机器学习纳米学位

毕业项目 检测走神司机

柴森

2018 年 5 月

目 录

1. 问题的定义

1.1.项目概述

随着道路交通的发展及私家车辆的普及,道路交通事故已经成为了人们关注的社会问题。这些交通事故大部分都或多或少与司机的驾驶行为相关，例如疲劳驾驶、注意力分散等。根据美国国家公路交通安全交通安全署的统计，在美国的公路上，每年由于司机驾驶过程中进入睡眠状态而导致大约 10 万起交通事故，其中约有 1500 起直接导致死亡,7 万起事故导致人员受伤。根据 CDC 机动车安全部门的统计，每年由于司机开车时精力不集中造成的人员受伤和死亡人数分别为 425,000 人和 3,000 人。

司机的不良驾驶行为已经对自身和乘客的生命造安全造成了及其严重的不良影响，解决该问题具有很强的必要性和紧迫性。

1.2.问题陈述

该项目是一个计算机视觉多分类问题，基于大量车内司机的 2D 图像，需要通过使用卷积神经网络构建模型实现对每张图片中的司机行为进行分类，判定他们是否在专心驾驶，是否系好安全带，是否在驾驶过程中使用了手机，针对每一张图像算法最终输出车内司机可能性最高的行为类别。

1.3.评价指标

kaggle 提高的 logloss

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

N 是测试集图片的数量，M 是图像类别标签，log 是自然对数，在图像 i 属于类别 j 时 y_{ij}=1，否则 y_{ij}=0，p_{ij}是模型预测的图像 i 数据类别 j 的概率

2. 分析

2.1.数据的探索

该项目数据由 State Farm 公司提供，提供的数据包括摄像头采在车内集到的司机 2D 图像和每张图像对应的类别标签。图像的类别标签包括以下 10 种：

- 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：操作收音机
- c6：喝饮料
- c7：向后方看

- c8：整理头发和化妆
- c9：与乘客交流

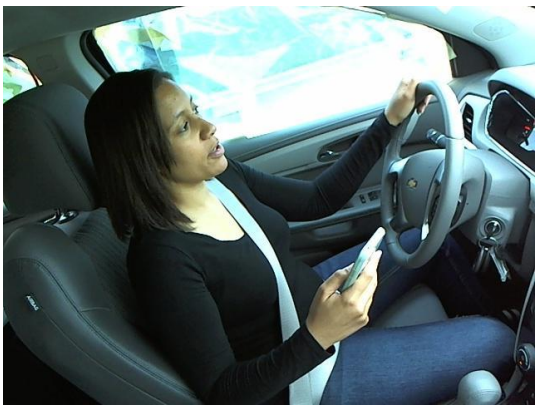
以上数据可以在 https://www.kaggle.com/c/state-farm-distracted-driver-detection/data 上进行下载，得到的三个文件为：

- imgs.zip：所有图像（训练/测试）文件的压缩包文件
- sample_submission.csv：正确地提交格式样本
- driver_imgs_list.csv 训练图像列表，司机 id 和类别 id

imgs.zip 文件大小为 4.29GB，解压后得到 train 和 test 两个文件夹，train 文件夹里的数据为训练集数据，共包含 22424 张图像，每张图像的大小为 640*480；test 文件夹里的数据为测试集数据，共包含 79726 张图像，每张图像大小为 640*480。由此可以看出测试集的数据量要多于训练集的数据量。

