所示。

在编译阶段，optimizer 使用 SGD，loss function 使用 categorical cross entropy，metrics 使用 accuracy。

其他的 hyper-parameter 有：

- 训练的 batch size: 64
- 训练迭代次数：5

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 96, 96, 32)	320	convolution2d_input_1[0][0]
convolution2d_2 (Convolution2D)	(None, 96, 96, 32)	9248	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 48, 48, 32)	0	convolution2d_2[0][0]
dropout_1 (Dropout)	(None, 48, 48, 32)	0	maxpooling2d_1[0][0]
flatten_1 (Flatten)	(None, 73728)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 128)	9437312	flatten_1[0][0]
dropout_2 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 10)	1290	dropout_2[0][0]
<hr/>			
Total params: 9,448,170			
Trainable params: 9,448,170			
Non-trainable params: 0			

图 3：简单 2 层卷积神经网络网络结构

## 使用 Pretrained VGG16

上一小节我们利用自建的卷积神经网络进行了测试，虽然得到了一些效果，但并没有达到预期的目标。对于图片分类而言，为了增加分类效果，一种有效的措施就是增加卷积神经网络的层数。但层数越深，网络 parameters 越多，权重空间变得巨大 (exponential)，由于 curse of dimensionality [8] 的影响，训练深度网络所需要的数据量也是巨大的。另外，网络结构越深，越容易出现 overfitting。简单来说层数越深也就越不容易训练。我们知道对于卷积神经网络，低层的权重记忆了很多通用的特性，比如颜色、边缘等信息，而高层的权重则记忆了具体特性，例如物体形状、大小等信息。所以我们可以利用一个已经经过大数据集充分训练的模型，保留该模型的底层 weights，来微调该模型的高层 weights。在微调高层 weights 后，我们也可以微调一些低层 weights 来让模型收敛到更佳的位置。

VGG 模型是在 LSVRC-14 提出的一系列模型，其中 16 层模型和 19 层模型在 ImageNet 上表现最好，这两个模型在后来就被称作 VGG16 和 VGG19。VGG 模型的结构很简单，整个模型由几组相同 filter 的卷积层叠加而成，filter 大小均是 3x3，每一组卷积层后面叠加一个相同的 MaxPooling 层。[3]

基于 VGG16 模型简洁和直观性，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 来对数据集进行验证，如果 VGG 模型能得到不错的效果，可以再考虑对模型进行优化或使用层数更多的模型。

在这个实现方案中，我们使用了 Keras 自带的 VGG16 模型，weights 使用 'imagenet' 预训练 weights，我们将原本 softmax-1000 classifier 去除，用 softmax-10 来替代。

训练阶段，首先锁住预训练模型的所有层，只保留新加的 softmax-10 的全连接层，这样能使得 softmax-10 的 weights 能收敛到最佳位置附近。如果不锁住预训练模型的 layer，由于新加的全连接层的 weights 是随机初始化的，该层的 gradients 会非常大，这样会使得预训练的层的 weights 有很大的更新幅度。但本来预训练层的 weights 就在最佳位置附近，如果有较大的更新，会使得偏离最佳位置，也就是说新加的随机初始化的全连接层会影响所有的预训练层，从而使得所有层需要重新训练。

所以我们需要先锁住预训练层，微调新加的全连接层，等到全连接层趋近收敛，在微调我们感兴趣的层。

### 使用 Pretrained InceptionV3

GoogLeNet 模型是 ILSVRC 2014 的冠军，不同于 VGG 模型，GoogLeNet 打破了以往串行叠加卷积层的策略。对于一个卷积层，使用不同大小的 filter 能得到不同的特性，选择什么样的 filter 很多是基于经验和实验结果来判断的。GoogLeNet 创新性地并行使用了多个不同的 filter，最后综合所有 filter 得到最后输出，该模块也被称为 `Inception Module`，如图 4 所示。

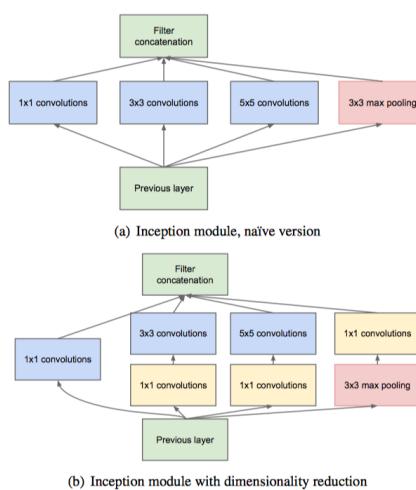


Figure 2: Inception module

图 4: Inception module [4]

