东北大学秦皇岛分校

计算机与通信工程学院

人工智能导论课程大作业

设计题目:司机驾驶行为检测模型

班级	计科 1803 班
学号	20188117
学生姓名	**项溢馨**
指导教师	徐长明
设计时间	**2020 年 5 月 8 日— 2020 年 6 月 15 日**

目录

1. 引言

有数据表明，在每年的事故死亡人数中，交通事故造成的伤害居于首位，占到 78%左右。其中有五分之一的车祸是由司机走神导致的。

“道路千万条，安全第一条。行车不规范，亲人两行泪。”如果能够自动检测司机的驾驶行为是否存在安全隐患，从而就能约束司机驾驶行为，降低交通事故的风险。

本设计测试数据来源于 Kaggle 上的一个竞赛项目，所用数据集来自 State Farm 收集的驾驶图片数据库。驾驶员可能存在的走神的行为大概有如下几种：左右手用手机发短信，左右手手持打电话，调收音机，喝饮料，拿后面的东西，整理头发和化妆，和其他乘客交谈。

该项目本质上是一个监督分类学习的问题，旨在得到一个可检测司机驾驶行为的模型。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。

通过分析已有研究成果，可知现有驾驶人行为识别方法主要存在两方面问题: 一是数据获取不稳定，无论是通过仪器估计汽车行驶状态，还是通过交通监控获取驾驶人图像，都无法保证稳定的数据源；二是识别率较低，由于传统识别方法本身的局限性，现有驾驶人行为识别方法的识别率均较低。

近年来，深度学习在图像分类、图像检测、自然语言处理等诸多领域表现优异。常用的深度学习算法包括深度置信网络(Deep Belief Networks，DBN)、卷积神经网络(Convolutional neural network，CNN) 和循环神经网络(Recurrent Neural Network，RNN) 。

其中 CNN 主要应用于计算机视觉领域的图像分类。传统的分类器需要人工提取特征，而 CNN 利用卷积池化层自动提取特征，再利用神经网络进行特征分类，使 CNN 表现出比传统方法优越的性能。由于其网络层表现出来的图像卷积效果，以及网络结构体现出来的从底层特征到高层特征的抽象过程，这种网络结构具有较高的平移不变性、尺寸不变性、倾斜不变性等特性。随着网络深度的增加和各种防止过拟合方法的提出，它在图像分类等任务上取得了一系列突破。

本设计主要通过对深度学习中经典的模型——卷积神经网络 CNN 进行改进，使用 Keras 和 TensorFlow 后端来训练 CNN 模型和迁移 VGG16 模型来检测司机在驾驶过程中出现不同的走神行为，从而达到发出精准警告信息的目的，进而有效地降低交通事故的发生。

2. 数据准备

2.1 原数据

数据集来自 State Farm 提供的训练和测试数据集，下载地址如下：https://www.kaggle.com/c/state-farm-distracted-driver-detection/data

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]：

（1） c0 safe driving：手握方向盘，目视道路，则为安全驾驶；
（2） c1 texting - right：右手拿手机或手指接触手机，置于目光前方或右前方；
（3） c2 talking on the phone – right：右手拿手机置于耳边，右侧耳边频率高；
（4） c3 texting – left：左手拿手机或手指触碰手机，置于目光前方或左前方；
（5） c4 talking on the phone – left：左手拿手机置于耳边，左侧频率高；
（6） c5 operating the radio：手指触收音机按钮部位，目光/面部姿态右下；
（7） c6 drinking：饮料瓶触碰手和嘴部；
（8） c7 reaching behind：身体/面部姿态右后，手方向向后；
（9） c8 hair and makeup：手位于头部或面朝镜子；
（10） c9 talking to passenger：司机面部姿态和身体姿态向右后或右侧，相应方向有人。

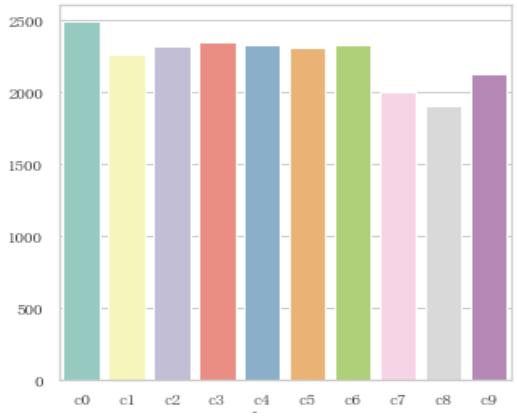


图 1 不同种类行为统计

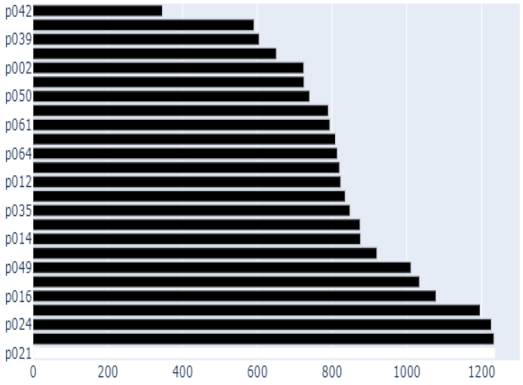
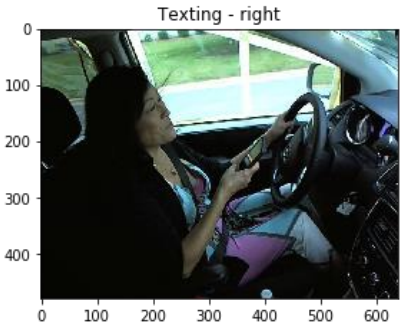


图 2 不同司机图片数量

训练集中包含 22424 张图片，测试集中包含 79726 张图片。数据集的图片尺寸均是 640x480。根据统计可以看出，每个状态下面的图片是比较均匀的。

2.2 数据导入

先导入 driver_imgs_list.csv 文件，使用 classname 和 img 标签对训练图片进行配对标记。根据标签对导入的图片集进行分类处理，可以在图三观察到每种类别的图例。



