
机器学习纳米学位

侦测走神司机 刘凯欣 优达学城

2018-4-4

目录

- 1 问题的定义0
 - 1.1 项目概述.....0
 - 1.2 问题陈述.....0
 - 1.3 评价指标.....2
- 2 分析2
 - 2.1 数据的探索.....2
 - 2.2 探索性可视化.....4
 - 2.3 算法和技术.....6
 - 2.4 基准模型7
- 3 方法8
 - 3.1 数据预处理.....8
 - 3.2 执行过程9
 - 3.3 完善10
- 4 结果18
 - 4.1 模型的评价与验证.....18
 - 4.2 合理性分析.....20
- 5 项目结论21
 - 5.1 结果可视化.....21
 - 5.2 对项目的思考22
 - 5.3 需要作出的改进23

1 问题的定义

1.1 项目概述

绝大多数交通事故的产生都是由于司机的分心驾驶, 根据美国政府官方网站的调查显示, 2015 年有 3477 人因为司机的分心驾驶死亡, 391000 人为此受伤^[1], 在科技迅速发展的互联网时代, 驾驶人员更容易做出接打电话、聊天、玩手机等影响自己和乘客安全的行为。

国外关于司机分心驾驶的研究较多, 有基于生理测量的机器学习方法检测驾驶员注意力不集中的研究^[2], 也有使用卷积神经网络对司机行为进行分类的研究^[1], 还有通过 Kinect 算法提取眼部特征、胳膊位置和头方向等特征实现分心驾驶检测的研究^[3], 相比而言, 国内相关研究较少, 关于在司机驾驶过程中状态检测的研究大多数为“检测司机疲劳驾驶”, 通过人脸识别技术检测人眼, 进而分析司机是否为疲劳驾驶^[4], 鲜有包含司机聊天、接打电话、玩手机等分心驾驶的研究文献。

随着数据量的增大、数据采集技术和神经网络技术的发展, 通过采集司机驾驶过程中的照片, 并对司机的动作进行分类是完全可能的。卷积神经网络在图像识别中表现很好, 在街道级别的图片中识别牌号达到 99.8% 的正确率。本项目使用 StateFarm^[5]提供的开源数据集, 其中含有大量图片和类别标签, 因此可以通过卷积神经网络, 训练一个能准确分类司机动作的卷积神经网络模型, 在司机分心驾驶时进行提醒, 避免悲剧的产生。

1.2 问题陈述

项目解决的主要问题是根据司机驾驶过程中的截图, 判定司机当前所处的状态。

将司机当前状态划分为以下十类: (1) 安全驾驶; (2) 右手打字; (3) 右手打电话; (4) 左手打字; (5) 左手打电话; (6) 调收音机; (7) 喝饮料; (8) 拿后面的东西; (9) 整理头发和化妆; (10) 和其他乘客说话。

将司机驾驶过程中的截图作为输入, 输出该输入图片分别属于十个分类的概率。概率最高者即为当前司机的状态。

项目使用 TensorFlow1.5.0 版本的高级 API——keras 作为深度学习框架解决问题, 希望

能准确识别司机驾驶过程中的状态。

项目用的卷积神经网络基础模型是 2014 年提出的经典模型 VGG16^[6]，相比于 GoogLeNet^[7]和 ResNet^[8]等网络，其深度较浅，训练快，需要的时间成本也低，虽然表现稍差，但是可以接受。具体的解决思路为：

- (1) 读取数据集：将所有图片路径信息和标签信息读入内存。由于机器内存限制，不能将所有图片一次性读入内存，然后训练，因此采用先将所有图片的路径读入内存，在训练时再根据图片路径分批读取相应的图片；
- (2) 拆分数据集：将训练数据根据司机拆为训练集与验证集，选取一个司机的所有图片作为验证集，其余司机的所有图片作为训练集。由于项目使用的数据集是从驾驶视频中截取而来，因此前后两帧的画面应该是相似设置相同的，如果两幅图分别在训练集与验证集中，则等于将验证集透露给了模型，会导致表现评价失去意义，因此训练集与验证集的拆分不能简单地在所有图片上用混洗然后按比例拆分；
- (3) 混洗训练集与验证集数据：拆分完数据集后，需要混洗数据。
- (4) 预处理输入数据：VGG16 使用图片大小为 224x224 的 RGB 图片，在 TensorFlow 中，需要将图片处理成 224x224x3 的图片，使用临近像素重构图片，而非中心裁剪；
- (5) 数据增强：为了防止图片过拟合及提升模型表现，每批次的图片在输入模型前还要增加水平变换、垂直变换还有移动填充的图片；
- (6) 构建模型，设置网络结构、评价指标、优化器等；
- (7) 读取测试数据集：同（1），将测试数据的图片路径信息一次性读入内存，然后根据批次大小分批读取图片，进行预测；
- (8) 预测测试数据；
- (9) 将测试数据组织成 Kaggle 大赛需要的提交格式，提交到官网，获得模型表现。

多次重复上述过程，不断进行实验，来对比修改模型，并做相应的记录，获得更优的模型表现。

1.3 评价指标

使用多类对数损失 (*multi-class logarithm loss*) 作为评估指标, 给出图片属于十个类的可能性, 计算公式为

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

其中 N 是测试集中图片的数量, M 是类别标签的数量, 如果 i 属于 j 则 $y_{ij} = 1$, 否则 $y_{ij} = 0$, p_{ij} 是 i 属于 j 的概率。

之所以使用 $mlogloss$ 而非 $accuracy$, 是因为 $accuracy$ 相同的情况下, $mlogloss$ 可以更进一步比较两个模型, 比如 A 和 B 两个模型的 $accuracy$ 相同, 预测同一张类型为 0 的图片, 都预测正确, 但是模型 A 认为是 0 类的概率比模型 B 认为是 0 类的概率大, 则 A 模型的 $mlogloss$ 就会小, A 模型比 B 模型更好, 因此使用 $mlogloss$ 的评判粒度更小, 因此使用 $mlogloss$ 作为评估指标。






2 分析

2.1 数据的探索

项目使用 StateFarm^[5]提供的数据集, 由以下三部分组成:

(1) 22424 张训练图片, 在 train 文件夹中, 按类别存储, 每个类一个文件夹, 包括安全驾驶、左右手打字、左右手接电话等 10 个类别, 图片格式个 RGB, 图中司机包含各个种族, 具体描述如表 2.1 所示;

表 2.1 数据集基本情况

类别	类别名称	图片总数 (张)	单张图片宽度 (像素)	单张图片高度 (像素)	详细描述	示例图
c0	安全驾驶	2489	640	480	司机双手握方向盘，目视前方，且无其他类别的行为	
c1	右手打字	2267	640	480	司机右手拿有手机	
c2	右手打电话	2317	640	480	司机右手拿手机且放于右耳	
c3	左手打字	2346	640	480	司机左手拿有手机	
c4	左手打电话	2326	640	480	司机左手拿手机且放于左耳	
c5	调收音机	2312	640	480	司机调整收音机	
c6	喝饮料	2325	640	480	司机喝饮料	
c7	拿后面东西	2002	640	480	司机转向后座或单手伸到后面拿东西	
c8	整理头发和化妆	1911	640	480	司机看着镜子整理头发或者化妆	
c9	和其他乘客说话	2129	640	480	司机未目视前方且与其他乘客说话	

(2) 79726 张测试集图片，在 test 文件夹中，乱序存储，大小像素类别信息等于训练集相同；

(3) 一个名为 driver_imgs_list.csv 的文件，该文件记录了训练集图片的司机 id、类别和图片名，该图片可以根据类别和名称在 train 文件夹中找到，文件具体描述如表 2.2 所示。

表 2.2 标签数据情况表

编号	司机 id	图片数（张）	约占总数比例	编号	司机 id	图片数（张）	约占总数比例
1	p002	725	0.032331	14	p045	724	0.032287
2	p012	823	0.036702	15	p047	835	0.037237
3	p014	876	0.039065	16	p049	1011	0.045086
4	p015	875	0.039021	17	p050	790	0.035230
5	p016	1078	0.048073	18	p051	920	0.041027
6	p021	1237	0.055164	19	p052	740	0.033000
7	p022	1233	0.054986	20	p056	794	0.035408
8	p024	1226	0.054674	21	p061	809	0.036077
9	p026	1196	0.053336	22	p064	820	0.036568
10	p035	848	0.037817	23	p066	1034	0.046111
11	p039	651	0.029031	24	p072	346	0.015430
12	p041	605	0.026980	25	p075	814	0.036300
13	p042	591	0.026356	26	p081	823	0.036702