如图 4 所示，module 中并行使用了不同的 filters，每一种 filter 都可能提取数据的不同特征信息，例如  $1 \times 1$  conv 可能提取数据的细微特征， $5 \times 5$  conv 有更大的感受野，可能提取其他的特征。另外，每一组 filters 之前都有一层 NiN  $1 \times 1$  convs，其目的是减少输入给尺寸较大的 convs 的维度，有效减少模型训练的计算量。

在这个实现方案中，我们使用了 Keras 自带的 InceptionV3 模型，该模型也被称为 GoogLeNet 的第三代。模型 weight 使用 'imagenet' 预训练 weights，与训练 VGG16 模型类似，我们用 softmax-10 来替代原模型的最后一层 softmax-1000。

该模型的训练阶段与 VGG16 也类似，首先锁住原模型的所有层，只训练新加的 softmax 层，等该层收敛一定程度后，再微调其他所有层。

### 使用 Pretrained ResNet50

除了 VGG16 和 InceptionV3 模型，本项目还使用了 ResNet 模型对图片进行了训练。

ResNet 赢得了 ILSVRC 2015 竞赛，其测试结果达到了 3.57% Top-5 error，历史上第一次超过了 5% Top-5 error。与 InceptionV3 模型一样，ResNet 同样具有非常深的网络结构。ResNet 另一个非常显著的特征是使用了 Residual Block，如图 5 所示。

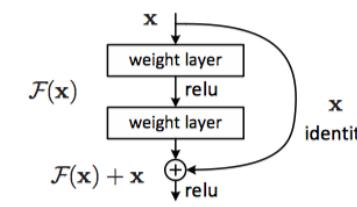


Figure 2. Residual learning: a building block.

图 5：Residual module [5]

之前的 CNNs 模型的每一层的输入都是上一层的输出，而 ResNet 的 Residual Block 第一次引入了「跨层」的概念，即某一层的输入不仅来自上一层的输出，还可能叠加了之前层的输入。如上图所示，某卷积层的  $F(x)$  叠加了之前的输入  $x$ ，叠加起来  $H(x) = F(x) + x$  作为下一层的输入。

比较 Redisual Block 和一般的卷积模块，一般的卷积模块每次将输入 map 到一个全新的输出，而 Residual Block 的输出则是对原有输入的一个微小改动。ResNet 作者假设「优化残差比优化直接输出更容易」[5]。

与之前模型的实现方案类似，我们使用了 Keras 自带的 ResNet50 模型，weight 使用 'imagenet' 预训练 weights，与训练 VGG16 模型类似，我们用 softmax-10 来替代原模型的最后一层 softmax-1000。

同样，在模型的训练中，首先锁住原模型的所有层，只训练新加的 softmax 层，等该层收敛一定程度后，再微调其他所有层。

备注：以上三种预训练模型的 hyper-parameter 是：

- fine-tune new top layer epoch 5
- Epoch 15
- KFold 5
- batch\_size 64
- random\_state 4

### 使用 CAM 技术可视化训练好的 CNNs 模型

通常而言，图片分类的任务往往也需要物体定位（localization），利用物体定位我们可以看出我们的分类模型是否能正确检测到目标分类的图像。本项目我们将使用 Bolei Zhou 提出的 Class Activation Mapping [10] 来对训练的模型进行物体定位。

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，通过将对目标分类敏感的 filters 按权重累加，即可得到目标分类的映射 mapping。

CAM 具体实现大致是，将深层的某卷积层的输出，与最后某目标分类的权重做数量积，再映射回 image space，这样就得到目标分类的可视化图像。该图像颜色深的部位就代表该目标分类的定位。如下图 6 所示。