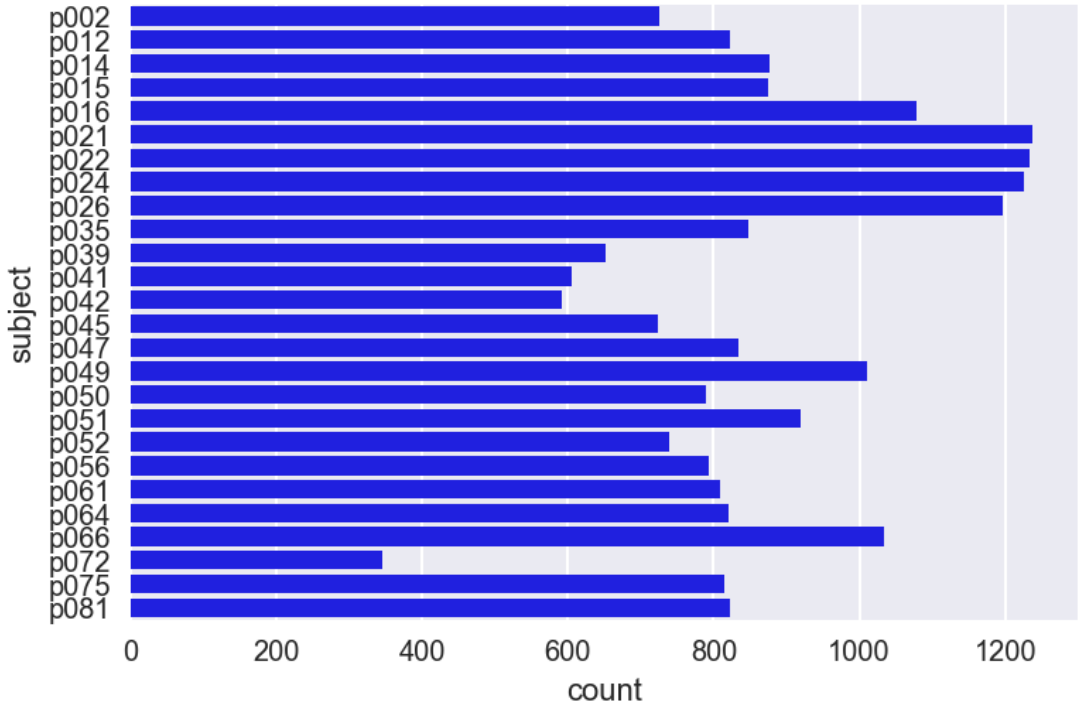
2.2. 探索性可视化

该问题的输入数据为下图所示的图像。

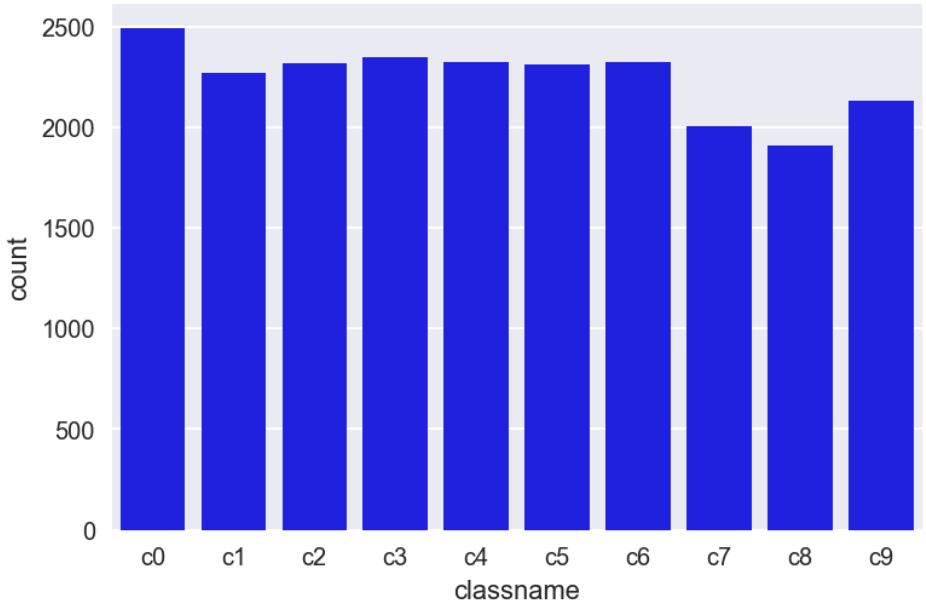


通过 driver_imgs_list.csv 文件分析得到训练集中共有 26 位司机，每位司机的图像总数有所不同，P021、P022、P024 的图像数较

多超过 1200 张；而 P072 只拥有不到 400 张图像。从下图可以看出，每位司机的图像数量总体来说是平衡的。



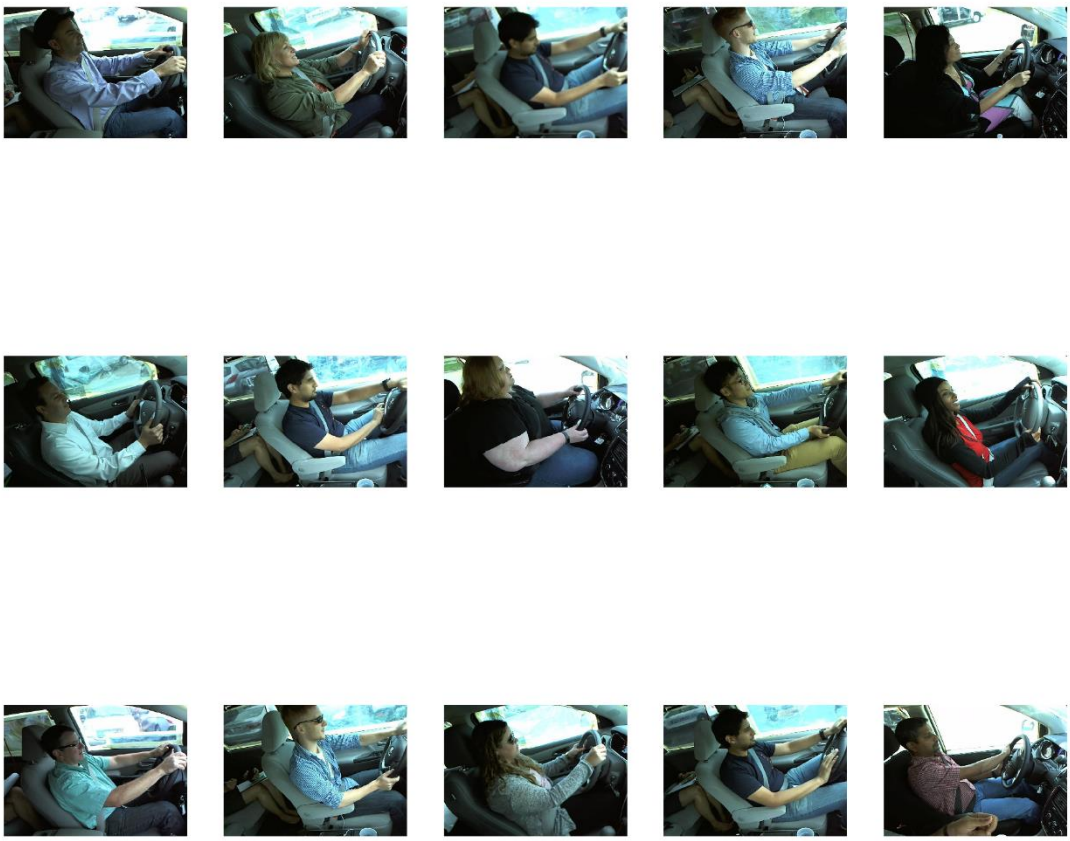
通过下面的直方图可以看出每种类别的图像个数相差不大，说明整个数据集在每个类别上是平衡的。