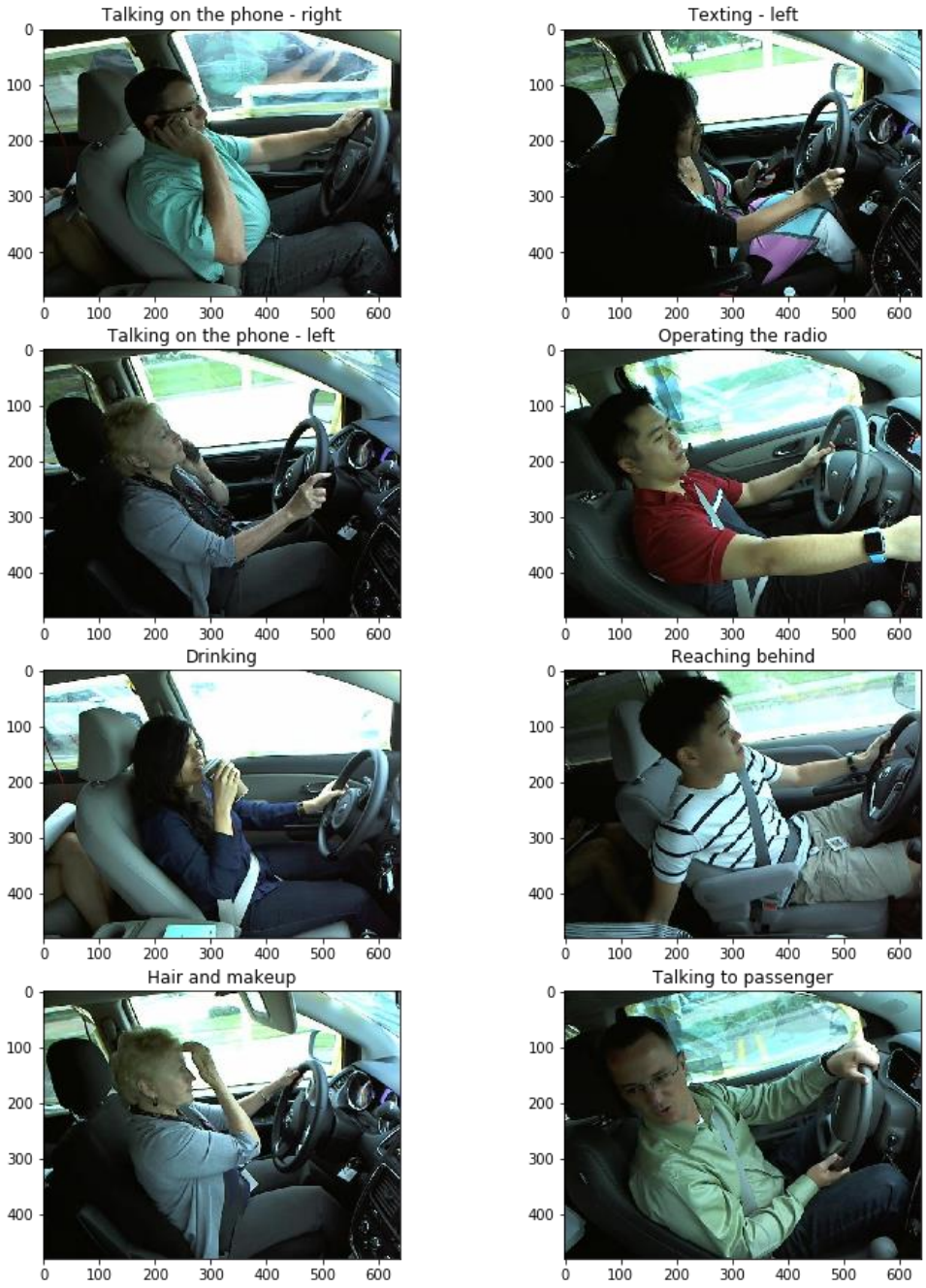


图 3 分类图例

可以发现，图片都是从同一个角度拍摄，且图片环境相似度极高，有一定可能性导致过拟合；不同司机图片数量不同，驾驶员特征各异，很容易成为模型学习的主要特征，这是我们所不希望看到的；图片明暗程度较差，可能导致部分特征缺失；此外个别图片还存在干扰的人物因素。

原图像大小为 480×640×3，太大的图像并不利于网络的训练，为适应网络的输入，将原图像压缩为 64×64×1。

3. 模型结构

3.1 模型 1 朴素 CNN 模型

采用 Vanilla CNN 的网络结构，旨在去掉多任务学习而且使用彩色图像。把图片的部分相邻像素变成长宽更小，高度越深的单元，从而提炼出局部的特征，一层一层将特征往后映射，之后提取出最终的特征。

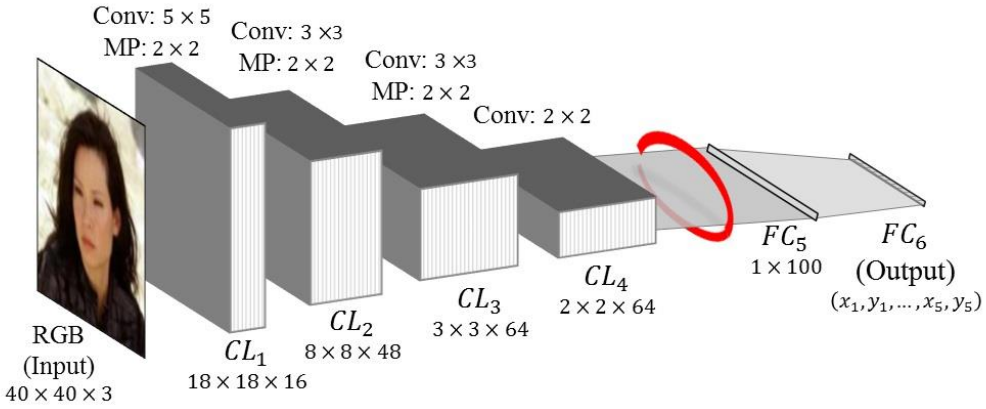


图 4 Vanilla CNN 结构示例

我们需要构造一个包含 4 层卷积层 Convolutional layer、1 个压平层 Flatten layer 和 2 个全连接层 Dense layer 的包含 1 层隐藏层的简单 CNN 模型。

卷积层使用卷积核进行特征提取和特征映射。对于图片中的每一个特征首先局部感知，然后更高层次对局部进行综合操作，从而得到全局信息。卷积操作是利用多个滤波器在高维度的输入特征图上运算生成新的低维度输出特征图的过程，在卷积过程中，特征图经过不同的滤波器处理后将生成具有不同局部特征信息的张量，将这些张量组合起来生成新的具有全局特征信息的输出特征图。

用 l 表示当前层数，对于卷积过程，上一层的输出a^l通过一组卷积权重矩阵w^{l+1}，加上偏差量b^{l+1}，再经过非线性激活函数 f (x)，生成一组特征图a^{l+1}。 卷积过程可表示为：

$$a^{l+1} = f(z^{l+1}) \tag{1}$$

$$z^{l+1} = W^{l+1} * a^l + b^{l+1} \tag{2}$$

式中 z 为未经过激活的卷积层输出；*表示卷积运算。

池化层的作用是提取局部均值与最大值，减小了来自上层隐藏层的计算复杂度，保留显著特征、降低特征维度，增大 kernel 的感受野。池化操作并不会改变特征图的通道数，它一方面是为了降低特征图的尺寸，减小网络中的参数量，精简模型的时间复杂度，另一方面是为了保留特征图中突出的特征点，筛除无用的杂乱信息，有效地避免过拟合发生。

Convolution 卷积层之后是无法直接连接 Dense 全连接层的，需要把 Convolution 层的数据压平（Flatten），然后就可以直接加 Dense 层了。

全连接层输出a^{L+1}，与一组输出权重矩阵W^L相乘，加上偏差量b^L，再经过非线性激活函数 f(*)，得到最终的预测概率向量输出a^L，其输出为

$$a^L = f(z^L) \tag{3}$$

$$z^L = W^L a^{L-1} + b^L = W^L f(z^{L-1}) + b^L \tag{4}$$

全连接层主要用于处理卷积层和池化层提取到的各个部分特征信息，通过权值矩阵将它们整合成一张完整的特征图。全连接层操作可以看作成一种特殊的卷积层操作，它的卷积核

的大小与输入的特征图的大小相等，经过卷积操作后形成 1x1 的卷积，最后采用归一化指数函数生成分类结果。

根据图像原大小为 64x64，第 1 层卷积的卷积核个数定为 64 个，尺寸为 3x3；第 2 层卷积的卷积核个数定为 128 个，尺寸为 3x3；第 3 层卷积的卷积核个数定为 256 个，尺寸为 3x3；第 4 层卷积的卷积核个数定为 512 个，尺寸为 3x3。每层卷积过后，都需要用 2x2 的池化层过滤。

为防止模型过拟合，我们需要以一定的概率 p 对参数进行随机采样。因此在 4 层卷积层上我们先添加一层概率为 0.5 的 Dropout layer，来随机删除神经网络中的部分神经元，正则化等来解决此问题，使网络更具有鲁棒性和普适性。之后对前面数据进行压平处理，送到全连接层进行输出。

CNN 有监督训练的实质是优化权重和偏差，使得网络输出与训练数据的期望输出或给定标签相匹配。在多特征提取阶段之后，输出和目标之间仍然存在误差，该误差可被描述为损失函数 E。

$$\mathrm{E} = \frac{1}{2}||a^L - y|| \tag{5}$$

式中 y 为期望输出值。通过迭代调整权重，来搜索上述损失函数 E 的最小值。权重调整公式为:

$$W_{t+1}^{l+1} = W_t^{l+1} - \eta \frac{\partial E_t}{\partial W_t^{l+1}} \tag{6}$$

$$b_{t+1}^{l+1} = b_t^{l+1} - \eta \frac{\partial E_t}{a b_t^{l+1}} \tag{7}$$

式中：η为学习率；t 为迭代次数；W_{t+1}^{l+1}为第 l+1 层权重第 t+1 次迭代的结果；b_{t+1}^{l+1}为第 l+1 层偏差量第 t+1 次迭代的结果；E_t为第 t 次迭代的损失函数。