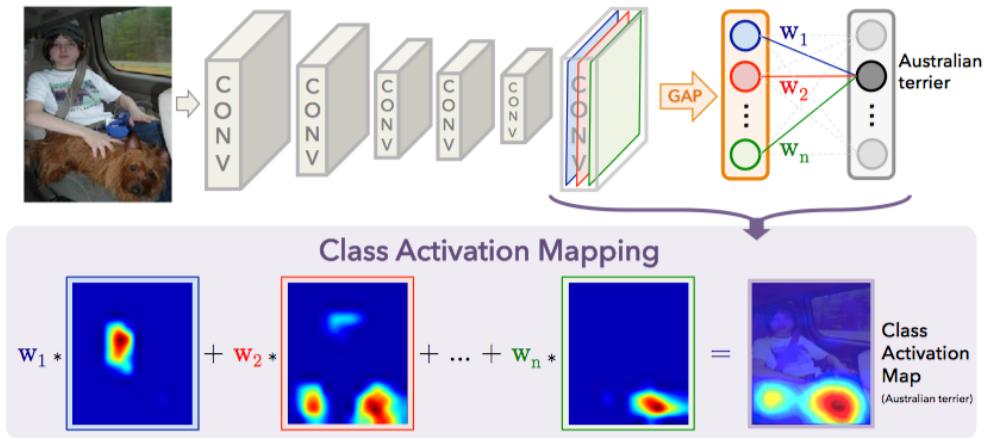


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

图 6：Class Activation Mapping 算法步骤 [10]

本项目利用 Keras 的 Tensorflow API 将训练好的 VGG16, InceptionV3 和 ResNet50 模型，修改成 VGG16-CAM, InceptionV3-CAM 和 ResNet50-CAM 后，再进行训练，最后得到三种模型的 CAM 可视化结果。

## 优化

在实现和训练简单两层 CNN 模型的过程中，发现了很多问题：

- 预测结果并不是很好。
- 该模型并没有防止 overfitting 的措施。
- 训练速度较慢。
- 模型训练不充分。

根据这些问题本项目主要采用了如下的优化方案：

- 对 driver id 使用 cross-validation。
- 训练多个模型参数，综合预测结果。
- 使用 Adadelta optimizer 来加快训练收敛速度。
- 增加训练迭代次数。
- 使用 Early stopping 来加快训练速度并防止 overfitting。

在确定这些优化方案之后，VGG16、InceptionV3、ResNet50、CAM 模型的训练全部都使用这些优化方案。这样的好处是，由于预训练模型的 parameters 都非常庞大，所以如果针对不同的优化方案去训练预训练模型的话，太耗时间了，所以一个取巧的办法是确定好优化方案后，直接将这些优化方案实施到预训练模型上。当然，如果有不确定的优化方案，还是应该在预训练模型上进行训练后，在判断该优化方案是否可行。

### Cross-validation

对于机器学习算法来说，其本身有很多 hypter-parameters 需要验证和调整，其目的是降低对现有数据的过度依赖，未来能泛化（generalize）到未见过的数据 [11]。目前一种常用的验证方法是 KFold cross-validation，我们将采用该方法对训练集进行切分，并对训练模型进行验证。

#### 训练多个模型参数，综合预测结果

我们数据集的分布有可能并不是随机的，如果只对其进行一次切分，训练集和验证集的分布有可能相差很大，训练出的模型并没有学习到数据分布的全貌。所以本项目的交叉验证将训练集分成多个组合，根据不同组合的训练集训练出不同的模型参数，最后预测会综合所有的模型的结果。

另外，本项目使用的 KFold 并不是简单的对所有图片进行切分，而是对不同汽车/司机 id 进行切分，并训练其相应的图片。这样做的原因是训练数据并不是很大，我们害怕模型学到了其他特征而忽略了真正的特征。这也是很多小数据集出现严重过拟合的原因。例如本项目中我们希望模型学到的是检测图片当中的司机行为，而不是司机的衣着或是汽车内装饰灯。并且从 driver id 列表可知，训练集一共有 26 个不同的司机，但我们无法知道测试集有多少个司机（这也与现实世界的假设一致，因为算法不可能一直应用于模型训练过的司机）。为了降低司机或汽车的属性影响到训练效果，我们有必要排除模型对相同司机的依赖。即模型训练和验证使用完全不同的司机/汽车 id 图片。

具体做法是，首先根据司机的 id 来将训练图片进行分类，然后对司机 id index 进行 KFold 处理，这样得到的 train set 和 validation set 分别是不同的司机 id 对应的图片，例如 train set 的司机 id 有 [ p015, p022, p051 ]，validation set 的司机 id 有 [ p002, p081 ]。对这些不同组合的训练集进行训练，得到不同的模型参数。测试阶段将这些不同的模

型分别对测试集进行预测，得到多个测试结果，最后将其进行合并，这样往往能得到更好的综合结果。

### **Adadelta Optimizer**

为了加快模型的训练速度，在编译阶段，所有模型的 optimizer 都使用 `adadelta` [9]，loss function 使用 categorical cross entropy，metrics 使用 accuracy。