2.2 探索性可视化

数据集图片部分截图如图 2.1，可以看出，数据集图片分辨率、拍摄角度、人物位置都很清晰统一，如红色框内图片，很明显正常驾驶。但是有少数图片包含较多可能性，带来分辨困难，比如绿色框内图片，可能是正常驾驶，也可能是与旁人说话，标签定义为正常驾驶；蓝色框内图片，可能是正常驾驶（不排除有人习惯于单手驾驶的可能），也可能是与旁人说话，标签定义为与旁人说话，这些可能属于多各类的图片，可能给模型识别带来些困难。

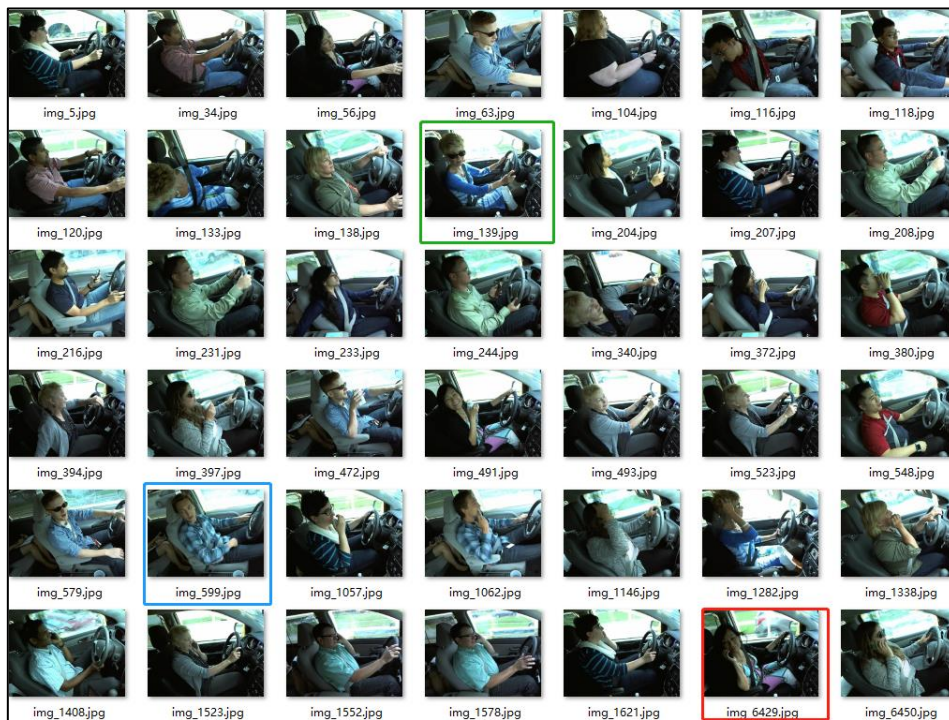


图 2.1 数据集样图

每个类别的图片数目统计和每个司机的图片数如图 2.2，可以看出各个类别图片数目比较均匀，每个司机的图片数据不太均匀，但也不影响。每个司机的每个类型图片数目统计如图 2.3，橙色为图片数目，蓝色为司机分隔线。

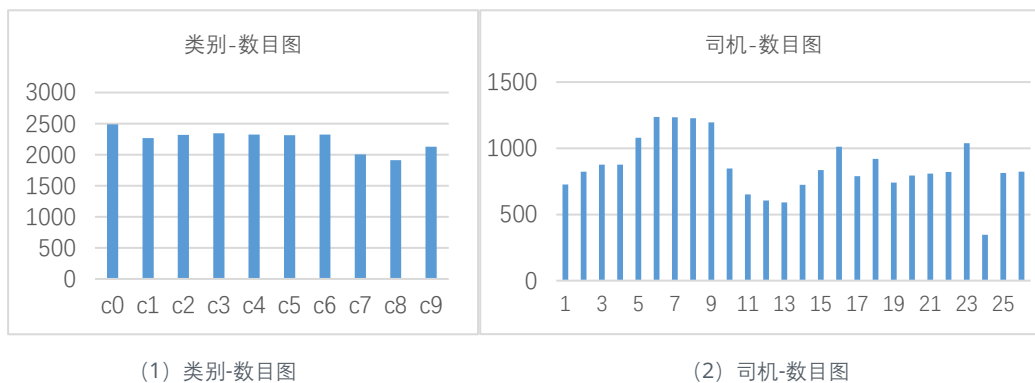


图 2.2 类别/司机数据分布图

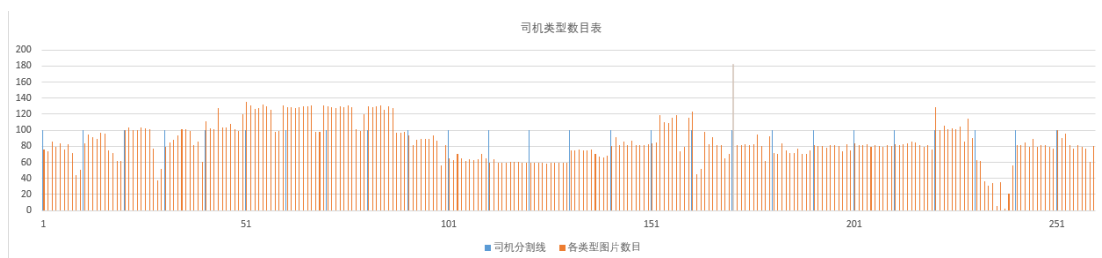


图 2.3 司机各类图片数目分布图

项目选择前 25 个司机（表 2.2 中编号为 1~25）为训练集，最后一个司机（编号 26）为验证集。从图 2.3 的最后一段蓝线分割的各类图片情况中可以看到第 26 个司机总图片数目适中，各类型图片分布也适中，所以作为验证集是合理的。

2.3 算法和技术

2.3.1 卷积神经网络

卷积神经网络是近年发展起来，并引起广泛重视的一种高效识别方法，受生物自然视觉认知机制启发而来^[9]。

1959 年，Hubel & Wiesel 发现，动物视觉皮层细胞负责检测光学信号。受此启发，1980 年 Kunihiko Fukushima 提出了 CNN 的前身——neocognitron。

20 世纪 90 年代，LeCun et al.等人发表论文，确立了 CNN 的现代结构，后来又对其进行完善。他们设计了一种多层的人工神经网络，取名叫做 LeNet-5，可以对手写数字做分类。和其他神经网络一样，LeNet-5 也能使用 backpropagation 算法训练。

CNN 能够得出原始图像的有效表征，这使得 CNN 能够直接从原始像素中，经过极少的预处理，识别视觉上面的规律。然而，由于当时缺乏大规模训练数据，计算机的计算能力也跟不上，LeNet-5 对于复杂问题的处理结果并不理想。

2006 年起，人们设计了很多方法，想要克服难以训练深度 CNN 的困难。其中，最著名的是 Krizhevsky et al.提出了一个经典的 CNN 结构，并在图像识别任务上取得了重大突破。其方法的整体框架叫做 AlexNet，与 LeNet-5 类似，但要更加深一些。

AlexNet 取得成功后，研究人员又提出了其他的完善方法，其中最著名的要数 ZFNet, VGGNet, GoogleNet 和 ResNet 这四种。从结构看，CNN 发展的一个方向就是层数变得更多，ILSVRC 2015 冠军 ResNet 是 AlexNet 的 20 多倍，是 VGGNet 的 8 倍多。通过增加深度，网络便能够利用增加的非线性得出目标函数的近似结构，同时得出更好的特性表征。但是，这样做同时也增加了网络的整体复杂程度，使网络变得难以优化，很容易过拟合。

2.3.2 技术

项目使用 TensorFlow1.5.0 版本的高级 API——Keras 作为深度学习框架解决问题，也可以使用 Keras。

Keras 由纯 Python 编写而成并基于 Tensorflow、Theano 以及 CNTK 后端。Keras 为支持快速实验而生，能够把想法迅速转换为结果，用户友好，具有以下特点^[11]：

- 简易和快速的原型设计（keras 具有高度模块化，极简，和可扩充特性）
- 支持 CNN 和 RNN，或二者的结合
- 无缝 CPU 和 GPU 切换

2.4 基准模型

将 Kaggle 官网上能找到的最高分 kernel“statefar_facepalm_fork”^[14]作为基准模型，该模型的 Private Score 是 1.80188，Public Score 是 1.55292，大约排名 670 左右。