定义误差项δ如下：

$$\delta = \frac{\partial E}{\partial z}，\ \delta \in \mathbb{R} \tag{8}$$

式中$\mathbb{R}$表示实数集。

由式(3)、式(4)可知：输出层的误差项δ^L为

$$\delta^L = (a^L - y) \odot f'(z^L) \tag{9}$$

式中⊙表示 Hadamard 乘积运算；f'(*)为激活函数 f(*)的导数。

则输出层前一个隐藏层的误差项δ^{L-1}为：

$$\delta^{L-1} = (W^L)^T \cdot \delta^L \odot f'(z^{L-1}) \tag{10}$$

式中$(\cdot)^T$表示取共轭转置。

全连接层的权重更新为：

$$\frac{\partial E}{\partial W^L} = \delta^L (a^{L-1})^T \tag{11}$$

如果用 l 代表卷积层，则卷积层前一层的误差项为:

$$\delta^{l-1} = \delta^l * \ (W^l)^\theta \odot f'(z^{l-1}) \tag{12}$$

式中$(*)^\theta$代表矩阵旋转 180°。

卷积层的权重更新为：

$$\frac{\partial E}{\partial W^l} = a^{l-1} * \delta^l \tag{13}$$

3.1 模型 1 的改进

在模型 1 单层隐藏层结构上，我们改成三层隐藏层结构。每层由 2 层卷积层和 1 层池化层组成，最后进行一次随机采样。在卷积核个数上，我们修改原来较大的值，三层结构的卷积核分别为 32、64、128。小卷积核能增强网络容量和模型复杂度，减少卷积参数个数。

示意图如下：

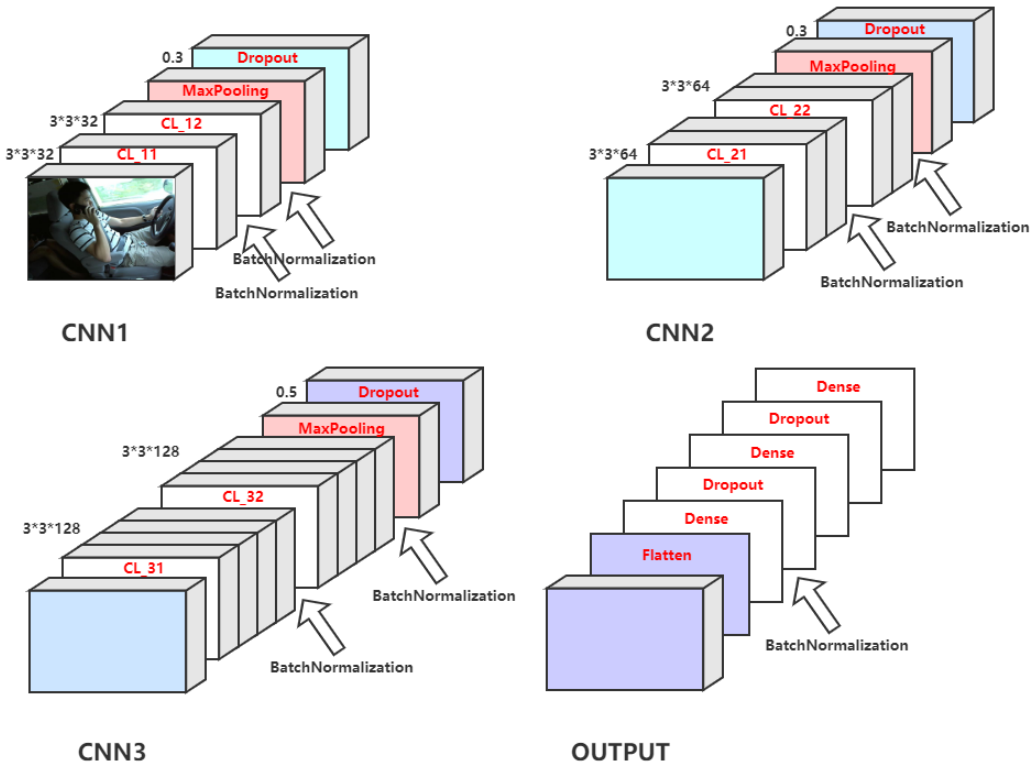


图 5 改进 Vanilla CNN 模型示意图

单层神经网络只能用于表示线性分离函数，也就是非常简单的问题，比如分类问题中的两个类可以用一条直线整齐地分开。多个隐藏层可以用于拟合非线性函数。三层隐藏层从理论上来说可以更好地拟合函数。

3.2 模型 2 VGG16 模型

VGG 是由 Simonyan 和 Zisserman 在文献 《Very Deep Convolutional Networks for Large Scale Image Recognition》中提出卷积神经网络模型，其名称来源于作者所在的牛津大学视觉几何组(Visual Geometry Group)的缩写。

VGG 中根据卷积核大小和卷积层数目的不同，可分为 A，A-LRN,B,C,D,E 共 6 个配置(ConvNet Configuration)，其中以 D,E 两种配置较为常用，分别称为 VGG16 和 VGG19。VGG16 共包含 13 个卷积层、3 个全连接层和 5 个池化层。

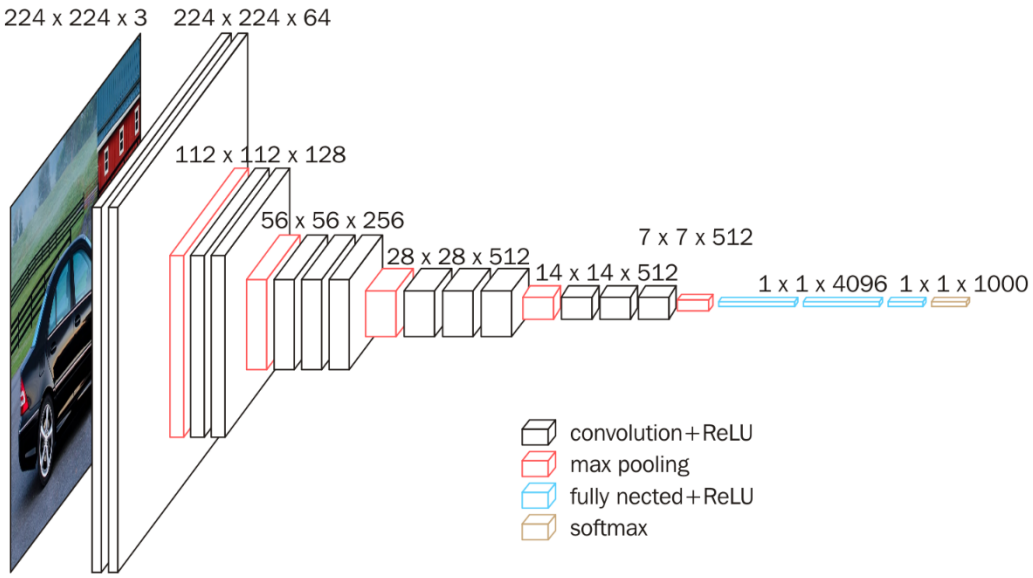


图 6 VGG16 模型示意图

使用 VGG16 模型进行迁移学习，与朴素 CNN 模型相比，具有更深层的网络结构。卷积串联比单独使用一个较大的卷积核,拥有更少的参数量,同时会比单独一个卷积层拥有更多的非线性变换。

但 VGG 耗费更多计算资源，并且使用了更多的参数（这里不是 3x3 卷积的锅），导致更多的内存占用（140M）。其中绝大多数的参数都是来自于第一个全连接层。

4. 模型的训练、测试、调优

4.1 评价函数

算法的预测结果根据预测值和真实值的真假组合可以划分为 TP、TN、FP、FN，表示的含义分别是预测值和真实值都为真、预测值和真实值都为假、预测值为真，真实值为假、预测值为假，真实值为真。通常情况下，采用准确度 ACccumry 指标评估目标分类网络精度的好坏，它的计算公式如式(14)。

$$\text{Accumry} = \frac{TP}{TP+FP} \tag{14}$$

采用 categorical_crossentropy 的计算原理来使用 loss 对预测结果进行评分，categorical _crossentropy 的数学公式如式(15)。

$$\log loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} y_{ij}\log(p_{ij}) \tag{15}$$