### **增加 Epoch**

从 learning curve 中可看出，val\_acc 仍然是下降趋势，说明我们的模型还没有达到最优位置，在这种情况下一种很常见的方法即是增加迭代次数。

### **Early Stopping**

在使用深度 CNN 时，往往需要很多 epoch 模型才能收敛。但需要多少 epoch 模型才收敛，这是一个经验问题，如果 epoch 过少则训练不完备，如果 epoch 过多则训练时间过长，并且可能会导致 overfitting。

这里我们使用 Early stopping 的方法来解决这个经验问题。具体而言，我们使用两种 early stopping 方法，一、设定 `val\_loss` 为 `1e-5`，如果验证集的损失小于这个值时则训练停止；二、监控 `val\_loss`，如果验证集损失有 5 次迭代都没有提高，则停止训练。

## **4. 结果**

### **模型评价与验证**

根据上一章的实现，我们依次得到下面几个测试结果。

## 2 层 CNN

自建的 CNN 在训练集中得到的结果如图 7 所示。最后的 Private Leaderboard 得分为: 3.52051。

比较训练集的测试结果和 leaderboard 得分，我们发现该模型在测试集上的表现并不好，从这点看可以判断出模型可能是对训练集过拟合了。

## 使用优化方案后的 2 层 CNN

使用优化方案后的自建的 CNN 在训练集中得到的结果如图 8。最后的 Private Leaderboard 得分为 2.54849。

优化后，模型虽然表现更好，但 Kaggle 上得分却并不是很理想。主要是因为我们的模型结构相对简单，没有能够学习到足够多的数据集重要特征。

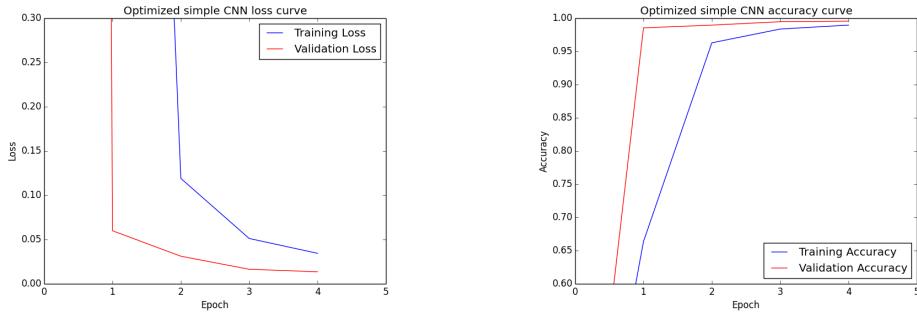


图 7: 2 层简单 CNN 模型训练结果

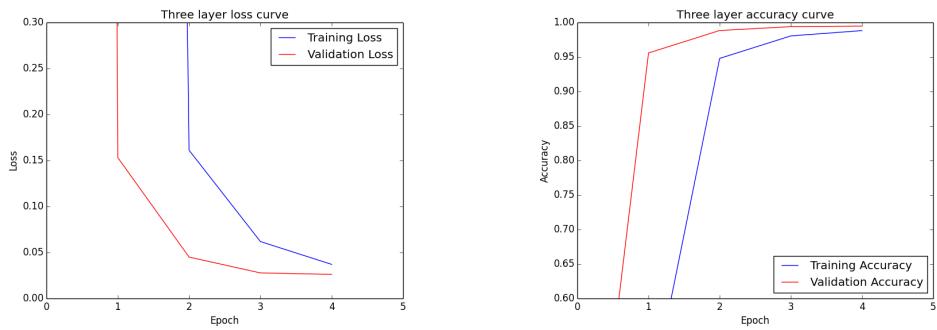


图 8: 优化后的 2 层简单 CNN 训练结果

## Pretrained VGG16

Fine-tune VGG16 最后得到的训练集结果如图 9 所示。上传该模型的测试结果，得到 private score 为 1.49448，kaggle 排名 655/1440。

比较以上结果和之前的自建模型，可以看出，利用模型的深度增大，得到非常显著的效果。也说明预训练模型进行迁移学习是非常有效的。

## Pretrained InceptionV3

该 Fine-tune InceptionV3 得到的结果如图 10 所示。上传该模型的测试结果，得到 private score 为 0.93604，排名 519/1440。