该 kernel 解决问题的大体思路为：

- (1) 将司机信息从 driver_imgs_list.csv 中读取到字典里，每一条字典数据 key 为图片名 (xxxx.jpg)，value 为司机 id；
- (2) 将训练集图片处理成大小为 40x50 的灰度图，每个像素除以 255，放入训练集 X_train，将图片标签放入 y_train，再将与 X_train 的每张图对应的司机 id 找出来放入 driver_id 中，将所有司机的 id 去重放入 unique_drivers；
- (3) 将 X_train 的 shape 变为(图片总数，颜色通道，行，列)，即 (22424, 1, 40, 50)，并除以 255 进行归一化，y_train 变为 10 个类别的 binary class matrices；测试集数据同样做次处理，得到 test_data 和 test_id(图片名称)；
- (4) 将训练数据中的第 1~25 位司机数据划分为训练集 (id 为 p049 的司机除外)，第 26 位司机数据划分为验证集；
- (5) 将输入数据输入图 2.4 中模型，进行拟合。该模型是一个两层卷积模型，有 8 个卷积核，大小为 4x4，滑动步长为 4，激活函数是‘relu’，两个卷积层后连接 2x2 的最大池化层，进行 0.6 的 dropout，然后连接含有 128 个神经元的全连接层，进行 0.5 的 dropout，最后通过 softmax 分成 10 各类，网络优化器是 sgd，初始学习率 0.1，动量 0.4，不衰减。

```
def create_model_v1(img_rows, img_cols, color_type=1):
    nb_classes = 10
    # number of convolutional filters to use
    nb_filters = 8
    # size of pooling area for max pooling
    nb_pool = 2
    # convolution kernel size
    nb_conv = 4
    model = Sequential()
    model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
                           border_mode='valid',
                           input_shape=(color_type, img_rows, img_cols)))
    model.add(Activation('relu'))
    model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
    model.add(Dropout(0.6))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(nb_classes))
    model.add(Activation('softmax'))

    sgd = SGD(lr=0.1, decay=0, momentum=0.4, nesterov=False)
    model.compile(loss='categorical_crossentropy', optimizer=sgd)
    return model
```

图 2.4 基准模型核心代码

3 方法

3.1 数据预处理

本文的基本思路是使用在 ImageNet 上预训练的 VGG16 网络，因此数据预处理方法为

- (1) 重定义图片大小, 原图大小为 680x480 的 RGB 图片, 处理为 VGG16 网络需要的 224x224 的 RGB 图片, 使用临近像素模式进行大小重定义 (resize), 而非中心裁剪 (crop);
- (2) 使用 TensorFlow 官方的图片预处理方法将每个像素除以 127 然后减 1。核心代码如下:

```
def get_im_cv2(paths, img_rows, img_cols, color_type=1):
    """
    参数:
        paths: 要读取的图片路径列表
        img_rows: 图片行
        img_cols: 图片列
        color_type: 图片颜色通道
    返回:
        imgs: 图片数组
    """
    # Load as grayscale
    imgs = []
    for path in paths:
        if color_type == 1:
            img = cv2.imread(path, 0)
        elif color_type == 3:
            img = cv2.imread(path)
        # Reduce size
        resized = cv2.resize(img, (img_cols, img_rows))
        resized = resized.astype('float32')
        resized /= 127.5
        resized -= 1.

        imgs.append(resized)

    return np.array(imgs).reshape(len(paths), img_rows, img_cols, color_type)
```

图 3.1 数据预处理核心代码

3.2 执行过程

数据集的划分为前 25 个司机的数据为训练集，最后一个司机的数据为验证集。在论文《Detecting Distraction of drivers using convolutional neural network》^[1]中提到 ImageNet 上的预训练权重初始化网络会产生比较好的结果，只是该论文的训练集与验证集划分不科学，因此得出的结果不可信，所以本次也做了使用预训练权重的实验，在预处理后输入网络，网络直接用的 keras 的 applications 中的 VGG16 模型，为了减少网络运行的时间成本，先去除两个 4096 的全连接层并将其换成全局平均池化层，然后分成 10 类输出，优化器为 SGD，初始学习率 $1e-5$ ，衰减 $1e-6$ ，动量 0.9，如果连续 10 代在验证集上的损失函数未减少，则提前终止训练。网络结构图如下。

网络结构
input
vgg16 (no top, use pre-trained weights)
GAP
output

图 3.4 网络结构图

代码在运行了 15 代后提前停止训练,平均每代 8 分钟,准确率及损失函数如下图所示。

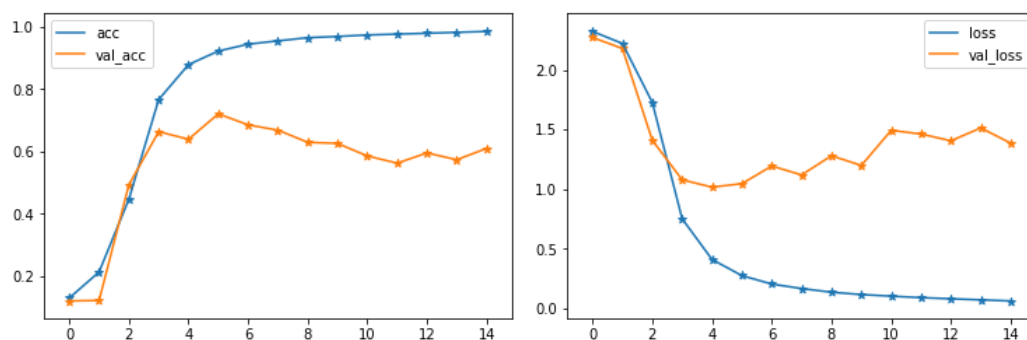


图 3.4 预训练权重的 VGG16 验证集准确率及损失函数

在训练集上，准确率从 0.1 平稳上升至 1.0 附近收敛，损失函数也从 2.3 平稳下降至 0.2 左右收敛，在验证集上，准确率从 0.1 平稳上升至 0.7 左右开始波动下降，损失函数从 2.3 平稳下降到 1.0 左右开始波动上升，模型开始对训练集过拟合。

最优模型是第 5 代模型，称为 VGG16-PT，使用第 5 代模型预测并提交至 Kaggle，在训练集上的 loss、在验证集上的 loss 以及提交至 Kaggle 的 loss 如下表所示。

表 3.2 VGG16-PT 最优结果

名称	loss	val_loss	kaggle_public_score
VGG16-PT	0.4084	1.0157	1.00401

3.3 完善

网络的主要问题是过拟合严重, 所以第一想到的完善方式是增加较小的全连接层并增加 dropout, 在增加全连接层前, 先对数据进行增强, 详细的实验方案为: (1) 根据数据集特点重新选择数据增强的方式, 数据集没有上下和左右翻转, 只有小幅旋转和平移, 还有因为摄像头距离司机的距离不同产生的缩放效果, 因此数据增强方式需要调整; (2) 从两个 512 的小全连接层开始实验, 根据表现进行调整全连接层的深度或宽度; (3) 全连接层必然会带来精度的提升, 但同时也会加重过拟合, 因此在两个全连接层后分别加上 0.5 的 dropout, 根据模型表现进行调整 dropout 值。

(1) 数据增强

先做数据增强的改动, 将数据增强方式改为每张图片随机记性保留原图、平移、缩放、旋转操作, 核心代码如下图。

```
def img_augmentation(X_train, y_train):
    """
    对传入的图片进行数据增强, 每个图片有4种随机操作: 不增强、移动、缩放、旋转
    参数:
        X_train: 要增强的图片
        y_train: 要增强的图片对应的标签
    返回:
        X_train: 处理后的图片
        y_train: 处理后的图片标签
    """
    for i in range(X_train.shape[0]):
        rand = random.randint(1, 4)
        if rand == 2:
            X_train[i] = tf.keras.preprocessing.image.random_shift(X_train[i], 0.2, 0.2)
        elif rand == 3:
            X_train[i] = tf.keras.preprocessing.image.random_zoom(X_train[i], zoom_range=(0.3, 0.3))
        else:
            X_train[i] = tf.keras.preprocessing.image.random_rotation(X_train[i], 10)
    return X_train, y_train
```

图 3.8 修改后的数据增强代码

(2) 增加全连接层

为网络添加两个小全连接层, 神经元数均为 512, 激活函数均为 relu, 全连接层后的 dropout 均为 0.5, 最后通过 softmax 全连接层分成 10 个类。网络结构如下图。