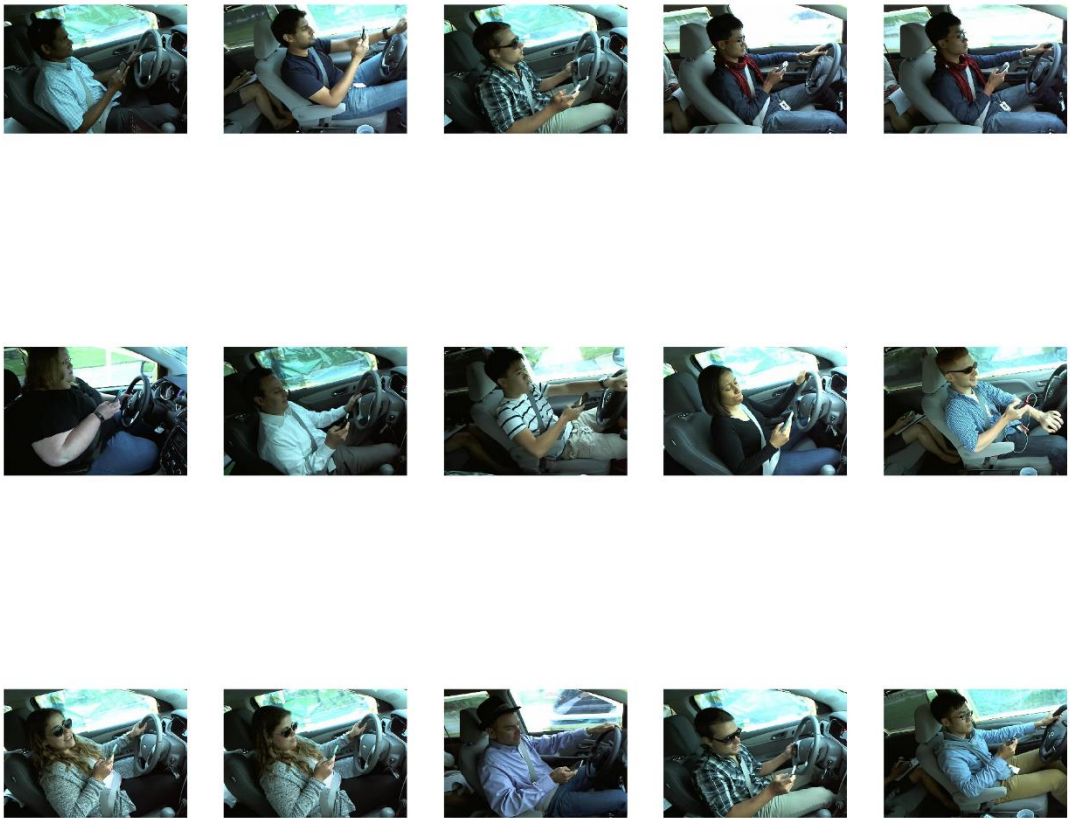
下图展示的是每个司机在每个类别上的图像数量，可以看出每

个司机在 c0-c9 中都有图像，除 P072 号司机数据量稍少，总体上每个司机在每个类别上的数据数量是平衡的。

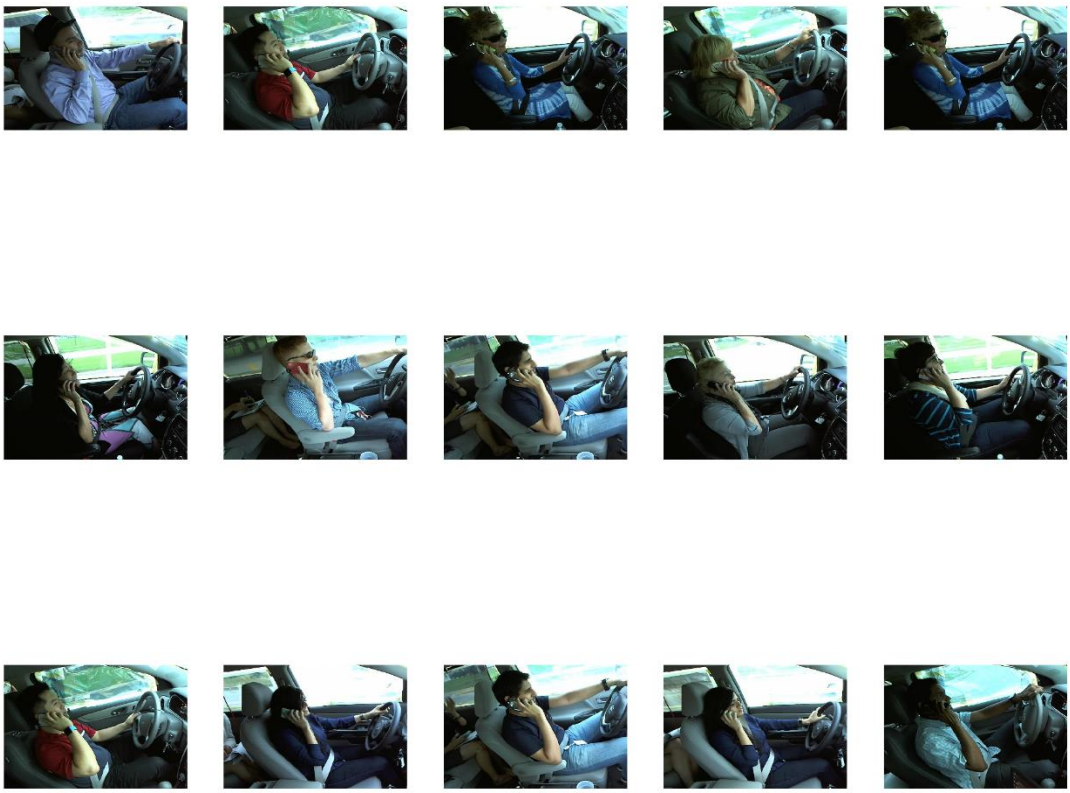
c0：安全驾驶部分图像如下：



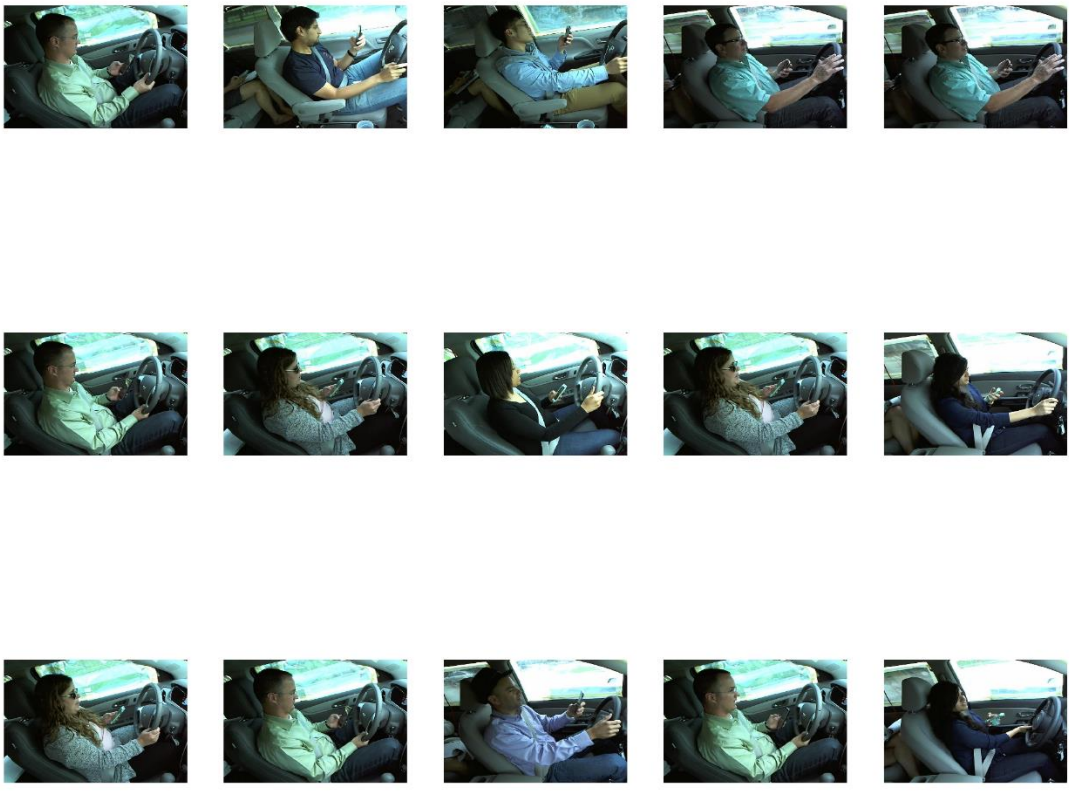
c1：右手发短信类部分图像如下：



c2：右手打电话

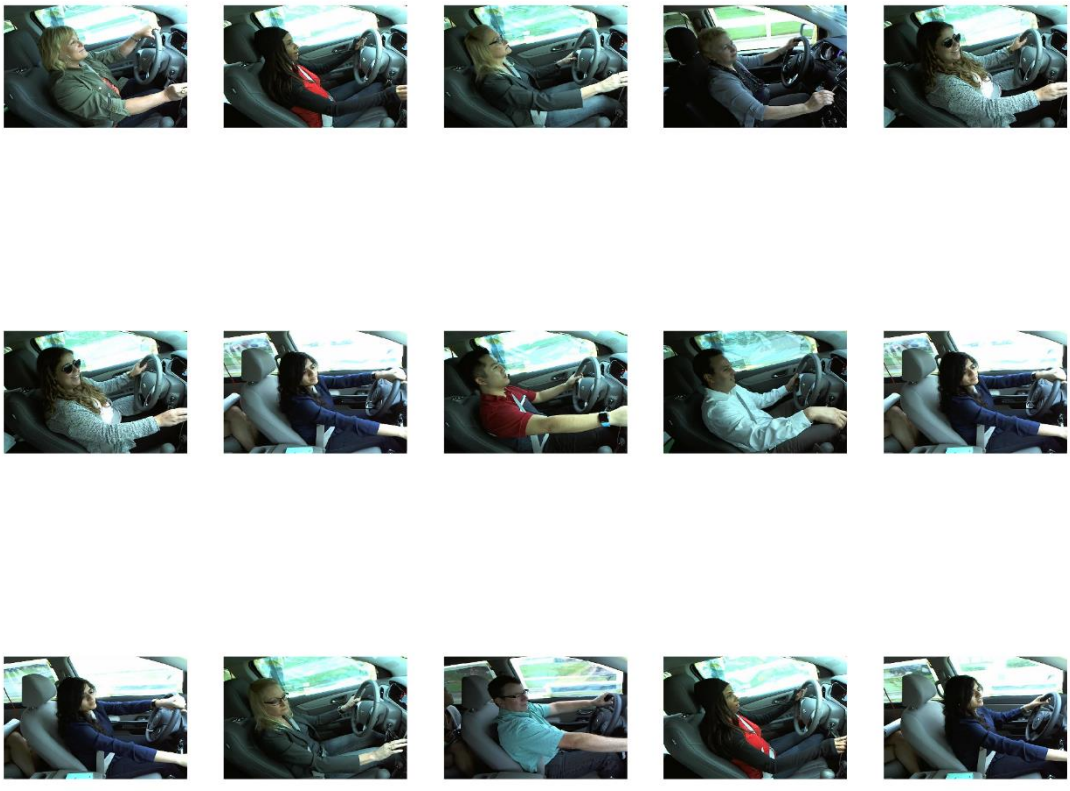


c3：左手发短信

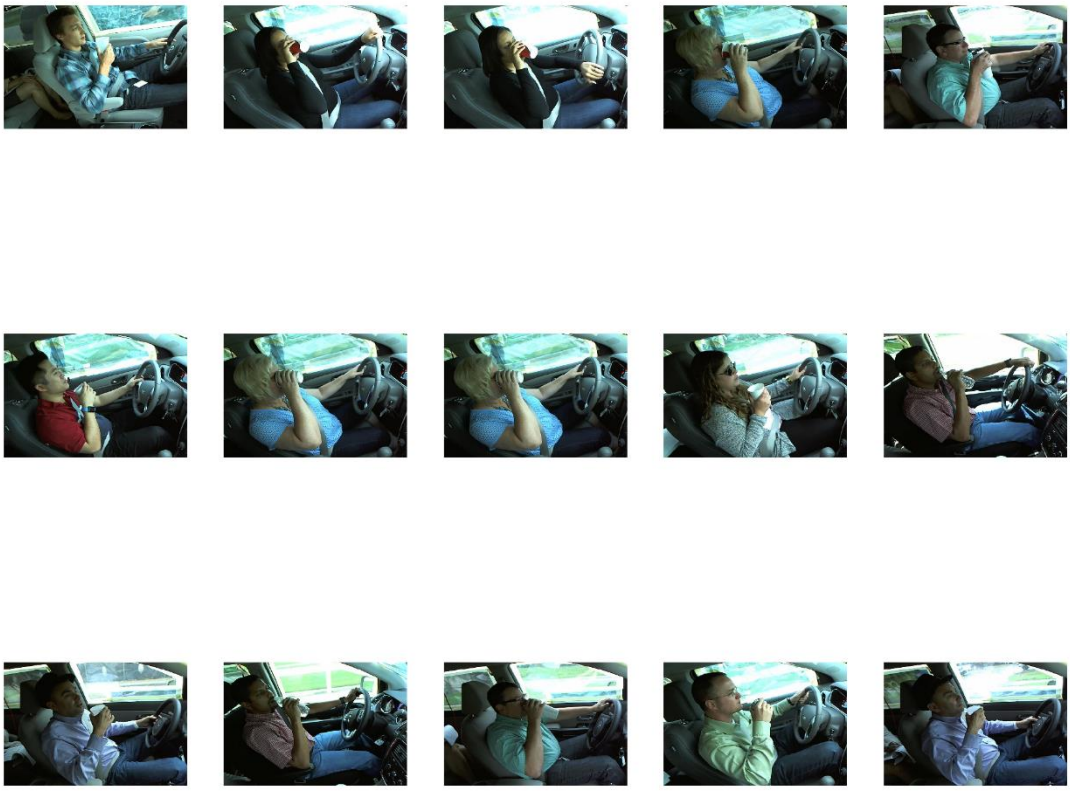


c4：左手打电话

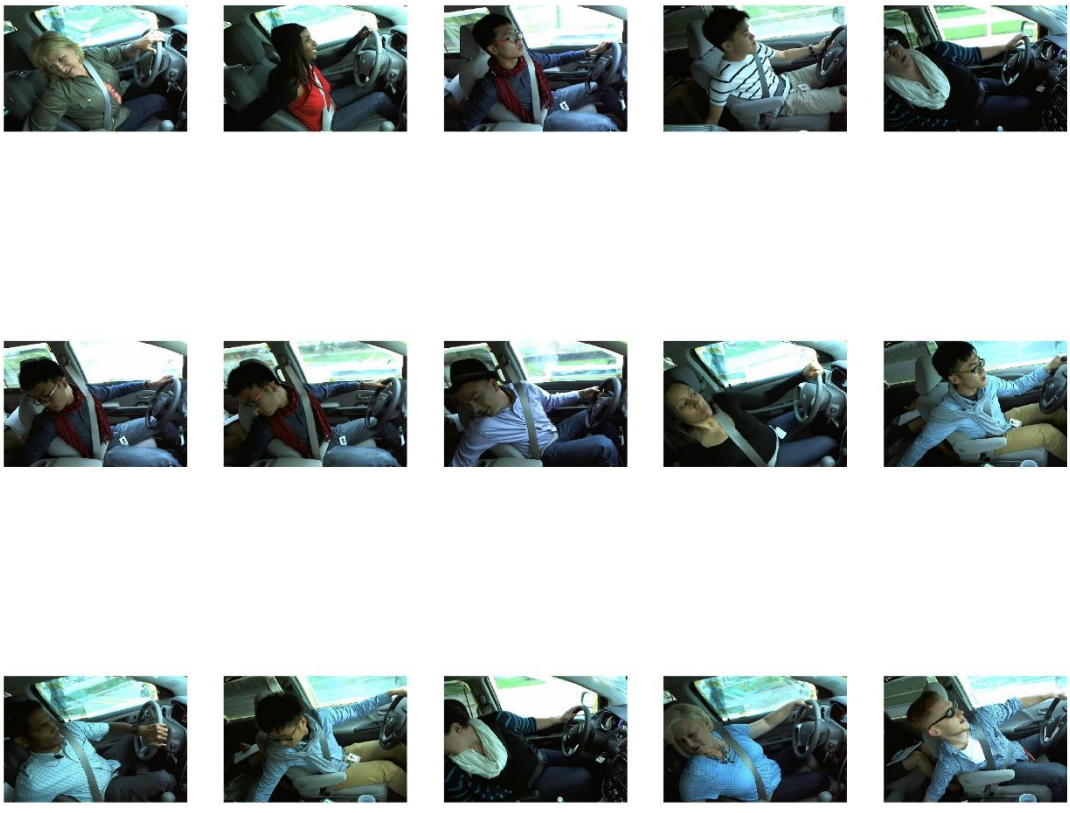
c5：操作收音机



c6：喝饮料

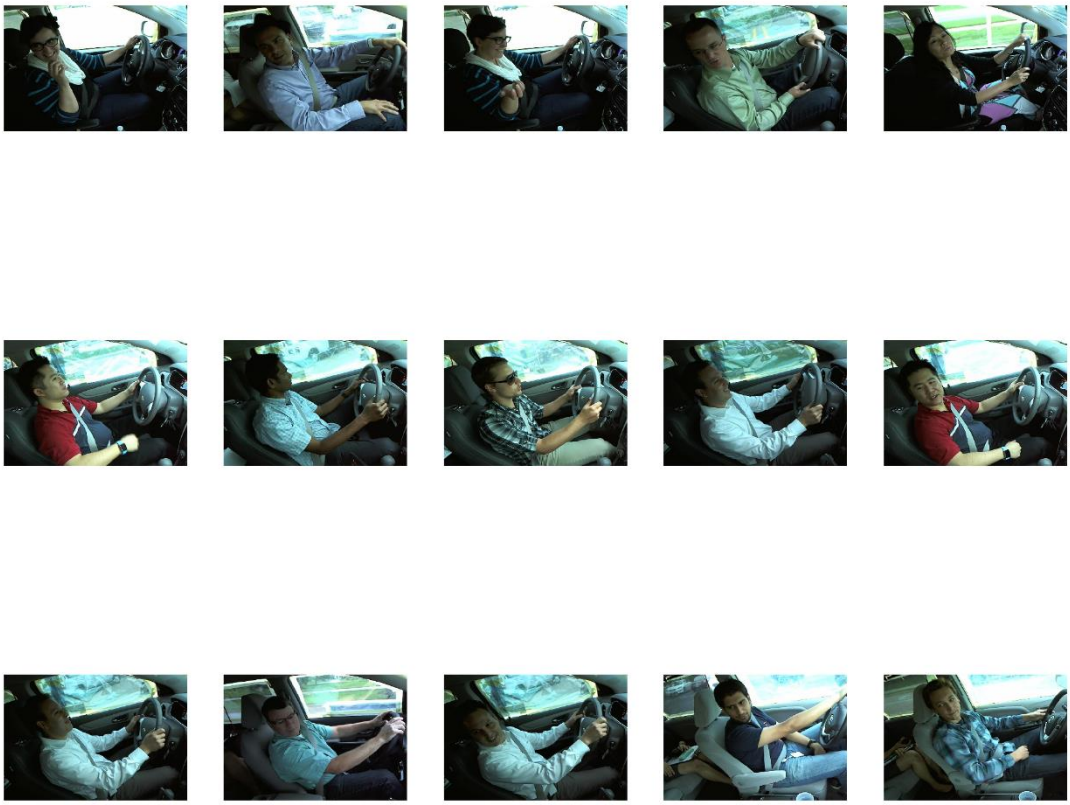


c7：向后方看



c8：整理头发和化妆

c9：与乘客交流



2.3.算法和技术

2.3.1.　卷积神经网络

1）VGG16