比较 VGG16 模型的测试结果，InceptionV3 模型得到了更好的分类效果。同样也说明了卷积层数越多，得到的效果越好。

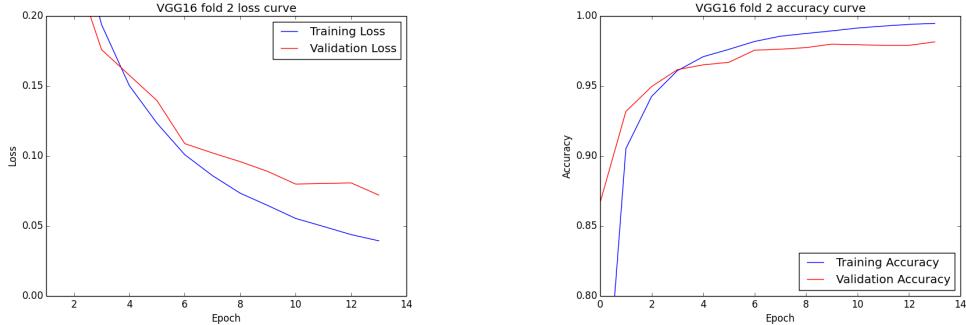


图 9：Pretrained VGG16 模型训练结果

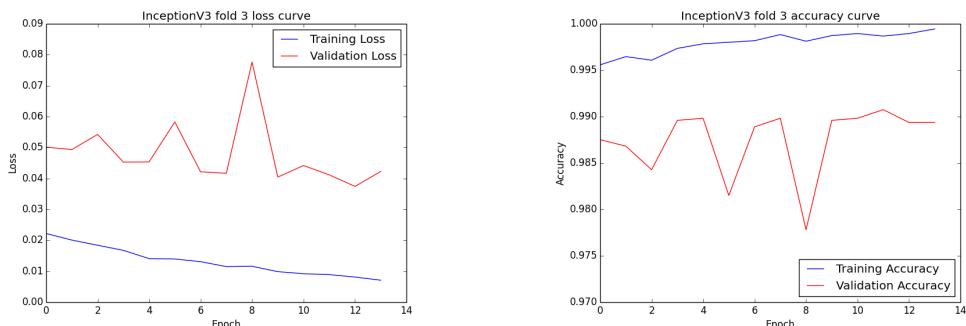


图 10：Pretrained InceptionV3 模型训练结果

## Merge VGG16, InceptionV3

通过比较以上两个模型的最后预测结果，我们发现对于有的图片 VGG16 能准确预测，但 InceptionV3 却预测失败；而另外一些图片 VGG16 得到错误预测，InceptionV3 却得到了正确预测。另外那些预测失败的图片，往往是模型对该图片的结果「拿不定主意」，也就是说该模型的预测结果的概率集中在两个或三个 label 上。该模型对某些图片「拿不定主意」，也可以说该模型对这些图片训练不完备。但另一个模型对该图片有可能训练完备。从以上分析可得出，如果我们综合不同模型的预测结果，可能会得到更好的预测结果。

我们综合了 VGG16 和 InceptionV3 的预测结果，得到的 private score 为 0.67249，排名 400/1440。

从 log loss 的表现结果来看，综合两个效果近似的模型往往能达到更好的表现。

## Pretrained ResNet50

Fine-tune ResNet50 得到的结果如图 11 所示。上传该模型的测试结果，得到 private score 为 0.20686，排名 70/1440。

从得分上来看，该模型在本项目中表现最好，得分已经达到了 Leaderboard 的 Top 5%，达到了本项目的预期目标。

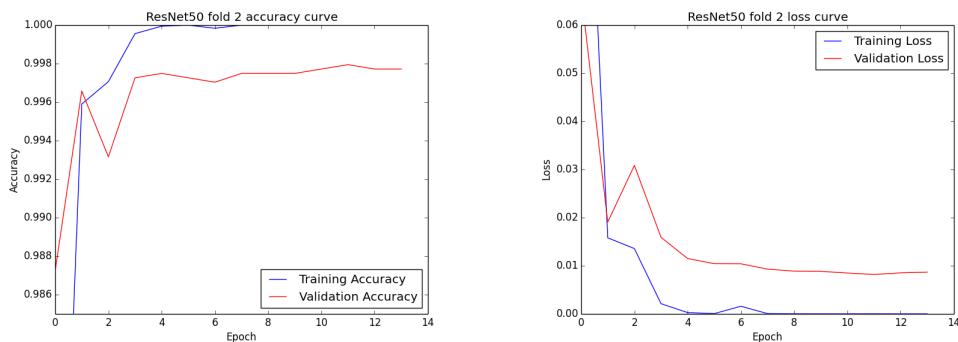


图 11：pretrained RestNet50 模型训练结果

图 12 为三个预训练模型在本项目的训练集上的结果对比。

表格 I 是所有训练模型在 Kaggle Leaderboard 上的表现。

表格 II 是所有模型在一个 Epoch 上的训练和一个图片的预测时间，测试机器为 AWS EC2 p2.xlarge。从训练时间上来看 ResNet50 虽然表现最好，可训练时间比其他预训练模型长很多。不过大多实际项目中使用的是训练好的模型，从表 II 中可知 ResNet50 模型的预测时间和其他预训练模型近似。如果将模型运用到实际项目当中，需要看实际应用对训练、预测时间的要求，这就需要在「效果 - 时间」上做 trade-off 了。