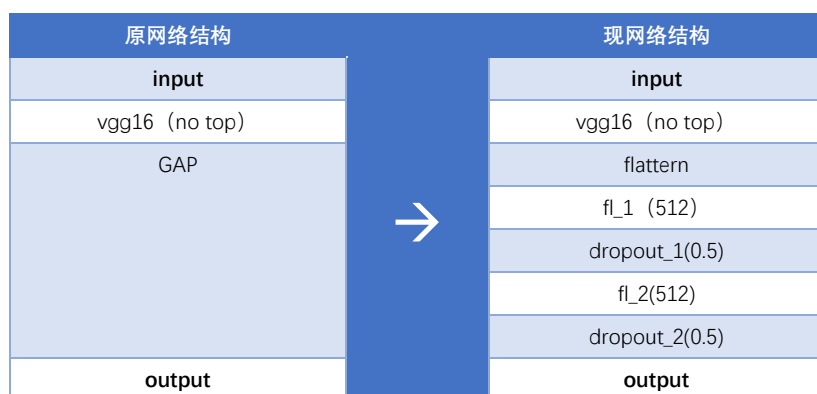


图 3.9 模型修改前后结构图

模型的优化器为 SGD，初始学习率为 $1e-5$ ，衰减 $1e-6$ ，动量 0.9，如果连续 10 代在验证集上损失函数不减小则提前停止训练。

模型运行了 27 代后提前停止训练，在训练集和验证集上的准确率及损失函数如图所示。

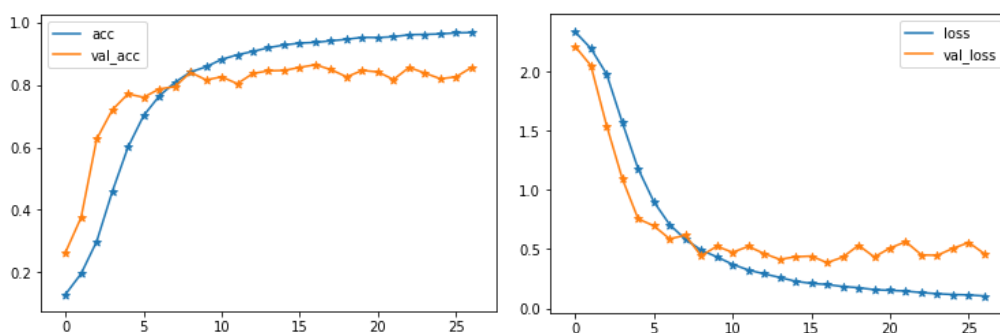


图 3.10 模型验证集的准确率及损失函数

曲线比 VGG16-PT 要好，训练集上准确率平稳上升至收敛，损失函数平稳下降至收敛，验证集上准确率平稳上升，最后在 0.8 左右微小波动，损失函数在 0.5 左右微小波动，就准确率和损失函数的差值减少，说明过拟合有所减少。

最优的模型是第 17 代，称作 VGG16-PT-0.46928，在训练集上的 loss、验证集上的 loss 以及 Kaggle 上的 loss 如下表，相比 VGG16-PT 有了明显的减小，与 VGG16-PT 的对比如下图。

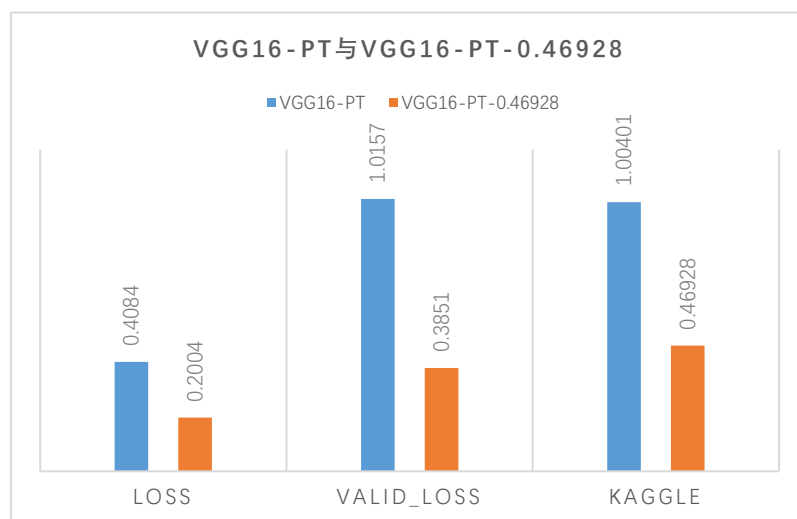


图 3.10 VGG16-PT 与 VGG16-PT-0.46928 损失函数对比

(3) 修改 dropout

两个全连接层的初始 dropout 均为 0.5，先调整第一个 dropout，根据上图表的情况，不妨先试一个较大的 dropout，再根据情况拟合情况进行调整，所以先将第一个 dropout 设置为 0.8，第二个依然是 0.5，优化器、初始学习率、学习率衰减、动量都不变，连续运行 5 代在验证集上损失函数不降低则提前停止训练，模型结构如图。

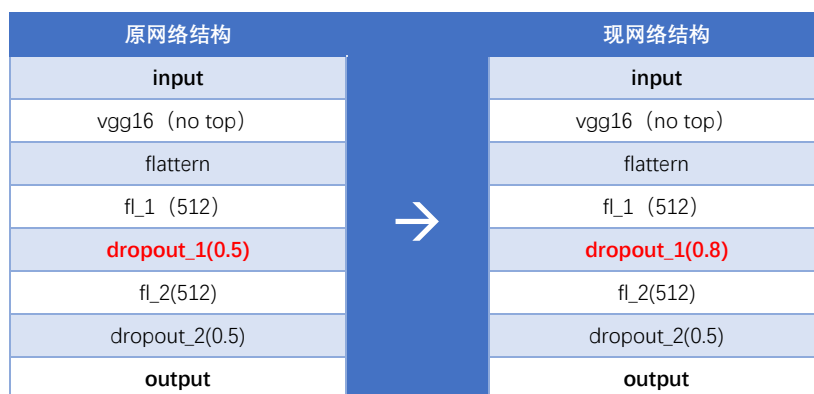


图 3.11 模型修改前后结构图对比

代码运行在第 34 代提前终止，在训练集上的准确率和损失函数、验证集上的准确率损失函数如图所示。

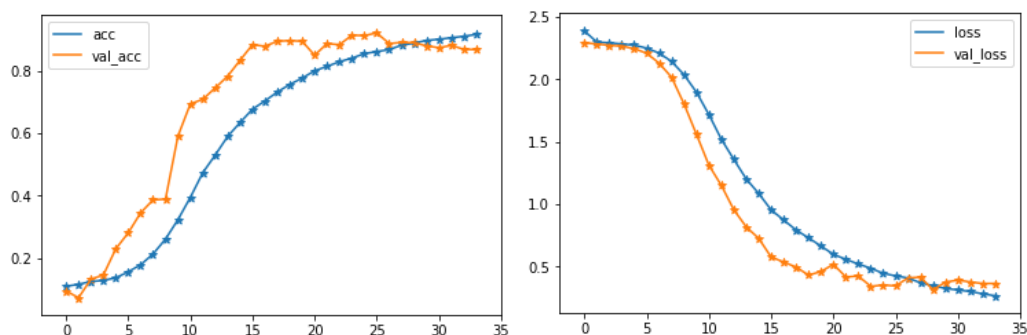


图 3.12 模型验证集的准确率及损失函数

训练集和验证集的过拟合情况明显好转，精度也有所提高，准确率都平稳上升在 0.85 左右收敛，损失函数平稳下降，在 0.3 左右收敛，验证集曲线在后期基本与训练集重合，前期甚至表现超过训练集，比起 VGG16-PT-0.46928，后期的波动较小。

模型最优的一代为第 29 代，与 VGG16-PT-0.46928 相比，在训练集上的损失函数上升，验证集上的损失函数下降，Kaggle 上的损失函数下降，情况如图。