VGG 模型输入图像大小为 224×224 的 RGB 图像，模型整体使用的卷积核都比较小（3×3），3×3 是可以表示左右、上下、中心这些模式的最小单元。还有比较特殊的 1×1 的卷积核，这种卷积核可看做是空间的线性映射。VGG 模型前面几层是卷积层，后面几层是全连接层，最后是进行分类的 softmax 层，所有隐层的激活单元都是 ReLU 单元。

使用多个较小卷积核的卷积层代替一个卷积核交大的卷积层，一方面可以减少参数，另一方面是模型进行了更多的非线性映射，可以增加网络的拟合或表达能力。下图是 VGG16 模型的整体结构图：

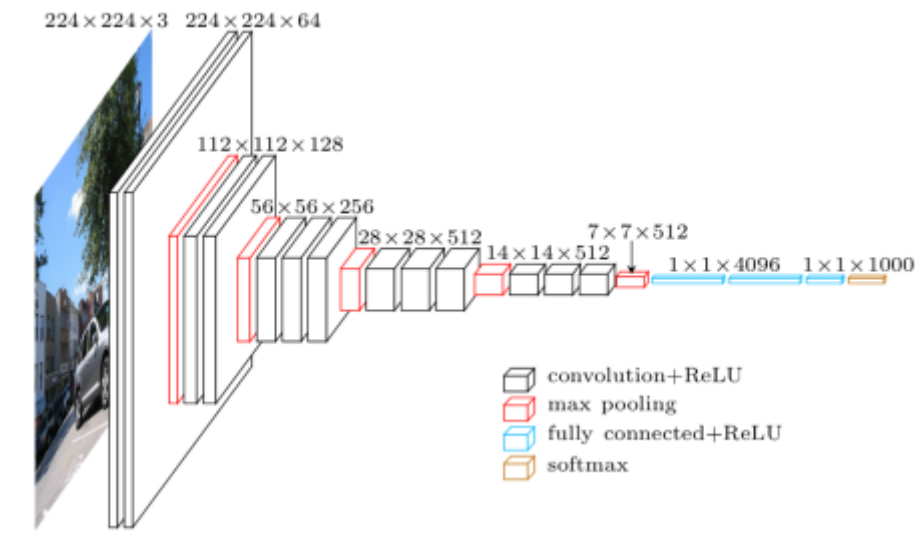


图 VGG16 模型结构

2）ResNet50

ResNet 主要是用于解决随着网络的加深训练集准确率下降的现象。

ResNet 针对这个问题提出了一种全新的网络，叫深度残差网络，它允许网络尽可能的加深，其中引入了全新的结构如下所示：

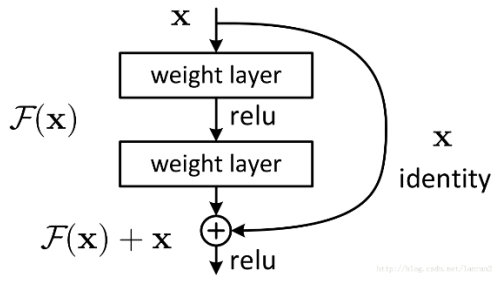


图 Shortcut Connection

ResNet 提出了两种 mapping：一种是 identity mapping，另一种是 residual mapping，所以最后的输出是$y = F(x) + x$。identity mapping 就是公式中的 x；residual mapping 指的是$y - x$，所以残差指的就是$F(x)$部分。

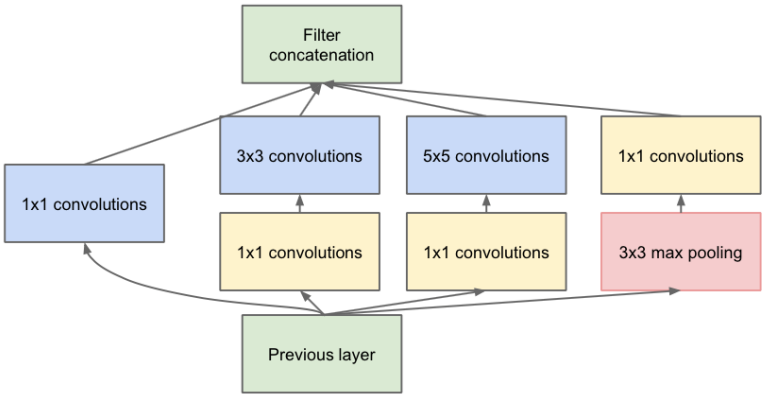