其中 N 表示测试条目数；M 表示标签数；y_{ij}为条目 i 的标签 j 编号；p_{ij}是条目 i 是标签 j 的概率。这和 Kaggle 比赛中要求的评估方法是一致的，对每个分类的概率分别求和。

4.2 模型 1 的测试过程

4.2.1 基于模拟的少量数据

为了检验模型对识别过程的表达能力，本文使用了 train 集中的 10 种模式图像，着力于识别司机的不同行为，分别在不同的数据量和不同的网络深度进行了训练，对朴素 CNN 模型迭代 10 次（迭代 3 次之后可以看到算法已完全收敛，损失率不再下降），对改进 CNN 模型迭代了 5 次，结果如图所示。

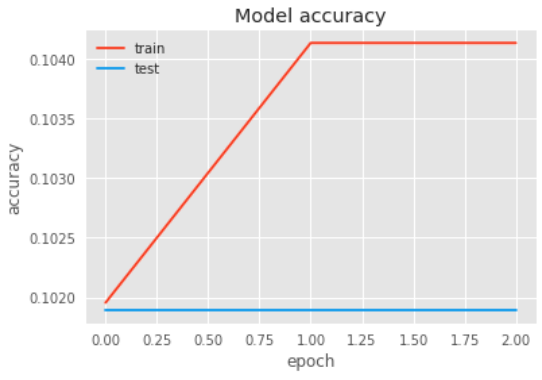


图 7 朴素 Vanilla CNN 模型准确率

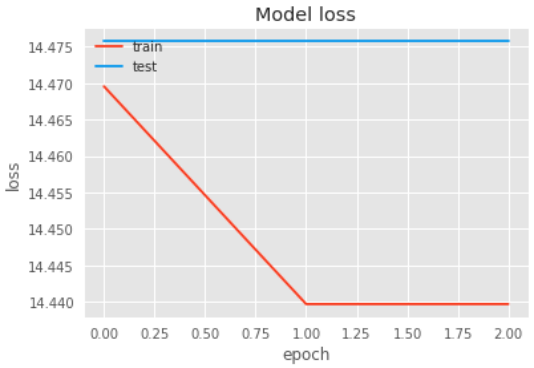


图 8 朴素 Vanilla CNN 模型损失率

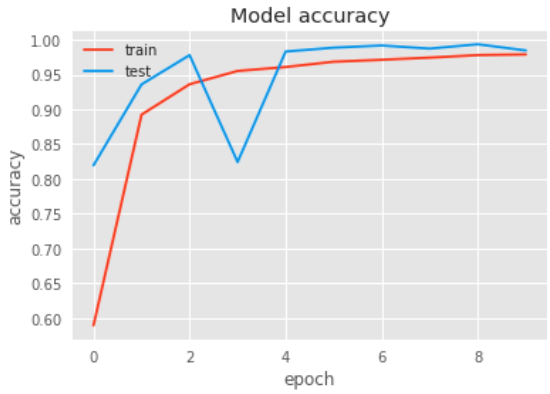


图 9 改进 Vanilla CNN 模型准确率

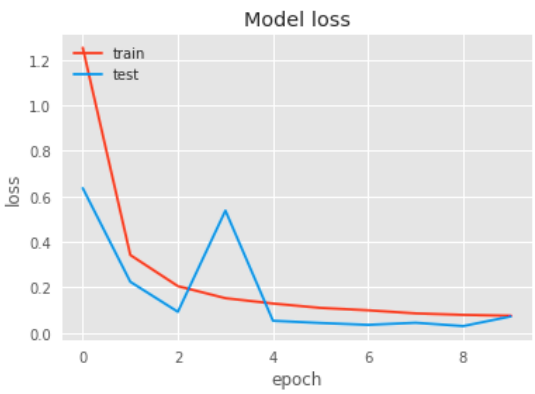


图 10 改进 Vanilla CNN 模型损失率

可以看出改进后模型的准确率更加趋近于 1，损失率基本持平。朴素 VCNN 模型 10 次迭代后最终的准确率达到了 0.102，损失率为 14.475。改进 VCNN 模型最终准确率为 0.994，损失率为 0.030。证明了使用三层隐藏层结构可以有效提高模型的识别精确度。

由图 10 可看出，改进 CNN 在训练过程中收敛稳定，随着训练次数的增加，损失函数逐渐趋近于 0。

由表 1 可知：10 类驾驶行为中，第 6 类驾驶行为（开收音机）和第 8 类驾驶行为（转身）的识别率最高，达到了 100%；第 10 类驾驶行为（和乘客交谈）的识别率最低，但也达到了 98.00%。第 10 类驾驶行为主要被错识别成了第 9 类（整理头发和化妆）与第 6 类（开收音机）驾驶行为，错误识别率分别为 1.25%和 0.50%，其错误识别的主要原因是因为“整理头发”和“开收音机”的行为 与“交谈”的行为较为相似。

预测值	0	1	2	3	4	5	6	7	8	9
True	461	499	478	489	464	461	452	377	383	393
False	4	7	2	4	1	0	1	0	1	8

表 1 改进 Vanilla CNN 模型不同行为模式识别率

由改进 VCNN 模型绘制的热力图如下。

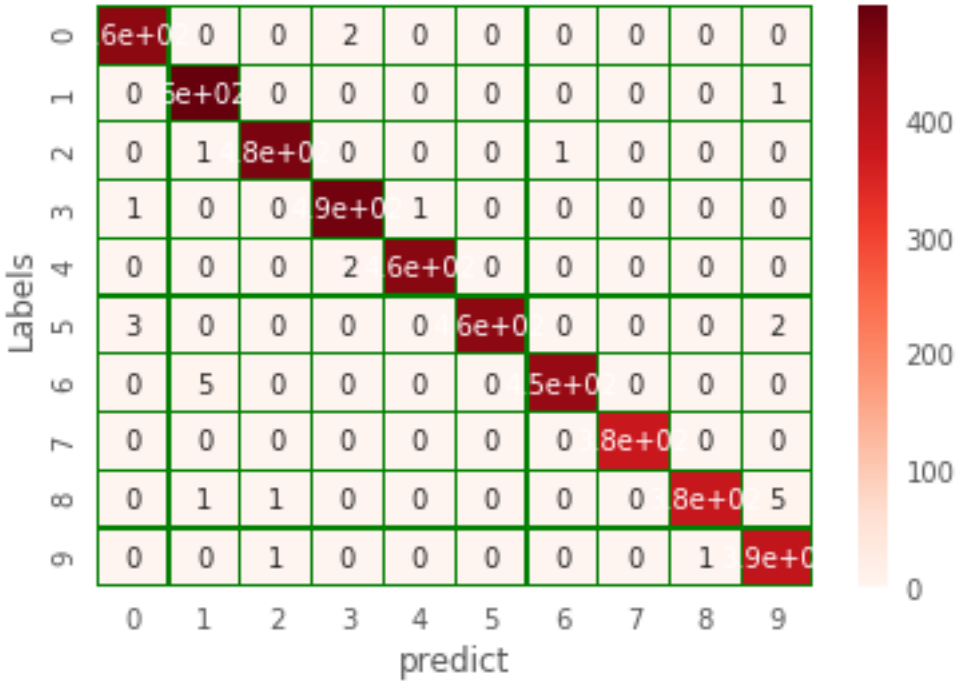


图 11 改进 Vanilla CNN 模型不同行为模式识别热力图

改进后 VCNN 对大部分类别的识别准确率远远高于朴素 VCNN，部分识别准确率达到了 100%。对测试数据进行抽样比对。

对于下面 4 张测试图，朴素 VCNN 的识别结果均为 drinking（1）；改进 VCNN 的识别结果分别为 Talking on the phone - right（1.00）、Safe driving（0.99）、Operating the radio（0.97）、Safe driving（0.98）。