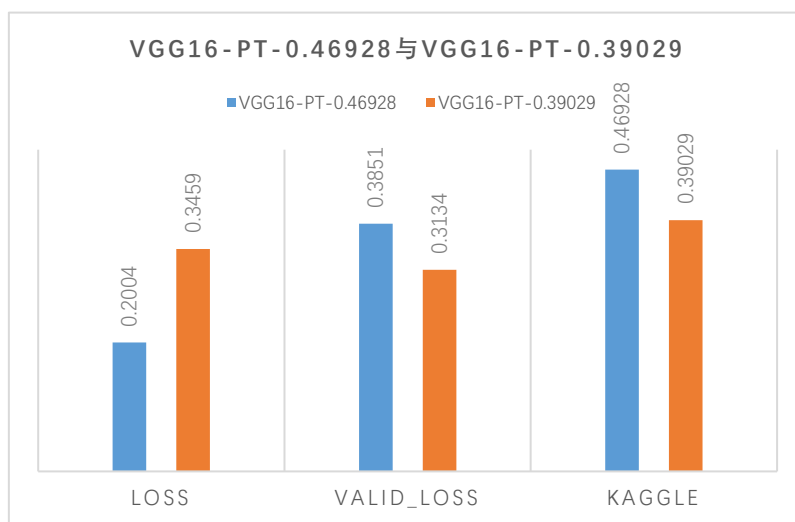


图 3.13 VGG16-PT-0.46928 与 VGG16-PT-0.39029 损失函数对比

上图训练集上的损失函数比起上一个网络模型要高，猜想原因是第一个 dropout 设为 0.8 后丢失较多信息，导致精度降低，所以将 dropout 改为第一层进行 0.7 的 dropout，第二层进行 0.8 的 dropout，模型的结构如下：

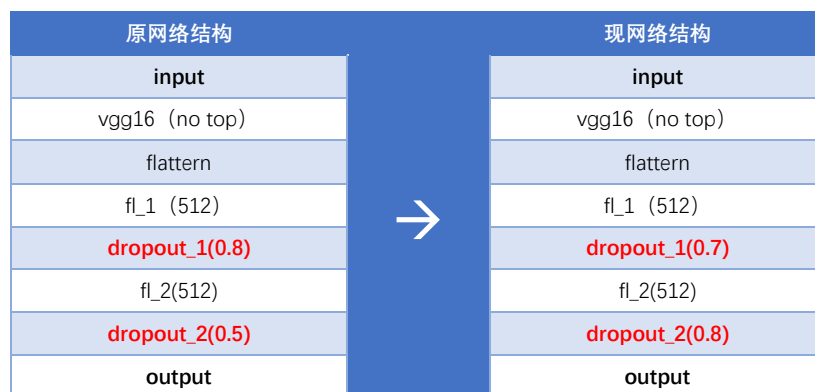


图 3.14 模型修改前后结构图对比

模型的准确率和损失函数对比图如下：

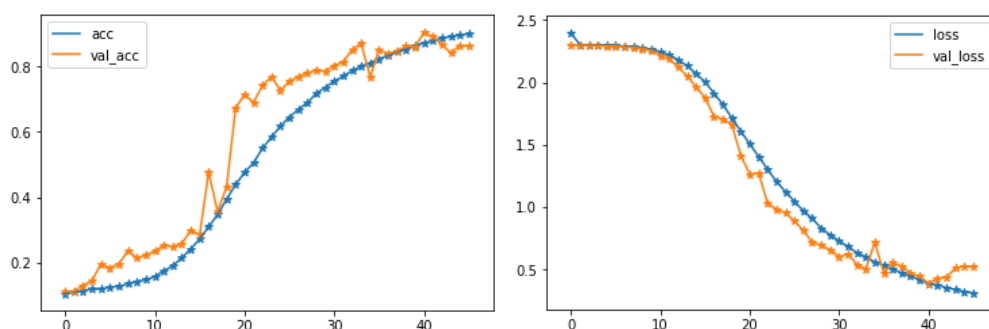


图 3.15 模型验证集的准确率及损失函数

产生的结果在 kaggle 上的得分为 0.38438，称为模型 VGG16-PT-0.38438，与上一个模型（VGG16-PT-0.39029）的对比图如下。

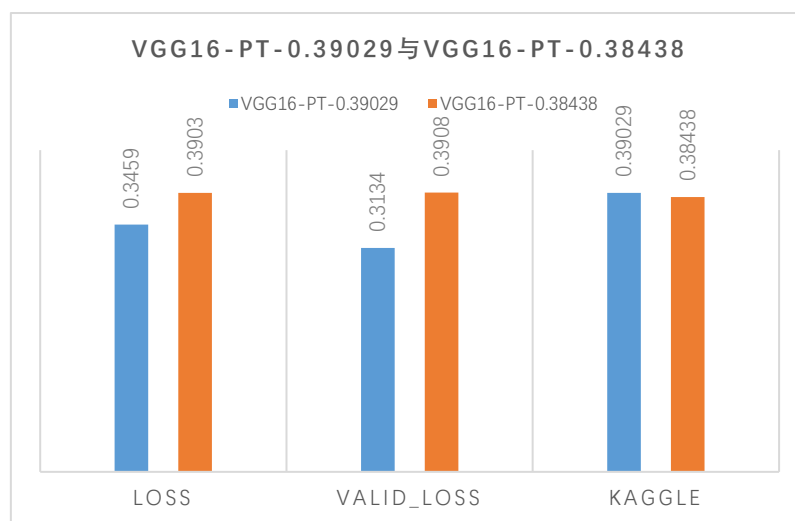


图 3.16 VGG16-PT-0.39029 与 VGG16-PT-0.38438 损失函数对比

新模型在训练集、验证集和测试集上表现都差不多，过拟合现象解决，但是可以看出已经到达精度的瓶颈，后续又陆续进行了两个全连接层增加至 1024、两个全连接层增加至 2048、两个全连接层增加至 4096、两个全连接层一个为 1024 一个为 2048、两个全连接层用 1x1

的卷积层加 1 个 GAP 层、将图片大小从 224x224x3 变为 256x256x3 的实验，表现均未有显著提高，在 (0.37, 0.45) 间波动，说明已经到该模型的极限。

(4) 模型融合

模型融合主要采用了在决策层面的融合——将多个模型的预测结果融合。整个融合实验分为三大部分，先分别得到前两部分的最优结果，最后将这两部分的最优结果融合，得到更好的表现，流程图如下图。

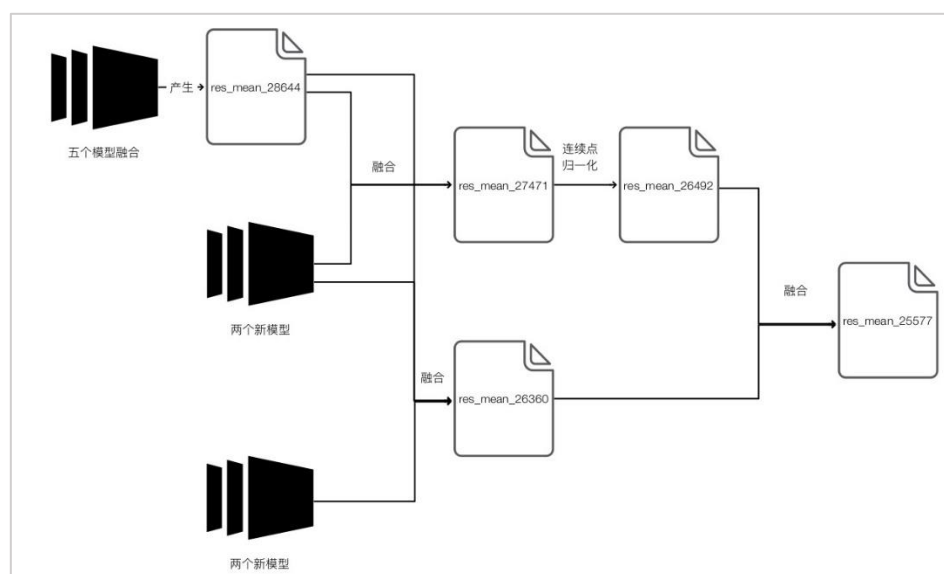


图 3.17 模型融合流程图

第一部分实验使用一系列上文提到的基于 VGG16 微调的模型, 前 25 个司机为训练集, 最后一个司机(id 为“p081”)为验证集, 使用在验证集上损失函数为 0.3908、0.3070、0.3509、0.3134 和 0.4213 的五个模型, 它们的混淆矩阵如下图所示, 红色代表差 (精确率</80%), 黄色代表适中 (80%≤精确率≤/90%), 绿色代表优秀 (精确率>/90%), 从混淆矩阵可以看出不同模型在各个类的预测上刚好互补, 如果以召回率为准赋予权重, 然后在测试集上测试并对算数平均, 可以得到 **0.29155** 的得分, 如果以**精确率**为准赋予权重, 可以得到 **0.28644** 的得分, 因此选择以精确率为准赋予权重的结果, 称为 res_mean_28644。