下表为不同 ResNet 模型结构，可以看出所有模型都由 5 部分组成，分别是：conv1, conv2_x, conv3_x, conv4_x, conv5_x，之后是 GPA 和用于分类的全连接层。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

3）InceptionV3

Inception 网络主要是解决了在增加网络深度和宽度的同时减少了整个网络的参数。Inception 网络将 1x1，3x3，5x5 的卷积核和 3x3

的 pooling 堆叠在一起，一方面增加了网络的 width，另一方面增加了网络对尺度的适应性。Inception 模型在 3x3 前，5x5 前，max pooling 后分别加上了 1x1 的卷积核,起到了降低特征图维度的作用。下图为 Inception V1 模型的网络结构。



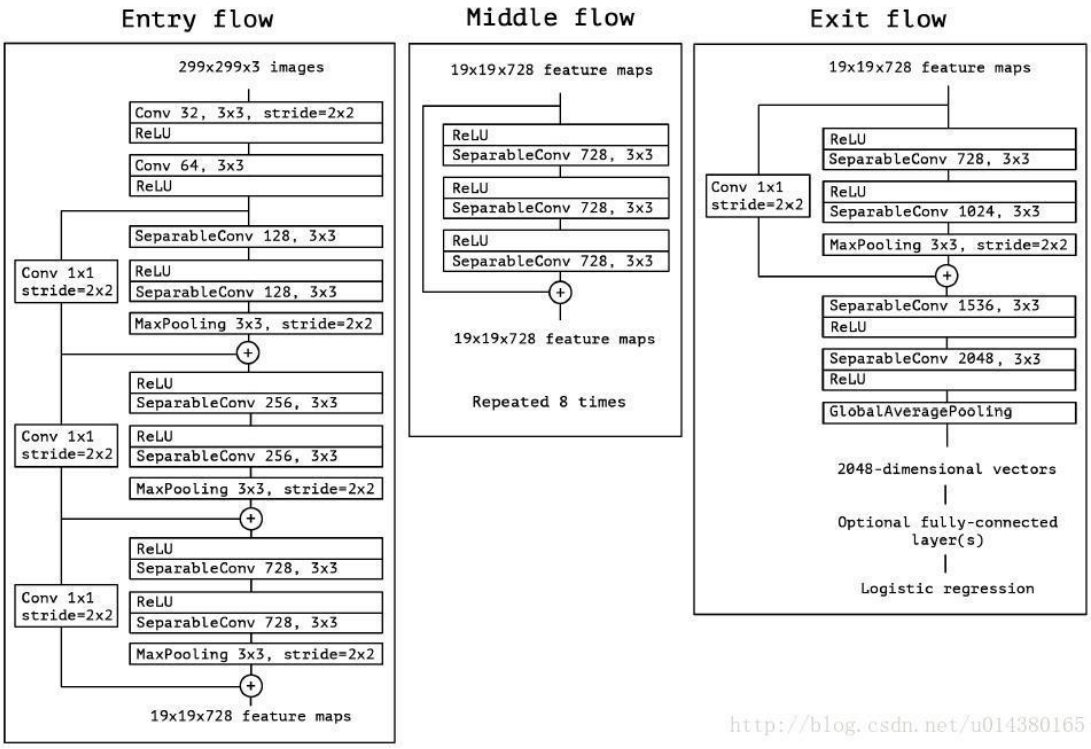
Inception V3 模型，相对于 Inception V1 和 V2 模型，一个最重要的改进是分解（Factorization），将 7x7 分解成两个一维的卷积（1x7,7x1），3x3 也是一样（1x3,3x1）。这样的好处，既可以加速计算，又可以将 1 个 conv 拆成 2 个 conv，使得网络深度进一步增加，增加了网络的非线性。