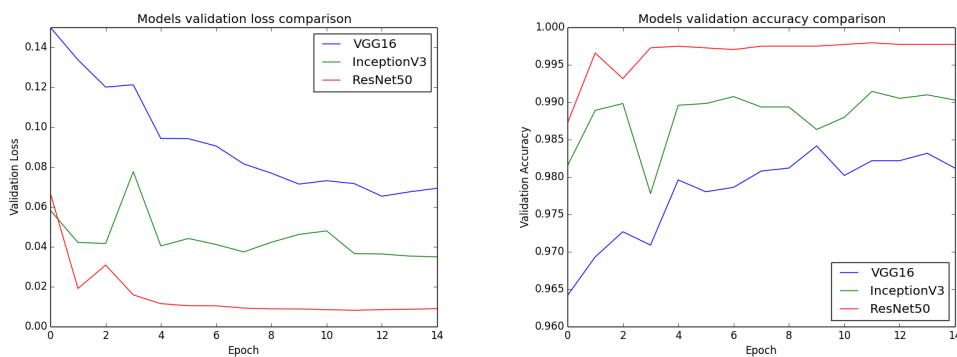


图 12：三个预训练模型在训练集上的表现结果对比

训练模型	Private Score	Public Score
2 层 CNN	3.52051	3.10518
优化后的 2 层 CNN	2.54849	2.25109
VGG16	1.49448	1.52322
InceptionV3	0.94853	0.93604
VGG16 & InceptionV3	0.67249	0.73494
InceptionV3-CAM	1.40722	1.32374
ResNet50	<b>0.20686</b>	<b>0.18362</b>
ResNet50-CAM	0.21638	0.23052
Merge VGG16 & InceptionV3 & ResNet50	0.33984	0.35276

表格 I：所有训练模型在 Kaggle Leaderboard 上的表现

模型	One epoch training time	One pic prediction time
2 层 CNN	31s	3e-4s
优化后的 2 层 CNN	25s	3e-4s
VGG16	330s	0.015s
InceptionV3	540s	0.011s
ResNet50	620s	<b>0.01s</b>

表格 II: 所有模型的训练时间和预测时间

## 结果分析

根据以上预训练模型的效果，本项目对训练后的 InceptionV3 和 ResNet50 模型进行了 CAM 模型训练，检查模型是否正确检测到了目标分类的物体。

InceptionV3-CAM 在 Kaggle private leaderboard 得分为 1.40722。ResNet50-CAM 在 Kaggle private leaderboard 得分为 0.36598。

从训练结果上来看，ResNet50 模型表现依然最优。以下我们将使用 ResNet50-CAM 来进行可视化检查。

我们进行可视化检查的目的主要是查看训练的模型是否正确输出了目标分类的物体。首先检查那些预测正确的图片，图 13 是可视化的结果。

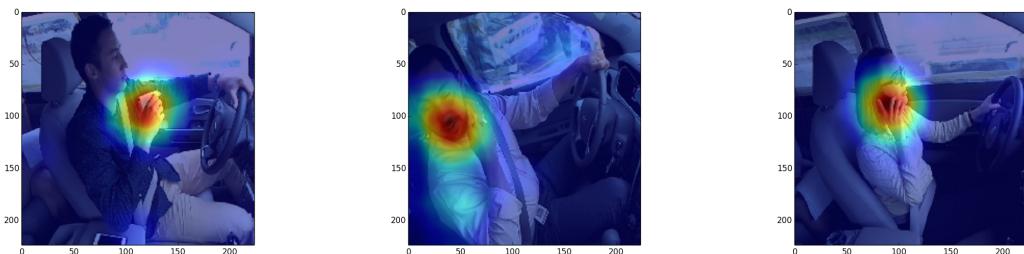


图 13: CAM 可视化结果

从图 13 的几个 CAM 可视化图片来看，那些预测正确的结果同样能检测出正确分类的目标物体。

接下来我们要检查那些预测错误的图片的可视化结果，该图片的原图见图 14。



图 14：模型预测错误的原图片

该图片的预测分类是 c9（和其他乘客说话），而正确分类是 c4（左手打电话）。从图片来看，图片内容相当有迷惑性，首先司机的头是转向右侧的，很像是与其他乘客说话的姿势，另外左手臂全部被身体挡住，只有手显示在了图片上，这也增加了迷惑性。