模型3908——kaggle0.38438 (224x224x3)													
		实际											
		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	精确率	
预测	c0	92	2	0	0	0	0	1	0	1	0	0.95	
	c1	0	88	8	0	0	0	0	0	0	3	0.88	
	c2	0	0	51	0	0	0	0	0	0	0	1	
	c3	0	0	1	81	0	0	0	0	1	0	0.97	
	c4	0	0	0	1	77	0	0	0	8	0	0.89	
	c5	0	0	0	0	0	80	0	0	0	0	1	
	c6	0	0	0	0	0	1	78	1	0	0	0.975	
	c7	0	0	0	0	0	0	0	76	0	3	0.96	
	c8	0	0	0	0	0	0	0	0	51	4	0.92	
召回率	c9	8	0	36	0	0	0	0	0	0	70	0.61	
		0.9	0.977778	0.53125	0.987805	1	0.987654	0.987342	0.987013	0.836066	0.875		
模型3070——kaggle0.37466 (256x256x3)													
		实际											
		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	精确率	
预测	c0	78	1	0	0	0	0	0	0	1	0	0.975	
	c1	0	76	0	0	0	0	0	0	1	0	0.98	
	c2	0	3	92	0	0	0	1	0	0	0	0.95	
	c3	0	0	0	79	0	0	0	0	0	0	1	
	c4	0	1	0	3	77	0	10	0	7	0	0.785	
	c5	0	0	0	0	0	81	0	0	0	0	1	
	c6	0	0	0	0	0	0	68	1	0	0	0.98	
	c7	0	0	0	0	0	0	0	75	0	0	1	
	c8	0	0	0	0	0	0	0	1	52	1	0.96	
召回率	c9	22	9	4	0	0	0	0	0	0	79	0.69	
		0.8	0.844444	0.958333	0.963415	1	1	0.860759	0.974026	0.852459	0.9875		

模型3509——kaggle0.35989 (256x256x3)													
		实际											
		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	精确率	
预测	c0	90	4	2	0	0	0	0	0	1	0	0.92	
	c1	0	76	0	0	0	0	0	0	0	0	1	
	c2	0	0	74	0	0	0	0	0	0	0	1	
	c3	0	0	0	79	0	0	0	0	0	0	1	
	c4	0	0	0	3	77	0	6	0	8	0	0.81	
	c5	0	0	0	0	0	81	0	0	0	0	1	
	c6	0	0	0	0	0	0	73	2	0	0	0.97	
	c7	0	0	0	0	0	0	0	75	0	0	1	
	c8	0	1	0	0	0	0	0	0	52	3	0.94	
召回率	c9	10	9	20	0	0	0	0	0	0	77	0.66	
		0.9	0.844444	0.770833	0.963415	1	1	0.924051	0.974026	0.852459	0.9625		
模型3134——kaggle0.39029 (224x224x3)													
		实际											
		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	精确率	
预测	c0	73	2	0	0	0	0	0	0	0	0	0.97	
	c1	0	88	10	0	0	0	0	0	0	6	0.89	
	c2	0	0	75	0	0	0	1	0	0	0	0.98	
	c3	0	0	10	79	2	0	0	0	1	1	0.84	
	c4	0	0	0	3	75	0	9	0	7	0	0.79	
	c5	0	0	0	0	0	81	0	0	0	0	1	
	c6	0	0	0	0	0	0	69	3	0	0	0.95	
	c7	0	0	0	0	0	0	0	73	0	2	0.97	
	c8	0	0	0	0	0	0	0	1	53	5	0.89	
召回率	c9	27	0	1	0	0	0	0	0	0	66	0.7	
		0.7	0.977778	0.78125	0.963415	0.974026	1	0.873418	0.948052	0.868852	0.825		

模型4213——kaggle0.41709 (320x320x3)												
		实际										
		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	精确率
预测	c0	45	0	0	0	0	0	0	0	0	0	1
	c1	1	84	0	0	0	3	0	0	0	5	0.93
	c2	0	5	88	0	0	0	4	1	0	0	0.89
	c3	0	0	0	80	0	1	2	0	0	0	0.96
	c4	0	0	0	2	77	0	0	0	3	0	0.93
	c5	0	0	0	0	0	77	0	0	0	0	1
	c6	0	0	0	0	0	0	73	0	0	0	1
	c7	0	0	0	0	0	0	0	76	0	0	1
	c8	0	1	8	0	0	0	0	0	58	41	0.58
c9	54	0	0	0	0	0	0	0	0	34	0.38	
召回率		0.5	0.933333	0.916667	0.97561	1	0.950617	0.924051	0.987013	0.95082	0.425	

图 3.18 模型混淆矩阵

对五个模型进行进一步观察，发现所有模型在召回率上看起来互补，但是在精确率上，对 c9 类普遍较低，因此认为如果有一个 c9 上表现很好的模型进行融合，应该会得到更好的表现。在模型中找到了两个在 c9 上表现较好的模型，分别为模型 3729（在验证集上损失函数为 0.3729）和模型 4210（在验证集上损失函数为 0.4210）。将模型 3729、模型 4210 和 res_mean_28644 做几何平均数，得到了 **0.27471** 的得分，称做 res_mean_27471。

将 res_mean_27471 数据进行分析，发现有的数据呈“扁平化”，如下图蓝线所示。

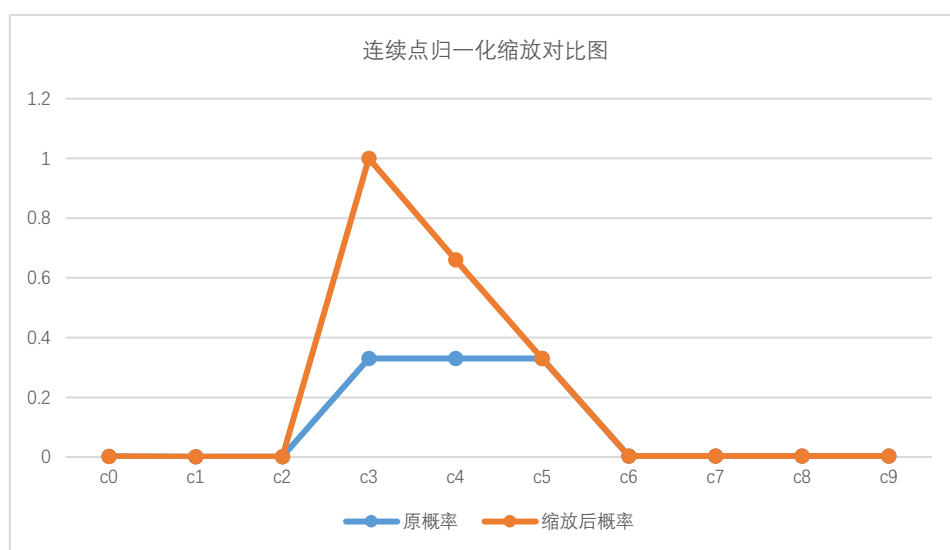


图 3.19 连续点归一化缩放对比图

即对同一类的预测有“不是很明确”的情况，比如有 0.3301 的概率为 a 类，有 0.3302 的概率为 b 类，有 0.3303 的概率为 c 类，按照 kaggle 的评价指标计算方式，0.3303 的可能性会折合成 $0.3303 / (0.3301 + 0.3302 + 0.3303) = 33\%$ ，但是如果将这三个数据进行 0 到 1 之间的归一化缩放，即将 0.3301、0.3302、0.3303 依次变为 0.33、0.66、1，如上图 b，则 0.3303 的可能性会变成 $1 / (0.33 + 0.66 + 1) = 50\%$ ，不论模型预测正确与否，猜测的可能性都会被放大，由于模型的正确率大概在 90% 以上，所以只有 10% 的猜测是错误的，进行 0~1 的缩放不仅能

让正确的更肯定，还能加大“认为是错误的”概率，使得到的损失函数也不会太大，如果正确了，得到的损失函数将会更小，经过归一化缩放后，res_mean_27471 得到了 **0.26492** 的得分，成为 res_mean_26492。算法的简单描述为：