4）Xception

Xception 是 google 继 Inception 后提出的对 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。

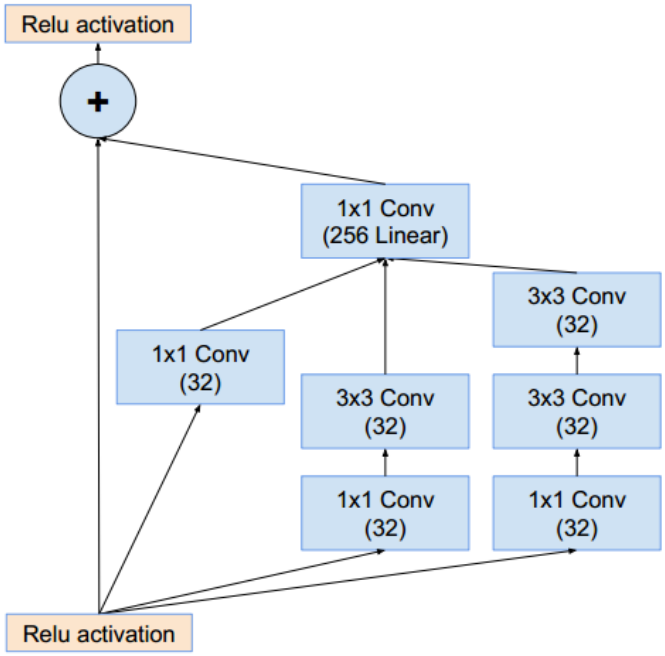
depthwise separable convolution 就是将传统的卷积操作分成两步，假设原来是 3*3 的卷积，那么 depthwise separable convolution 就是先用 M 个 3*3 卷积核一对一卷积输入的 M 个 feature map，不求和，生成 M 个结果；然后用 N 个 1*1 的卷积核正常卷积前

面生成的 M 个结果，求和，最后生成 N 个结果。下图是 Xception 网络结构示意图。

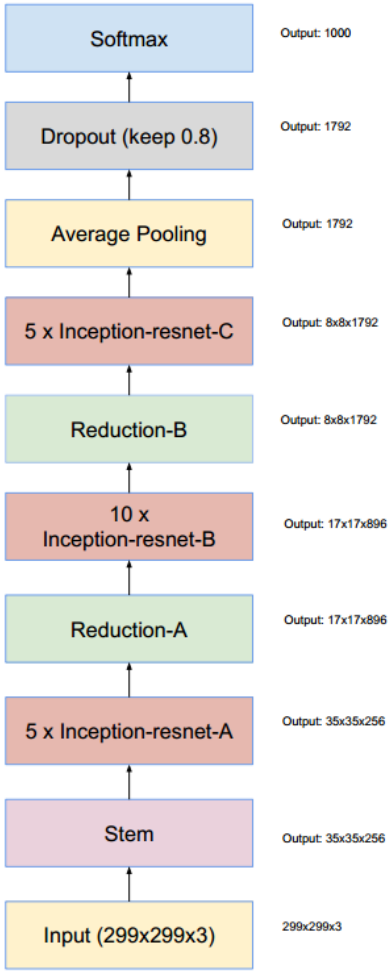


5）InceptionResNetV2

InceptionResNet 网络是 Inception V4 和 ResNet 结合产生的模型，试验结果表明，residual connections 可以提高 Inception 网络的准确率，并且不会提高计算量。下图为 Iception-ResNet 其中的一个典型模块结构图。



下图为 InceptionResNet 网络结构：



2.3.2. 损失函数

本项目使用了对数损失函数：

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

N 是测试集图片的数量，M 是图像类别标签，log 是自然对数，在图像 i 属于类别 j 时 y_{ij}=1，否则 y_{ij}=0，p_{ij}是模型预测的图像 i 数据类别 j 的概率。

2.3.3. 优化算法

1）Adadelta

这个算法是对 Adagrad 的改进，和 Adagrad 相比，就是分母的 G 换成了过去的梯度平方的衰减平均值：

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t.$$

这个分母相当于梯度的均方根 root mean squared (RMS)，所以可以用 RMS 简写：

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t.$$

其中 E 的计算公式如下，t 时刻的依赖于前一时刻的平均和当前的梯度：

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2.$$

梯度更新规则:

此外，还将学习率 η 换成了 RMS[Δ θ]，这样的话，我们甚至都不需要提前设定学习率了：

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t.$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t.$$

超参数设定值:

γ 一般设定为 0.9。

2）RMSprop

RMSprop 和 Adadelta 都是为了解决 Adagrad 学习率急剧下降问题的，梯度更新规则:

RMSprop 与 Adadelta 的第一种形式相同：

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2.$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t.$$

超参数设定值:

建议设定 γ 为 0.9, 学习率 η 为 0.001。

3）Adam

该算法是另一种计算每个参数的自适应学习率的方法。

除了像 Adadelta 和 RMSprop 一样存储了过去梯度的平方 vt 的指数衰减平均值 ，也像 momentum 一样保持了过去梯度 mt 的指数衰减平均值：

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t.$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2.$$

如果 mt 和 vt 被初始化为 0 向量，那它们就会向 0 偏置，所以做了偏差校正，通过计算偏差校正后的 mt 和 vt 来抵消这些偏差：

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}.$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

梯度更新规则:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

超参数设定值:

建议 β1 = 0.9，β2 = 0.999，ϵ = 10e-8。

实践表明，Adam 比其他适应性学习方法效果要好。

2.3.4. GAP

Global Average Pooling 主要是用来解决全连接的问题，其主要是讲最后一层的特征图进行整张图的一个平均池化，形成一个特征点，将这些特征点组成最后的特征向量进行 softmax 分类。使用 Global Average Pooling 可以减少参数数量。

2.3.5. 迁移学习和模型微调

迁移学习是机器学习技术的一种，其中在一个任务上训练的模型被重新利用在另一个相关的任务上。

模型微调是指利用已经训练好的模型权重，锁住前面几层权重不变，只对顶层未锁定的层的权重进行重新训练。

2.4.基准模型

我的目标是超过 Kaggle Private Leaderboard 第 144 名，模型 logloss 小于 0.25634，排入 Private Leaderboard 前 10%。

3. 方法

3.1.数据预处理

数据集中每张图片大小为（480， 640， 3）的 RGB 图片,数据预处理先是根据模型需要，将原始图像数据从（480,640）缩放到了（224,224）或（299,299）。然后分别对缩放后的图像进行了图像旋转、图像平移及去中心化处理。