可见改进模型不仅在识别准确度上大幅提升，还提高了可能性概率。很多图片的可能性已经趋近于 1，对于判断行为模式给出了一个巨大的飞跃。但对于测试数据，中期呈现不稳

定状态，变化震荡。说明应用图像变换后，模型准确度存在进一步提升，但可能导致网络过拟合现象。

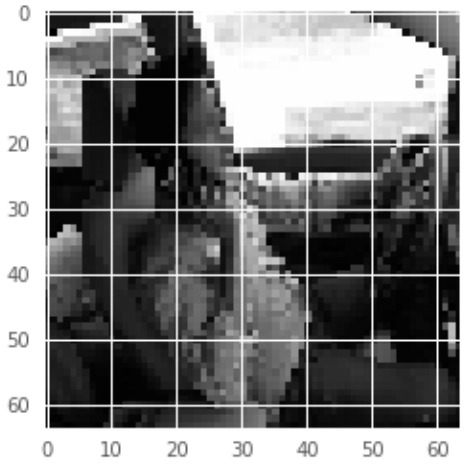


图 12 Talking on the phone - right

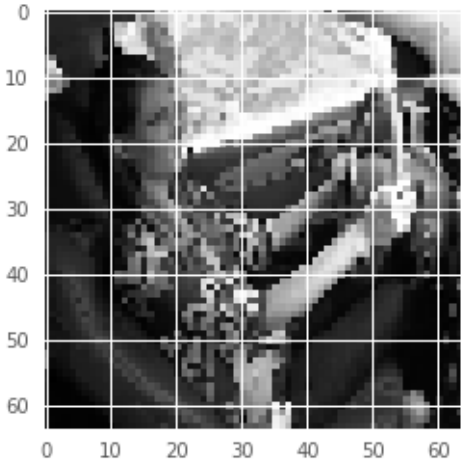


图 13 Safe driving

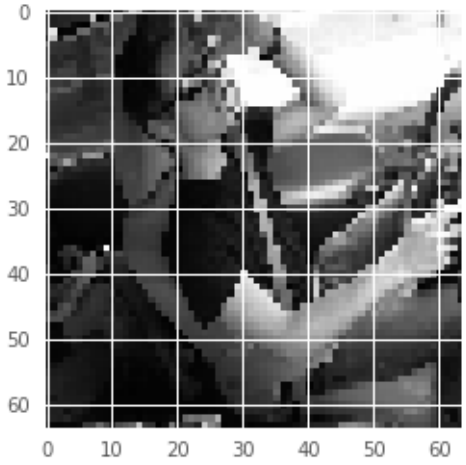


图 14 hair and makeup

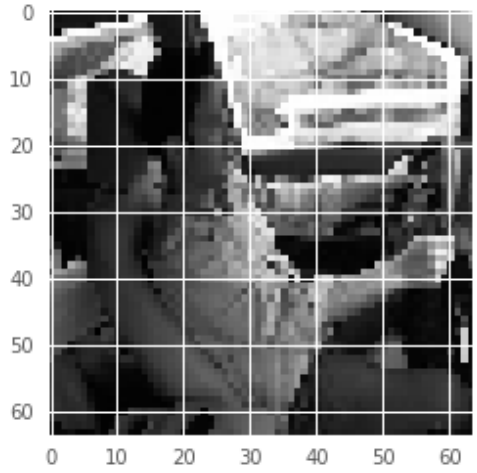


图 15 Safe driving

4.2.2 基于生成的大量数据

利用实际车内图片经过各种变化得到的图像，进行了更大规模的数据训练与测试，迭代次数缩小为 5 次，训练策略与少量数据时相同。

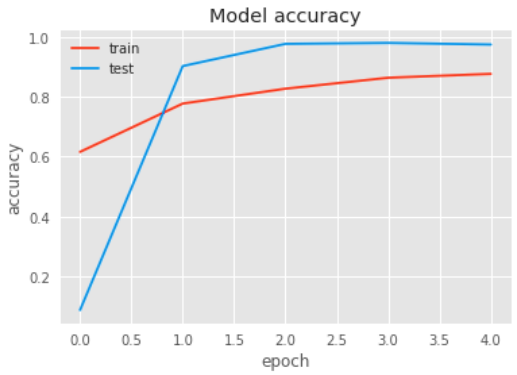


图 16 改进 Vanilla CNN 模型大数据训练准确率

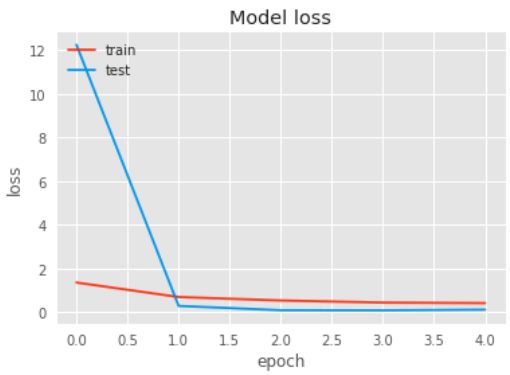


图 17 改进 Vanilla CNN 模型大数据训练损失率

可以看到模型在数据训练时的 loss 在不断下降，accuracy 在不断上升，符合数据训练预期。最终模型的准确率和损失率分别是 0.980 和 0.083，较之前模型有所下降，证明图像调整会对模型产生一定影响。

4.3 模型 2 的测试过程

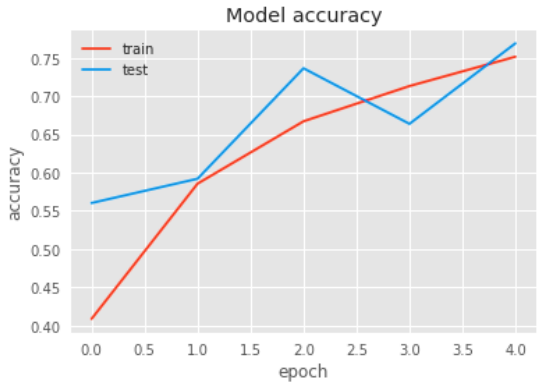


图 18 VGG16 模型准确率

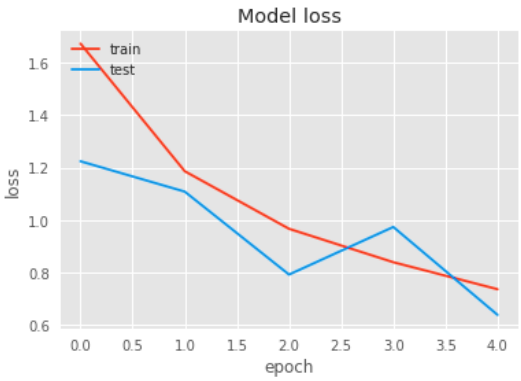


图 19 VGG16 模型损失率

从模型准确率上来说，VGG16 模型的训练效果比较明显，上升幅度比较大；虽然在损失率的测试中出现了不稳定性，但整体下降幅度也不错。本次测试迭代次数为 5 次，从曲线趋势上看，继续增加迭代次数应该可以进一步优化模型。

4.4 模型 1 与模型 2 的对比

模型 1 最终的 Test Accuracy 达到了 0.9938，Test Score 更是低至 0.0296；模型 2 的 Test Accuracy 为 0.7679，Test Score 为 0.6423。模型 2 的准确率与损失率都远劣于模型 1。