图 15 依次是可视化实际分类 c4 和可视化预测错误分类 c9。可以发现虽然正确分类 c4 的主要物体「拿电话的手部」被检测出来了，但 c9 的可视化同样检测到了「司机脸部」，从结果来看，c9 的权重更高，所以模型预测错误。

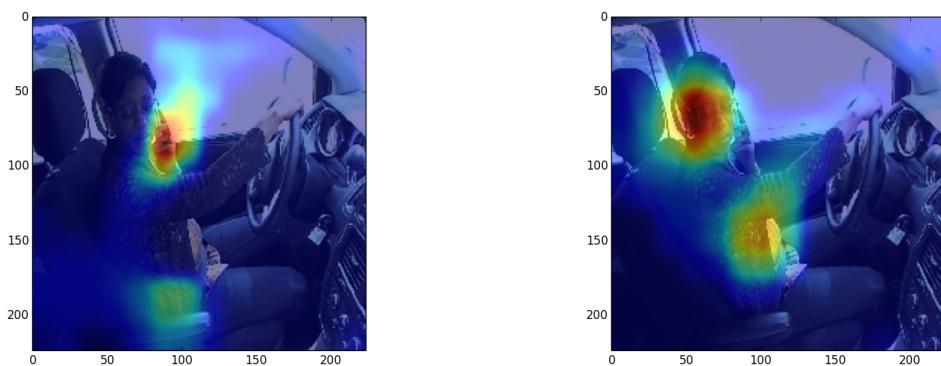


图 15：CAM 可视化结果

再来分析另外一张预测错误的图片，原图见图 16。



图 16：模型预测错误的原图片

该图片的预测分类是 c9（和其他乘客说话），而正确分类是 c0（安全驾驶）。同样图片内容有一定的迷惑性，司机的头部并不是面向正前方，而是转头向了左侧。

图 17 依次是可视化实际分类 c0 和可视化预测错误分类 c9。可以发现正确分类 c0 的主要检测目标「双手握住方向盘」被检测出来了，但错误预测的 c9 却检测到了人的颈部位置，这是模型本不应该发生的错误。一种造成这种错误分类的原因是，模型可能过拟合了，记忆了训练数据的一些额外的、不重要的特征。

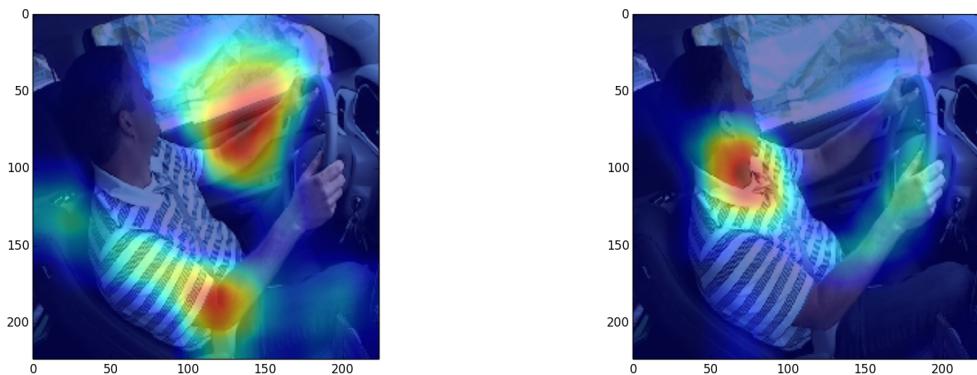


图 17：CAM 可视化结果

表格 III 是上述两个预测错误图片的预测概率表，其中加粗字体是实际分类的概率和预测分类的概率。从表中可以明显看出实际正确分类的概率也相当高，说明模型对这样的图片并不是很确定。实际上从最后的结果来看，几乎所有预测错误的概率都在 0.5 附近，而实际正确的概率与预测错误的概率相差不大，从这点可以看出模型还有很大的提升空间。