遍历 csv 中每行数据：

 寻找这一行的峰值（即概率最大值）；

 如果峰值附近存在大小在[峰值，峰值*0.6]的点，则放入连续点集中（注：0.6 为最终调参得到的结果，依次尝试了从 0.5~0.9）；

 对 n 个连续点从大到小排序；

 按照排序将其缩放至[0, 1]间，最大值为 1，最小值为 1/n，其余点依次为 $1/n * i$ ，i 是点的排名；

第二部分是分别使用两个基于 VGG16、一个基于 VGG19、一个基于 Xception 和一个基于 ResNet50 微调的五个模型，每个随机选择 2 个司机作为验证集，其余 24 个司机作为训练集，更换验证集司机的好处在于增加模型之间的差异性。第一部分实验的最大缺陷在于 c9 类预测不太好，且基于 VGG16 的模型较多，学习虽然有差异但是不够大，所以第二部分融合了第一部分的模型以及两个差异较大的模型——VGG 19 和 Xception，Xception 在 c9 类的预测上表现较好，VGG19 是与 Xception 差异最大的一个模型，除了 c9 类，其余都很好，尤其是 c2~c7 类，基本全部正确。将预测结果取几何平均数，得到 0.26360 的得分，称为 res_mean_26360。

最后将 res_mean_26492 和 res_mean_26360 进行几何平均，得到了 **0.25577**，终于进入了 kaggle 的 top10%。

submission_2018-04-01-22-00.csv 18 hours ago by Happy Kai 两个2.6的平均	0.26531	0.25577	<input type="checkbox"/>
--	---------	---------	--------------------------

图 3.20 kaggle 得分截图

4 结果

4.1 模型的评价与验证

模型的最终 Kaggle 得分（测试集上的多分类损失函数）为 **0.25577**，已经超过基准模型，刚好到达 kaggle 的前 top10% (0.25634)。由于最终的得分是由模型融合得到的，除了 kaggle 的最终得分外难以找到合适又权威的评价标准，因此本章的结果评价与验证使用最好的单个模型——模型 VGG16-PT-0.38438，该模型在 kaggle 的得分为 0.38438。

(1) 准确性

为了更好地表示模型 VGG16-PT-0.38438 对各个类的预测情况，绘制了模型在验证集上预测情况的混淆矩阵，验证集共有 823 张图，混淆矩阵如下表，红色越深代表错误数目越多，可以看到，模型对总是把 c2（右手打电话）当做 c9（和后面乘客说话），初次之外都识别较好，平均精确率为 91.55%，平均召回率为 90.89%。

表 4.1 混淆矩阵

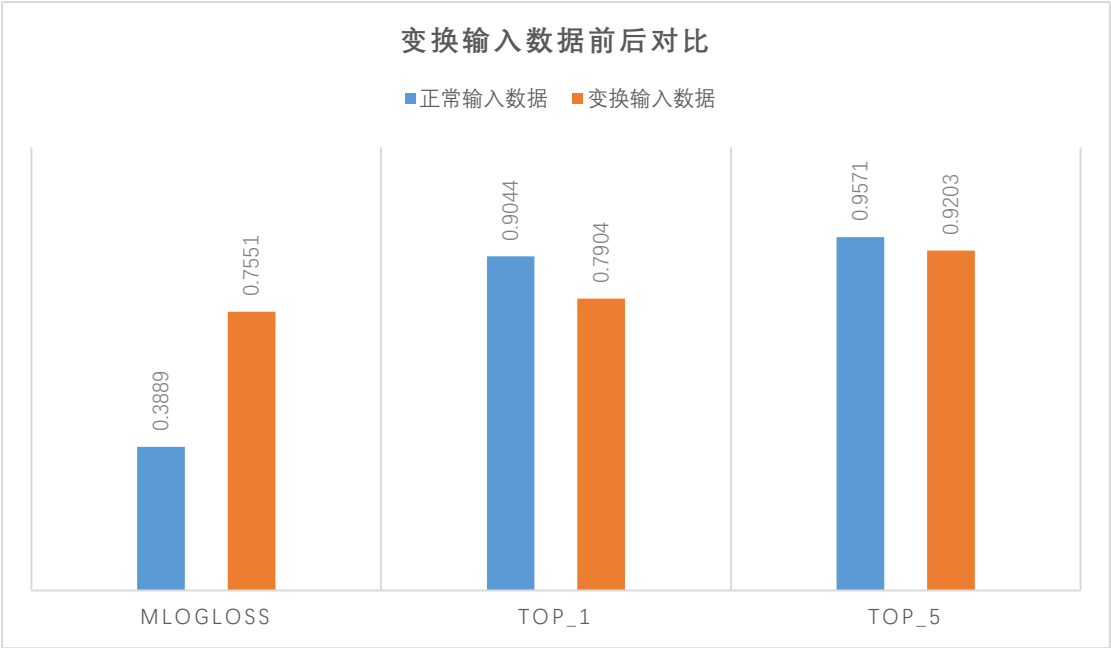
		预测										召回率
		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	
实际	c0	92	0	0	0	0	0	0	0	0	8	92.00%
	c1	2	88	0	0	0	0	0	0	0	0	88.89%
	c2	0	8	51	1	0	0	0	0	0	36	53.12%
	c3	0	0	0	81	1	0	0	0	0	0	98.78%
	c4	0	0	0	0	77	0	0	0	0	0	100.00%
	c5	0	0	0	0	0	80	1	0	0	0	98.76%
	c6	1	0	0	0	0	0	78	0	0	0	98.73%
	c7	0	0	0	0	0	0	1	76	0	0	98.70%
	c8	1	0	0	1	8	0	0	0	51	0	83.60%
	c9	0	3	0	0	0	0	0	3	4	70	87.50%
精确率		95%	88%	100%	97%	89%	100%	97%	96%	92%	61%	

(2) 鲁棒性

使用验证集的图片做模鲁棒性的评估，先使用正常的输入数据在验证集上预测，记录得到的 mlogloss（多分类损失函数）、top1 准确率和 top5 准确率，然后随机对验证集输入

数据进行垂直翻转、水平翻转、移动等变换，再用模型测试，记录得到的 mlogloss（多分类损失函数）、top1 准确率和 top5 准确率，输入数据前后变换结果对比见表 8，损失函数升了原来的 50%，top1 准确率下降了 10%，top5 准确率下降了 3%，说明模型的鲁棒性还不是很好。

表 4.2 变换输入数据前后对比



4.2 合理性分析

基准模型在 Kaggle 中的得分为 1.55292，模型 VGG16-PT-0.38438 在 Kaggle 中的得分为 0.38438，比基准模型表现更好。

使用基准模型提供的代码进行实验，可能代码提供的预处理等信息不全，得到的 Kaggle 得分为 2.30301，运行 50 代，如果 val_loss 连续 5 代不提升则提前终止训练，基准模型运行的结果如图 12，可以看出，val_loss 收敛很慢，在 2.297 附近呈现收敛趋势。