从结果上看，自主构建的三层 CNN 模型在这个项目上的契合度比 VGG16 的迁移模型更高。但从时间效率上来说，VGG16 的训练时间远小于改进 CNN 模型。

5. 部署

5.1 编写推理代码与配置文件

在 modelarts 上部署上线，需要按照要求把模型、推理代码、配置文件放在一起。将需要的文件放在 model 文件夹下，如下图所示：

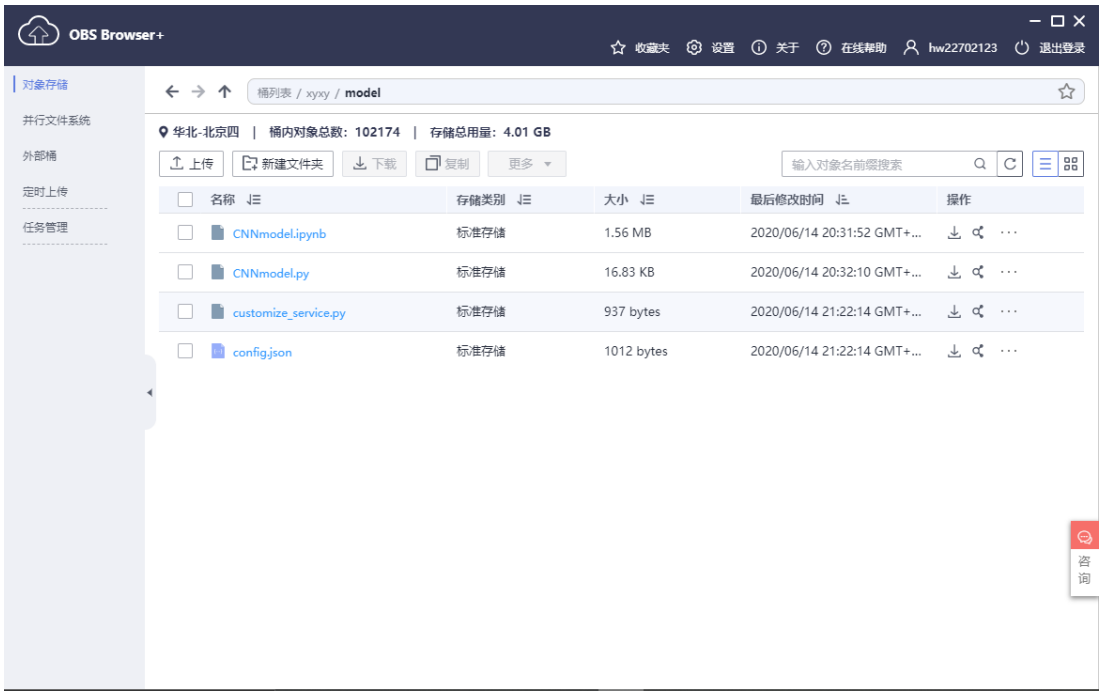


图 20　推理代码与配置文件

官方给出了使用 MNIST 数据集的一个推理代码，我们在它的基础上进行修改，主要是对输入的数据进行处理，使他适合模型的输入。

模型配置文件描述模型用途、模型计算框架、模型精度、推理代码依赖包以及模型对外 API 接口。根据自身需求进行修改。

5.2 正式部署

在 ModelArts 管理控制台，单击左侧导航栏中的“模型管理>模型”，进入“模型”页面，单击左上角“导入”。 在 “元模型来源”中，选择 “从 OBS 中选择”页签，然后在“选择元模型”选项中设置为训练作业中的 “model”文件夹上一级。

在“模型”页面，当模型状态变更为“正常”时，表示模型已导入成功。在对应版本所在行，操作列单击“部署>在线服务”，将模型部署为在线服务。

6. 总结

本实验先从最基础的 CNN 模型入手，对模型进一步增加层数，改进为三层 CNN 模型，测试效果得到了显著提高。随后对图像进行了一定的变换，增加模型的识别能力。

之后调用了 VGG16 模型，进行迁移学习。最终识别准确率良好，但存在进一步提升空间。

实验中提到的两种模型虽然都是较为基础的图像分类模型，但具有如下优点：

（1） 在阐述自己的评估指标时，尽量把所有重要的元素都包含进去，使评估更加准确。准确率和损失率都在模型中讨论。

（2） 陈述了一个独特的量化系统，将图像分类转换为了对每种类别的概率计算。

（3） 模型可以在现实中应用。通过根据具体情况改变初始参数，可以确定最佳结果。

（4） 我们提出了各种标准来比较不同的情况。因此可以根据这些标准进行整体比较。

（5） 我们的模型对基于敏感性分析的参数变化是相当稳健的，这意味着参数的轻微变化不会导致结果的显著变化。

模型仍可以提高和改进的地方在于：

（1） 在模型运行时，我们只能在本地进行运行，部署工作没有完成，操作受限。

（2） 在对图像进行训练时，对图像的处理训练有些马虎，这使得模型无法较好识别一些不同角度的图像

（3） 没有考虑模型的融合。使用更多模型进行综合考虑，能提高整体准确度。

下一步模型改进可以从多模型融合入手，可以使用 VGG16、ResNet50、InceptionV3 与 Xception 的融合模型，并使用 CAM 进行可视化。

7. 参考文献

[1]卷积神经网络研究综述[J].周飞燕,金林鹏,董军.计算机学报. 2017(06)

[2]杨林川. 基于深度神经网络的司机行为识别技术研究与实现[D].电子科技大学,2018.

[3]徐建君. 基于人脸特征的列车司机疲劳驾驶检测与识别系统研究[D].西南交通大学,2010.

[4]李俊俊,杨华民,张澍裕,李松江.基于神经网络融合的司机违规行为识别[J].计算机应用与软件,2018,35(12):222-227+319.

[5]基于改进卷积神经网络算法的研究与应用[D]. 王飞飞.南京邮电大学 2016

[6]基于多特征信息融合的违规驾驶行为检测方法研究[D].唐小淋.华南理工大学 2013

[7]基于机器视觉的违规驾驶行为检测研究[D].卓胜华.华南理工大学 2012

[8]基于自适应特征聚类的机车司机驾驶行为检测算法研究[A].周雯,吕晓军,程清波,刘小燕,强万福.2014 第九届中国智能交通年会大会论文集[C]. 2014

[9]徐丹,代勇,纪军红.基于卷积神经网络的驾驶人行为识别方法研究[J].中国安全科学学报,2019,29(10):12-17.

[10]基于贝叶斯结构方程模型的疲劳驾驶行为意图研究[J].邓院昌,史晨军.安全与环境学报. 2019(02)

[11]基于稀疏时空特征描述的驾驶者多种非安全驾驶行为识别[J].杜勇,王春明,崔金,李磊军,崔尧,郭培智.智能计算机与应用. 2018(06)

[12]基于个性化行为模型的驾驶疲劳识别方法[J].楚文慧,吴超仲,张晖,杨曼,李思瑶.中国安全科学报.2018(06)

[13]SF-CNN 在驾驶行为识别中的应用研究[J].王忠民,张瑶,衡霞.计算机工程与应用. 2018(11)

[14]褚晶辉,张姗,汤文豪,吕卫.基于导师-学生网络的驾驶行为识别方法[J].激光与光电子学进展,2020,57(06):211-218.

[15]Ask Your Neurons: A Deep Learning Approach to Visual Question Answering[J].Mateusz Malinowski,Marcus Rohrbach,Mario Fritz.International Journal of Computer Vision.2017 (1-3)

[16]田文洪,曾柯铭,莫中勤,吝博强.基于卷积神经网络的驾驶员不安全行为识别[J].电子科技大学学报,2019,48(03):381-387.

[17]郭佳伟.基于计算机视觉的驾驶员异常行为识别与预警[D].大连海事大学,2019.

[18]张德明.基于深度学习的驾驶行为识别算法研究[D].东南大学,2019.

[19] elninowang.基于深度学习驾驶员状态检测，不仅仅可以识别出疲劳驾驶，还能够识别出各种各样的状态[CP].(2017-10-03)[2020-6-13]. https://gitee.com/sunhao95/mlnd_distracted_driver_detection/tree/master

[20] Ismail CHAIDA.CNN to detect driver actions[CP].(2019-07-15)[2020-06-13]. https://www.kaggle.com/ismailchaida/cnn-to-detect-driver-actions?select=sample_submission.csv

[21] wisdom-bob. Based on Kaggle Distracted Driver Detection, to do some try.[CP].(2019-07-10)[2020-06-13]. https://github.com/wisdom-bob/distracted_driver_detection

[22]Praba Hridayami,I Ketut Gede Darma Putra,Kadek Suar Wibawa. Fish Species Recognition Using VGG16 Deep Convolutional Neural Network[J].2019,13(3).

[23]Sarfaraz Masood,Abhinav Rai,Aakash Aggarwal,M.N. Doja,Musheer Ahmad. Detecting distraction of drivers using Convolutional Neural Network[J]. Elsevier B.V.,2017.