图像预处理使用了 Keras 的 ImageDataGenerator 函数，例如对 ResNet50 模型的训练集图像预处理的函数具体参数设置如下：

```
resnet50_train_datagen = ImageDataGenerator(
    rotation_range=10.,
    width_shift_range=0.05,
    height_shift_range=0.05,
    preprocessing_function=resnet50.preprocess_input)
```

3.2.执行过程

首先,模型建立使用了Keras内置预训练模型VGG16，ResNet50，Inception V3，Xception，InceptionResNet V2，这些预训练模型的权重都是在ImageNet上训练得到的。

加载模型后，对于VGG16模型去掉了预训练模型的全连接层，然后重新增加了1个flatten层和3个全连接层;其他模型每个都去掉了顶部的全连接层，使用 Keras 内置的 GlobalAveragePooling2D()和Dropout(0.5)代替，最后再加一层softmax分类层。

模型训练前进行了模型微调，主要是采用了锁层的方式，形成的模型如下：

序号	模型	锁定层	模型名称
1	VGG16	不锁层	Vgg16
2	ResNet50	77	ResNet50_77
3		139	ResNet50_139
4	Inception V3	100	InceptionV3_100
5		196	InceptionV3_196
6	Xception	35	Xception_35
7		75	Xception_75
8	InceptionResNet V2	274	InceptionResNetV2_274
9		616	InceptionResNetV2_616

由于每个模型的收敛速度不同，通过反复测试，每个模型训练的epoch次数、优化器和学习率如下：

序号	模型	Epoch次数	优化器	学习率
1	VGG16	10	Adam	1e-4
2	ResNet50	5	Adam	1e-4
3	Inception V3	1	Adam	1e-4
4	Xception	1	Adam	1e-4
5	InceptionResNet V2	1	Adam	1e-4

在训练集和测试集划分方面，在训练集里选取了 p045，p056，p061 3 名司机作为验证集。

训练过程中使用了 Keras 提供的 ImageDataGenerator，来提供训练集数据，验证集数据和测试集数据。

最后用 Pandas 生成了提交 kaggle 的预测结果文件。

3.3.完善

在对上述 11 个模型进行训练后，预测结果都没有达到 logloss 小于 0.25634 这个目标，主要采取了以下两种方式将 logloss 值降到了 0.25634 以下：

1）采用了模型融合的方式，对多个模型的预测结果取平均值；

2）采用了 Bagging 的方式，首先将训练集数据分成了 5 份，每次训练将其中 4 份用作训练集，另外 1 份作为验证集，然后对模型进行 5 次训练和预测，最后将 5 次的预测结果进行平均。

4. 结果

4.1.模型的评价与验证

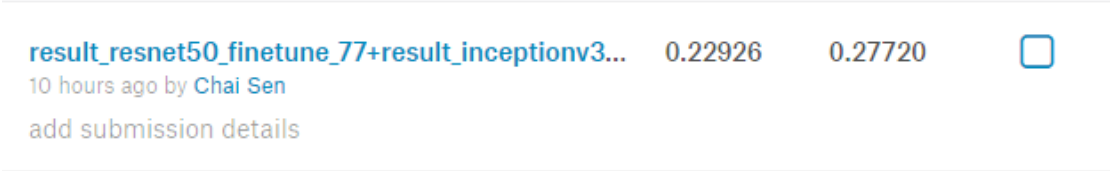
3.2 节 11 个模型在 kaggle 上的预测结果如下表：

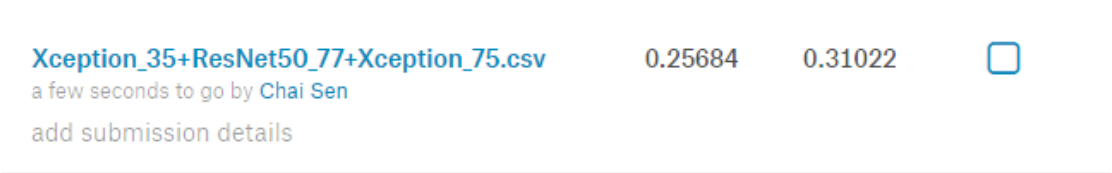
No.	Model	Private Score	Public Score	val_loss
1	Vgg16	0.65615	0.59138	0.4112
2	InceptionResNetV2_616	0.79481	0.83071	0.4622
3	InceptionResNetV2_274	0.81639	0.85342	0.4033
4	InceptionV3_196	0.30665	0.33734	0.4576

5	InceptionV3_100	0.30665	0.33734	0.2925
6	Xception_75	0.35558	0.48830	0.3238
7	Xception_35	0.36397	0.40250	0.3499
8	ResNet50_139	0.61750	0.58870	0.4400
9	ResNet50_77	0.31987	0.39835	0.2201

两个融合模型在 kaggle 上的预测结果如下表：

No.	Model	Private Score	Public Score	val_loss
1	Inception_V3_100 ResNet50_77 Xceptoin_75	0.22926	0.27720	-
2	Xception_35 InceptionResNetV2_274 InceptionV3_100	0.25684	0.31022	-





Bagging 方法生成的模型在 kaggle 上的预测结果如下表：

No.	Model	Private Score	Public Score	val_loss	Valid Dataset
1	Xception_35_1	0.45896	0.62099	0.7809	p061, p064, p066, p072, p075, p081
2	Xception_35_2	0.32761	0.31200	0.0779	p021, p022, p024, p026, p035
3	Xception_35_3	0.40406	0.42207	0.2303	p039, p041, p042, p045, p047
4	Xception_35_4	0.28817	0.35420	0.0834	p049, p050, p051, p052, p056
5	Xception_35_5	0.44825	0.43456	0.6258	p002, p012, p014, p015, p016
6	Xception_35_Merge	0.22072	0.26245	-	

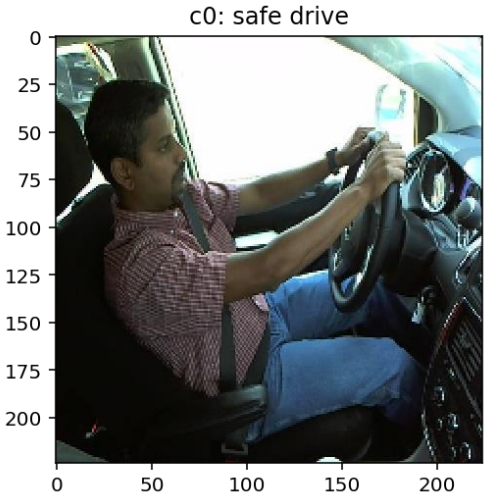
可以看出通过 bagging 方式创建的融合模型的预测结果最好 kaggle 上在 Private Score 上的得分为 0.22072，低于 0.25634，排名 91 名。

5. 项目结论

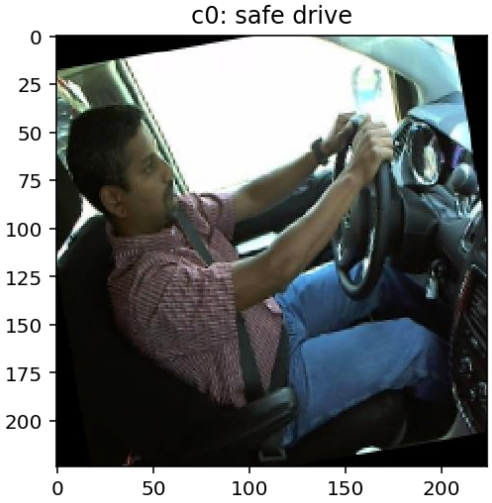
5.1.结果可视化

本项目中为了增强模型的泛化能力，利用 opencv 对图像数据进行了数据增强。原始图像为（480,640,3）的 RGB 图像，如下图所示：

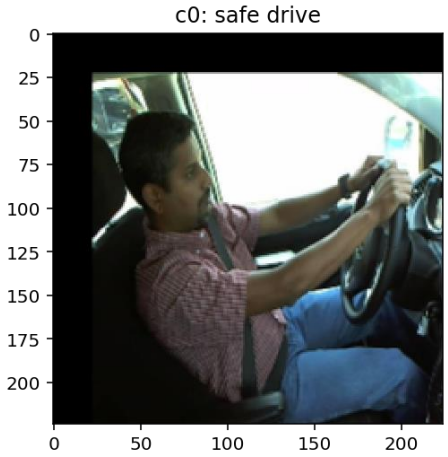
利用 opencv 的 resize 函数将图像大小缩放到（224,224），如下图所示：