预测图片	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
img_11754	0.1008	0.0109	0.0888	0.0039	<b>0.2020</b>	0.0139	0.0158	0.200	0.0042	<b>0.5395</b>
img_11507	<b>0.4661</b>	2.559E-05	7.983E-06	4.567E-05	5.881E-05	0.0007	2.298E-06	4.639E-05	0.0001	<b>0.5329</b>

表格 III 预测错误图片的概率表

## 5. 结论

### 总结

本项目首先使用了简单的 2 层 CNN 来对数据进行测试，然后确定了优化方案，之后对三种 pretrained models 进行了训练，最后通过比较和合并结果，选出最优的解决方案。

本项目最后能取得 Kaggle Leaderboard Top 5% 的结果，主要得益于两点，首先充分利用了 Transfer learning 的优势，使用 pretrained models 来对训练进行热启动，极大加快了训练的速度，并且避免了深度学习所必须的海量的数据量。其次，谨慎对训练集进行了分割，即按司机/汽车 id 进行分割，这样有效避免模型学习到我们不期望它学到的特征。

### 后续改进

由于本项目的数据集比较大，训练 pretrained models 所需的计算资源很高，训练时间很长，所以本项目 fine-tune pretrained layers 只用了 15 epoch 来训练。从 Learning curve 上可以看出，有的 fold 结果在最后一个 epoch 仍呈现下降趋势，所以如果再增加 epoch 可能会得到更好的效果。

同样受限于训练时间，本项目使用了 5 folds，最后综合 5 个 fold 的预测结果，得到最终的结果。从 CAM 可视化分析上来看，很多预测错误的概率都在 0.5 左右，如果增加 folds 为 10，将会得到更多的模型，最后得到的平均得分可能会更稳定。图 18 是整个训练所用 p2.xlarge 实例的时间。

另外本项目没有使用 data augmentation，在分析训练图像和可视化预测结果的时候可以发现，影响目标分类的主要图像区域集中在司机的头部，以及司机的手部，如果能将图片按照司机的头和手进行分割，将分割后的图片作为训练集的一部分，可能会得到更好的结果。

图 19 是本项目提到的所有训练模型在 Kaggle 上的评分。

p2.xlarge Linux/UNIX Spot Instance-hour in US East (Virginia) in VPC Zone #12	60 Hrs
p2.xlarge Linux/UNIX Spot Instance-hour in US East (Virginia) in VPC Zone #7	74 Hrs

图 18: p2.xlarge 实例训练时间

<a href="#">submission_3layers_cnn.csv</a> 3 hours ago by <a href="#">ijinmao</a> Create 3 Conv layer simple CNN model.	3.52051	3.10518	<input type="checkbox"/>
<a href="#">submission_optimized_3layers_cnn.csv</a> a minute ago by <a href="#">ijinmao</a>	2.54849	2.25109	<input type="checkbox"/>
<a href="#">submission_ResNet50.csv</a> 5 hours ago by <a href="#">ijinmao</a> ResNet50 folds 5, epoch 15	0.20686	0.18362	<input type="checkbox"/>
<a href="#">submission_inceptionV3.csv</a> 5 hours ago by <a href="#">ijinmao</a> InceptionV3 folds 5, epoch 15	0.93604	0.94853	<input type="checkbox"/>
<a href="#">submission_VGG16.csv</a> 5 hours ago by <a href="#">ijinmao</a> VGG16 folds 5, epoch 15.	1.49448	1.52322	<input type="checkbox"/>
<a href="#">merge_ResNet_CAM.csv</a> 4 days ago by <a href="#">ijinmao</a> Merge ResNet50 and ResNet50-CAM.	0.21638	0.23052	<input type="checkbox"/>

图 19: 本项目的训练模型在 Kaggle 上的评分

## 6. 参考文献

- [1] Convolutional Neural Networks: Architectures, Convolution / Pooling Layers, <http://cs231n.github.io/convolutional-networks/>
- [2] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.
- [3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
- [4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015.
- [5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.
- [6] Pan SJ, Yang Q. A survey on transfer learning. IEEE Trans Knowl Data Eng. 2010;22(10):1345–59.
- [7] Transfer Learning and Fine-tuning Convolutional Neural Networks, <http://cs231n.github.io/transfer-learning/>
- [8] [https://en.wikipedia.org/wiki/Curse\\_of\\_dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality)
- [9] An overview of gradient descent optimization algorithms, <http://sebastianruder.com/optimizing-gradient-descent/>
- [10] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In In Advances in Neural Information Processing Systems, 2014.
- [11] [https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))