[24]Shuai Zhao,XiangLei Zhu,Yingbo Li,Lu Zhang. Research on the Recognition of Car Models Based on Deep-Learning Networks[C].Proceedings of The 2nd Asia-Pacific Computer Science and Application Conference (CSAC 2017). 2017:669-674.

[25]Zichao Jiang,Zichao Jiang.A Novel Crop Weed Recognition Method Based on Transfer Learning from VGG16 Implemented by Keras[J].IOP Conference Series: Materials Science and Engineering,2019,677(3):032073 (8pp). DOI:10.1088/1757-899X/677/3/032073.

[26]Hossain, M. Shamim,Al-Hammadi, Muneer,Muhammad, Ghulam.Automatic Fruit Classification Using Deep Learning for Industrial Applications[J].IEEE transactions on industrial informatics,2019,15(2):1027-1034.

[27]Cibuk, Musa,Budak, Umit,Guo, Yanhui, et al.Efficient deep features selections and classification for flower species recognition[J].Measurement,2019,137:7-13.

[28]Li, Heyi,Tian, Yunke,Mueller, Klaus, et al.Beyond saliency: Understanding convolutional neural networks from saliency prediction on layer-wise relevance propagation[J].Image and vision computing,2019,83/84(Mar./Apr.):70-86.