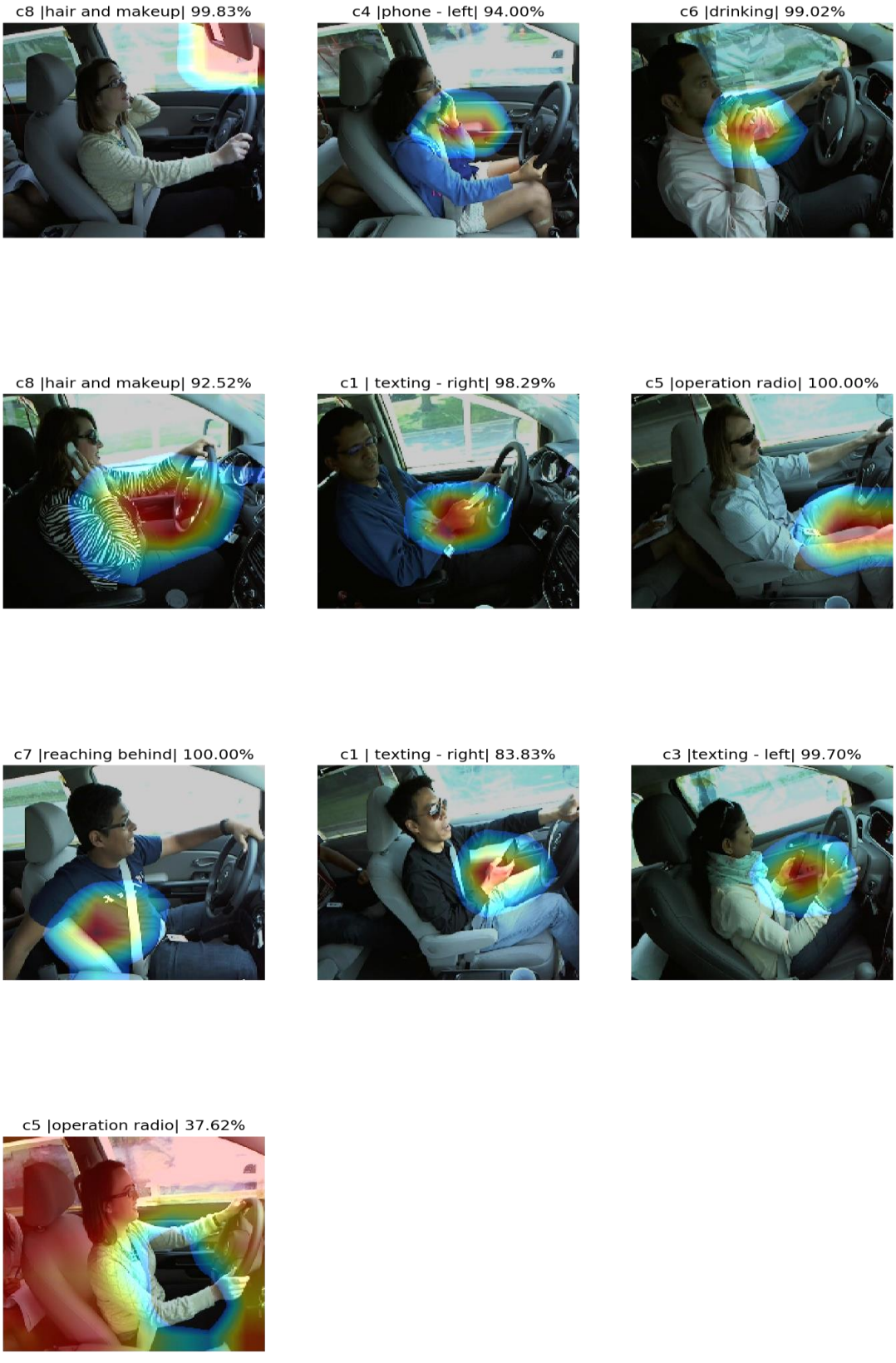
利用 opencv 的 getRotationMatrix2D 和 warpAffine 函数将图片逆时针旋转 10°，如下图所示：



利用 warpAffine 函数将图片向下和向右平移 10%，如下图所示：



利用 Class Activation Mapping 可以看到 CNN 网络对于每一个分类更关注图像的哪些区域。



下图为 Xception_35 + InceptionResNetV2_274 + InceptionV3_100 融合模型在测试集中随机选取的 20 张图片的预测

结果。

5.2.对项目的思考

该项目使用了 CNN 完成了对分神司机图像的分类，整个处理流程分为数据预处理，模型构建，模型训练，模型改进等阶段。其中模型改进阶段，产生的单个模型都没有达到预期目标，最终通过模型融合的方式到达了预计目标。

整改项目最困难的部分为模型微调阶段，需要尝试锁定一个模型不同的层进行模型训练，耗时巨大。VGG16 模型对顶层的 3 个全连接层依赖比较大，刚开始训练 VGG16 模型时将顶层的 3 个全连接层替换成了 GAP 和 1 个全连接分类层，模型的预测准确性很差，恢复 3 个全连接层后，模型预测准确性提升很多。

在对数据进行预处理的过程中学习掌握了 opencv 基本的图像处理方法。

5.3.需要作出的改进

经过思考我认为项目还可以在以下方面做出改进：

1) 模型目前使用的 Adam 优化器的学习率为 1e-4，以后可以尝试在模型训练几代之后收到调节优化器学习率，或每个几代设置学习率衰减率，同样达到调剂学习率的目的；
2) 更灵活的划分测试集与验证机，目前除了 bagging 方法所有模型都是在同样的三个司机的图像文件上进行验证，今后可以使用不同验证集对模型进行验证，或者使用 K-fold 方式对模型进行训练。

参考文献

[1]. 黄创新，汽车驾驶员的行为状态识别[D]，2007，北京科技大学

[2]. https://www.kaggle.com/c/state-farm-distracted-driver-detection

[3]. Simonyan Karen, Zisserman, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. eprint arXiv:1409.1556

[4]. https://www.kaggle.com/c/state-farm-distracted-driver-detection#evaluation

[5]. He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep Residual Learning for Image Recognition[J]. eprint arXiv:1512.03385

[6]. Christian Szegedy, Vincent Vanhoucke, Sergey, et al. Rethinking the Inception Architecture for Computer Vision[J]. eprint arXiv:1512.00567v3

[7]. Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning[J]. eprint arXiv:1602.07261v2

[8]. Francois Chollet, Xception: Deep Learning with Depthwise Separable Convolutions[J]. eprint arXiv:1610.02357v3

[9]. Bolei Zhou, Aditya Khosla, Agata Lapedriza, et al. Learning Deep Features for Discriminative Localization[J]. eprint arXiv:1512.04150v1

[10]. https://github.com/elninowang/mlnd_distracted_driver_detection/blob/master/keras-resnet50-visual-finetune.ipynb