8. 附录与代码

```
import os
from glob import glob
import random
import time
import tensorflow
import datetime
os.environ['KERAS_BACKEND'] = 'tensorflow'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
from tqdm import tqdm
import numpy as np
import pandas as pd
from IPython.display import FileLink
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
%matplotlib inline
from IPython.display import display, Image
import matplotlib.image as mpimg
import cv2
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_files
from keras.utils import np_utils
from sklearn.utils import shuffle
from sklearn.metrics import log_loss
from keras.models import Sequential, Model
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.applications.vgg16 import VGG16
current_path=os.getcwd()
print(current_path)
from modelarts.session import Session
session=Session()
session.download_data(bucket_path="/xyxy/input/",path="/home/ma-user/work")
# 导入司机信息，并显示前五行检查数据
dataset = pd.read_csv('input/driver_imgs_list.csv')
dataset.head(5)
# 对司机进行分类，统计不同司机数目
by_drivers = dataset.groupby('subject')
```

```
unique_drivers = by_drivers.groups.keys()
print(unique_drivers)
# 行为标签分为 10 类
NUMBER_CLASSES = 10

# Color type: 1 代表 grey, 3 代表 rgb
def get_cv2_image(path, img_rows, img_cols, color_type=3):
    if color_type == 1:
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    elif color_type == 3:
        img = cv2.imread(path, cv2.IMREAD_COLOR)
    # 切割尺寸
    img = cv2.resize(img, (img_rows, img_cols))
    return img

# 加载训练集
def load_train(img_rows, img_cols, color_type=3):
    start_time = time.time()
    train_images = []
    train_labels = []
    # 循环读取文件夹
    for classed in tqdm(range(NUMBER_CLASSES)):
        print('Loading directory c{}'.format(classed))
        files = glob(os.path.join('input', 'imgs','train', 'c' + str(classed),
'*.jpg'))
        for file in files:
            img = get_cv2_image(file, img_rows, img_cols, color_type)
            train_images.append(img)
            train_labels.append(classed)
    print("Data Loaded in {} second".format(time.time() - start_time))
    return train_images, train_labels

def read_and_normalize_train_data(img_rows, img_cols, color_type):
    X, labels = load_train(img_rows, img_cols, color_type)
    y = np_utils.to_categorical(labels, 10)
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    x_train = np.array(x_train,
dtype=np.uint8).reshape(-1,img_rows,img_cols,color_type)
    x_test = np.array(x_test,
dtype=np.uint8).reshape(-1,img_rows,img_cols,color_type)

    return x_train, x_test, y_train, y_test
```

```
# 加载测试集
def load_test(size=200000, img_rows=64, img_cols=64, color_type=3):
    path = os.path.join('input','imgs','test', '*.jpg')
    files = sorted(glob(path))
    X_test, X_test_id = [], []
    total = 0
    files_size = len(files)
    for file in tqdm(files):
        if total >= size or total >= files_size:
            break
        file_base = os.path.basename(file)
        img = get_cv2_image(file, img_rows, img_cols, color_type)
        X_test.append(img)
        X_test_id.append(file_base)
        total += 1
    return X_test, X_test_id

def read_and_normalize_sampled_test_data(size, img_rows, img_cols,
color_type=3):
    test_data, test_ids = load_test(size, img_rows, img_cols, color_type)

    test_data = np.array(test_data, dtype=np.uint8)
    test_data = test_data.reshape(-1,img_rows,img_cols,color_type)

    return test_data, test_ids

# 统一图片为 64x64 的灰度图形
img_rows = 64
img_cols = 64
color_type = 1

# 切割训练集和测试集
x_train, x_test, y_train, y_test = read_and_normalize_train_data(img_rows,
img_cols, color_type)
print('Train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')

# 随机抽取测试集中 200 张图片
nb_test_samples = 200
test_files, test_targets =
read_and_normalize_sampled_test_data(nb_test_samples, img_rows, img_cols,
color_type)
print('Test shape:', test_files.shape)
```

```
print(test_files.shape[0], 'Test samples')

# 输出统计信息
names = [item[17:19] for item in sorted(glob("input/imgs/train/*/"))]
test_files_size = len(np.array(glob(os.path.join('input', 'imgs','test',
'*.jpg'))))
x_train_size = len(x_train)
categories_size = len(names)
x_test_size = len(x_test)
print('There are %s total images.\n' % (test_files_size + x_train_size +
x_test_size))
print('There are %d training images.' % x_train_size)
print('There are %d total training categories.' % categories_size)
print('There are %d validation images.' % x_test_size)
print('There are %d test images.'% test_files_size)
# 指定输出图表大小
plt.figure(figsize = (5,5))
sns.set(style="whitegrid", context="notebook")
# 计算每个类别的数量
sns.countplot(x = 'classname', data = dataset , palette="Set3")
# 改变轴名称
plt.ylabel('Count')
plt.title('Different Numbers statistical behavior')

plt.show()

# 统计每个司机拥有的图片数量
drivers_id = pd.DataFrame((dataset['subject'].value_counts()).reset_index())
drivers_id.columns = ['driver_id', 'Counts']
drivers_id
activity_map = {'c0': 'Safe driving',
                'c1': 'Texting - right',
                'c2': 'Talking on the phone - right',
                'c3': 'Texting - left',
                'c4': 'Talking on the phone - left',
                'c5': 'Operating the radio',
                'c6': 'Drinking',
                'c7': 'Reaching behind',
                'c8': 'Hair and makeup',
                'c9': 'Talking to passenger'}
plt.figure(figsize = (12, 20))
image_count = 1
BASE_URL = 'input/imgs/train/'
for directory in os.listdir(BASE_URL):
```

```
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(5, 2, image_count)
                image_count += 1
                image = mpimg.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(activity_map[directory])
def create_submission(predictions, test_id, info):
    result = pd.DataFrame(predictions, columns=['c0', 'c1', 'c2', 'c3', 'c4',
'c5', 'c6', 'c7', 'c8', 'c9'])
    result.loc[:, 'img'] = pd.Series(test_id, index=result.index)

    now = datetime.datetime.now()

    if not os.path.isdir('kaggle_submissions'):
        os.mkdir('kaggle_submissions')

    suffix = "{}_{}".format(info,str(now.strftime("%Y-%m-%d-%H-%M")))
    sub_file = os.path.join('kaggle_submissions', 'submission_' + suffix +
'.csv')

    result.to_csv(sub_file, index=False)

    return sub_file

# 迭代次数设置为 10 次
batch_size = 40
nb_epoch = 10

!rm -f saved_models/weights_best_vanilla.hdf5

models_dir = "saved_models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

checkpointer =
ModelCheckpoint(filepath='saved_models/weights_best_vanilla.hdf5',
                                monitor='val_loss', mode='min',
                                verbose=1, save_best_only=True)
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
callbacks = [checkpointer, es]
```

```
def create_model_v1():
    # Vanilla CNN model
    model = Sequential()

    model.add(Conv2D(filters = 64, kernel_size = 3, padding='same', activation
= 'relu', input_shape=(img_rows, img_cols, color_type)))
    model.add(MaxPooling2D(pool_size = 2))

    model.add(Conv2D(filters = 128, padding='same', kernel_size = 3, activation
= 'relu'))
    model.add(MaxPooling2D(pool_size = 2))

    model.add(Conv2D(filters = 256, padding='same', kernel_size = 3, activation
= 'relu'))
    model.add(MaxPooling2D(pool_size = 2))

    model.add(Conv2D(filters = 512, padding='same', kernel_size = 3, activation
= 'relu'))
    model.add(MaxPooling2D(pool_size = 2))

    model.add(Dropout(0.5))

    model.add(Flatten())

    model.add(Dense(500, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation = 'softmax'))

    return model
model_v1 = create_model_v1()
model_v1.summary()

model_v1.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# 训练朴素 Vanilla CNN Model v1
history_v1 = model_v1.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          callbacks=callbacks,
          epochs=nb_epoch, batch_size=batch_size, verbose=1)

model_v1.load_weights('saved_models/weights_best_vanilla.hdf5')
```

```
def plot_train_history(history):
    # 整理训练过程中的精确度
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

    # 整理训练过程中的损失率
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

plot_train_history(history_v1)

def plot_test_class(model, test_files, image_number, color_type=1):
    img_brute = test_files[image_number]
    img_brute = cv2.resize(img_brute,(img_rows,img_cols))
    plt.imshow(img_brute, cmap='gray')

    new_img = img_brute.reshape(-1,img_rows,img_cols,color_type)

    y_prediction = model.predict(new_img, batch_size=batch_size, verbose=1)
    print('Y prediction: {}'.format(y_prediction))
    print('Predicted:
{}'.format(activity_map.get('c{}'.format(np.argmax(y_prediction)))))

    plt.show()

score = model_v1.evaluate(x_test, y_test, verbose=0)
print('Score: ', score)

!rm -f saved_models/weights_best_vanilla.hdf5

def create_model_v2():
    # 改进 Vanilla CNN 模型
    model = Sequential()
```

```
    ## CNN 1
    model.add(Conv2D(32,(3,3),activation='relu',input_shape=(img_rows,
img_cols, color_type)))
    model.add(BatchNormalization())
    model.add(Conv2D(32,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 2
    model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 3
    model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.5))

    ## Output
    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(128,activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(10,activation='softmax'))

    return model
model_v2 = create_model_v2()

model_v2.summary()

# 输出评价参数
model_v2.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])
# 训练模型
history_v2 = model_v2.fit(x_train, y_train,
```

```
          validation_data=(x_test, y_test),
          callbacks=callbacks,
          epochs=nb_epoch, batch_size=batch_size, verbose=1)
plot_train_history(history_v2)
model_v2.load_weights('saved_models/weights_best_vanilla.hdf5')
score = model_v2.evaluate(x_test, y_test, verbose=1)
print('Score: ', score)

y_pred = model_v2.predict(x_test, batch_size=batch_size, verbose=1)
score = log_loss(y_test, y_pred)
print('Score log loss:', score)
y_test1=np.argmax(y_test,axis=1).reshape(-1)
y_true=np.array(y_test1)
y_pred=model_v2.predict_classes(x_test)
pd.crosstab(y_true,y_pred,rownames=['true'],colnames=['predict'])
y_test1=y_test1.tolist()
a=pd.crosstab(np.array(y_test1),y_pred,rownames=['Labels'],colnames=['predi
ct'])
plt.style.use('ggplot') # 使用'ggplot'风格美化显示的图表
df=pd.DataFrame(a)
ax=sns.heatmap(df,annot=True,cmap="Reds",linewidths=0.2,linecolor='G')
ax.set_ylim(10.0, 0)
plt.show()
plt.tight_layout()
plot_test_class(model_v1, test_files, 20)
plot_test_class(model_v2, test_files, 20)
plot_test_class(model_v1, test_files, 60)
plot_test_class(model_v2, test_files, 60)
plot_test_class(model_v1, test_files, 90)
plot_test_class(model_v2, test_files, 90)
plot_test_class(model_v1, test_files, 150)
plot_test_class(model_v2, test_files, 150)
!rm -f saved_models/weights_best_vanilla.hdf5
# 对训练图片进行一定的调整
train_datagen = ImageDataGenerator(rescale = 1.0/255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True,
                                   validation_split = 0.2)

test_datagen = ImageDataGenerator(rescale=1.0/ 255, validation_split = 0.2)
nb_train_samples = x_train.shape[0]
nb_validation_samples = x_test.shape[0]
print(nb_train_samples)
```

```
print(nb_validation_samples)
training_generator = train_datagen.flow(x_train, y_train,
batch_size=batch_size)
validation_generator = test_datagen.flow(x_test, y_test,
batch_size=batch_size)
checkpoint = ModelCheckpoint('saved_models/weights_best_vanilla.hdf5',
monitor='val_acc', verbose=1, save_best_only=True, mode='max')

history_v3 = model_v2.fit_generator(training_generator,
                      steps_per_epoch = nb_train_samples // batch_size,
                      epochs = 5,
                      callbacks=[es, checkpoint],
                      verbose = 1,
                      validation_data = validation_generator,
                      validation_steps = nb_validation_samples // batch_size)

model_v2.load_weights('saved_models/weights_best_vanilla.hdf5')
plot_train_history(history_v3)
score = model_v2.evaluate_generator(validation_generator,
nb_validation_samples // batch_size)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
y_test2=np.argmax(y_test,axis=1).reshape(-1)
y_true=np.array(y_test2)
y_pred=model_v2.predict_classes(x_test)
pd.crosstab(y_true,y_pred,rownames=['true'],colnames=['predict'])
plot_test_class(model_v2, test_files, 90)

!rm -f saved_models/weights_best_vanilla.hdf5

def vgg_std16_model(img_rows, img_cols, color_type=3):
    nb_classes = 10
    # 迁移模型
    vgg16_model = VGG16(weights="imagenet", include_top=False)

    for layer in vgg16_model.layers:
        layer.trainable = False

    x = vgg16_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(nb_classes, activation = 'softmax')(x)

    model = Model(input = vgg16_model.input, output = predictions)
```

```
    return model

# 导入 VGG16 模型
print("Loading network...")
model_vgg16 = vgg_std16_model(img_rows, img_cols)

model_vgg16.summary()

model_vgg16.compile(loss='categorical_crossentropy',
                        optimizer='rmsprop',
                        metrics=['accuracy'])
training_generator = train_datagen.flow_from_directory('input/imgs/train',
                                                       target_size = (img_rows,
img_cols),
                                                       batch_size = batch_size,
                                                       shuffle=True,
                                                       class_mode='categorical',
subset="training")

validation_generator = test_datagen.flow_from_directory('input/imgs/train',
                                                         target_size = (img_rows,
img_cols),
                                                         batch_size = batch_size,
                                                         shuffle=False,
                                                         class_mode='categorical',
subset="validation")
nb_train_samples = 17943
nb_validation_samples = 4481

!rm -f saved_models/weights_best_vgg16.hdf5

# 训练模型
checkpoint = ModelCheckpoint('saved_models/weights_best_vgg16.hdf5',
monitor='val_acc', verbose=1, save_best_only=True, mode='max')
history_v4 = model_vgg16.fit_generator(training_generator,
                        steps_per_epoch = nb_train_samples // batch_size,
                        epochs = 5,
                        callbacks=[es, checkpoint],
                        verbose = 1,
                        class_weight='auto',
                        validation_data = validation_generator,
                        validation_steps = nb_validation_samples // batch_size)
model_vgg16.load_weights('saved_models/weights_best_vgg16.hdf5')
```

```
plot_train_history(history_v4)
score = model_vgg16.evaluate_generator(validation_generator,
nb_validation_samples // batch_size, verbose = 1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```