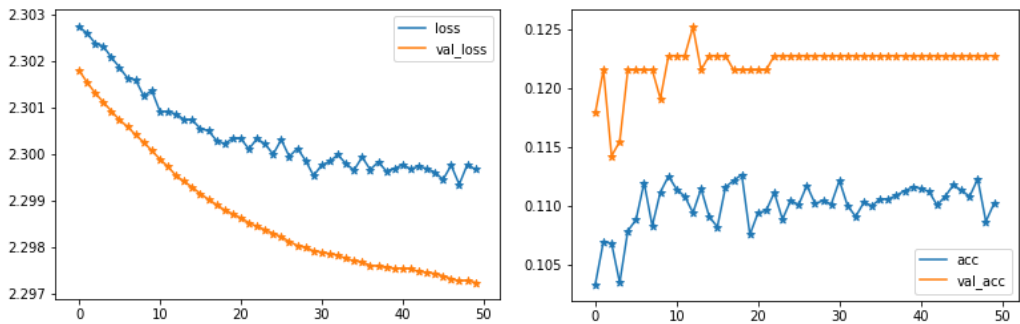


图 4.1 基准模型表现

与本文模型的各方面结果对比如图 13。基准模型的表现虽然不如本文模型，但是鲁棒性却优于本文模型，且网络训练速度非常快，是本文模型的 1/4。

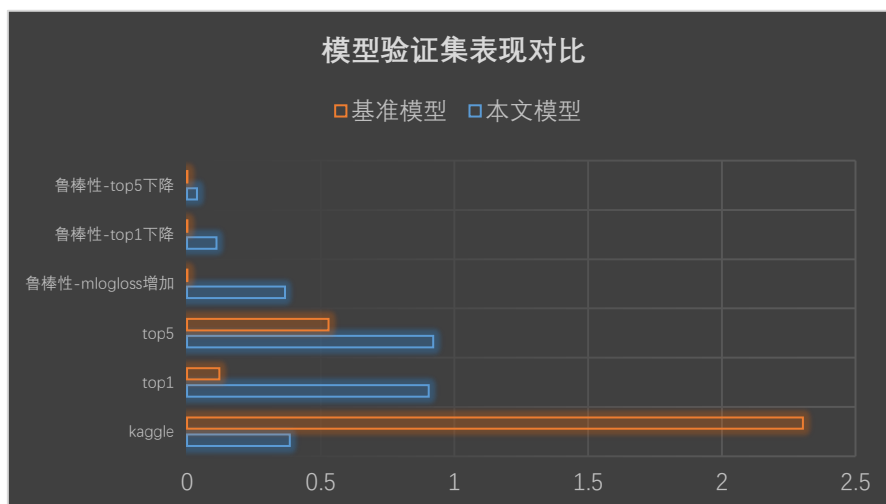


图 4.2 模型对比

5 项目结论

5.1 结果可视化

项目使用 Class Activation Mapping 方法进行可视化，该方法在《Learning Deep Features for Discriminative Localization》被详细介绍，主要原理是使用全局平均池化层（Global Average Pooling, GAP）代替全连接层，虽然精度有所减少，但是可以通过提取卷积层的特征结合全连接层的权重生成热成像图来对分类进行定位。

由于模型 VGG16-PT-0.38438 没有 GAP 层，并且除了最后一个 softmax 连接层外还有两个 512 的全连接层，所以可视化的实验只能采用 VGG16-PT-0.38438 的卷积层权重，通过一个 GAP，直接连分 10 类的 softmax 全连接层，并且在训练时不改变卷积层权重，通过这样的设置来勉强可视化卷积神经网络看到了什么，下图为 10 个类的正确预测的热成像图。

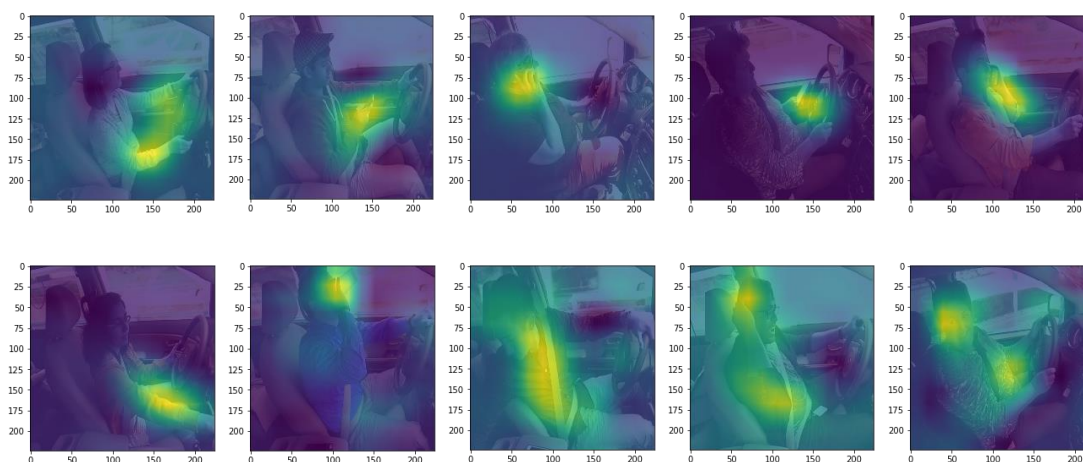


图 5.1 正确预测的十个热成像图

通过上面的热成像图，可以看出网络看的特征还是很正确的，但是可能不是“太肯定”，比如除了 c3~c5，其余图片都整体偏黄绿色，说明概率相差不是很大。但是个人认为这样的结果可视化是不准确的，但是也能反映出卷积层的特征提取的还不错比如向后拿东西（c7）和与旁人说话（c9），都比上一版提交的特征要好。

5.2 对项目的思考

本文详细地记录了项目的整个流程，从第一版改造没有预训练权重的 VGG16，到第二版改造使用预训练权重的 VGG16，到第三版使用 VGG16、VGG19、Xception 和 ResNet50 等模型进行结果融合，经历了将近 2 个月，损失函数从 0.86 降到 0.25，kaggle 提交了近 300 次。从刚开始不会分批读取数据，到会使用预训练模型，从使用预训练模型到学会简单的模型融合，整个过程收获很大。

项目比较有意思的地方在于很多处理细节以前没有注意到，比如以前划分训练集与验证集直接 shuffle，而这个项目提醒我需要考虑数据集的情况；网络上很多参考代码基于 caffe，输入向量与 TensorFlow 有所差别，这个项目告诉了我不同框架的一些基本差别；在模型可视化方面，通过 CAM 的学习知道了神经网络看到的是什么，这也是很大的收获；数据增强你一定要根据样本集的特点进行增强，而不是随意增强；根据模型的曲线猜测模型需要做哪方面的改进，每次猜测得到印证的时候很兴奋；之前只知道要分验证集和训练集，还要进行数据混洗的重要性不是很清楚，现在深有体会；可以通过分析不同模型的混淆矩阵知道模型在哪一方面表现较好，进行融合，可以得到很大的提升；模型融合很有意思，很难想象一群

表现不怎么样的模型聚合在一起能得到极好的表现；单个模型的性能始终受限，想得到很好的表现一定要进行模型融合。这个项目的每一次完善都让人很兴奋，收获很大。

项目比较难的地方在于（1）训练时可以分批训练，但是训练前需要把图片都加载到内存中，数据规模较大，机器内存不足，找了很多方法终于找到现将数据分批读到内存然后再从内存中取数据到 GPU 分批训练，模型才可以正常进入调优阶段；（2）在网络模型的优化过程中，一开始走偏了，一路暴力调参，调整全连接层、学习率、dropout……一遍一遍测试，但是表现提升的微乎其微，时间成本还很高，这个过程很绝望。后来将方向转向数据增强，模型表现才有所提高。最后通过分析模型结果发现模型的主要问题，定位于识别 c0、c5、c9 的准确度上，这三类有时候人类识别都很困难，是一个比较有趣的挑战；（3）有两个星期陷入了“单一验证集”的死局，一直认为验证集只能是那个始终没见过的司机，无论怎样进行模型融合，优化融合结果，都无法得到好的表现，后来经过 Reviewer 的提醒恍然大悟，我可以不同模型有不同的验证集，这样可以加大模型间的差异性，最后通过这种方式艰难地达到了最低要求。

5.3 需要作出的改进

模型目前处于刚刚达到“最低要求”的阶段，考虑到马上就要毕业截止，所以先提交这一版本，后续还将继续进行优化实验，争取进入前十名。后续的改进思路如下：

（1）在数据增强上做些改进，通过训练集图片拼接（比如分别使用同一类的两张图的左边和右边，产生新的图片），增加训练集样本。

（2）学习并实验 stacking 等模型融合方式，短期内看的不是很明白（把一个模型对训练集的 5 份预测拼成一个新的训练集，给同一层的下一个模型学习，这样不就相当于把错误样本给了下一个模型吗，这样能学好？还是我哪里理解错了）。

（3）可视化方式还局限于 CAM，CAM 只能由卷积层、GAP 和分类的全连接层组成，像没有用 GAP 的卷积神经网络想进行可视化，就需要别的方法，对 keras 提取已经训练好的模型的不同层并进行特征提取还不是很熟，最近也找了很多资料，但是它们提供的方法都不能实验成功，有向无环图总是连接失败，还需要静下心来好好研读 API，如果 Reviewer 对可视化及 keras 有向无环图有什么好的学习资料，感谢提供。

（4）模型进行结果融合的部分，每个模型的权重占比，可能能够通过模糊数学的方法科学

地给出，这里也需要进一步研究。

参考文献

- [1] MASOOD S, RAI A, AGGARWAL A, et al. Detecting Distraction of drivers using Convolutional Neural Network[J]. Pattern Recognition Letters (2018), 2017.
- [2] SAHAYADHAS A, SUNDARAJ K, MURUGAPPAN M, et al. A physiological measures-based method for detecting inattention in drivers using machine learning approach[J]. Biocybernetics and Biomedical Engineering, 2015,35(3):198-205.
- [3] CRAYE C, KARRAY F. Driver distraction detection and recognition using RGB-D sensor[J]. arXivJournal, 2015.
- [4] 赵雪竹. 基于AdaBoost算法的驾驶员疲劳检测[D]., 2010.
- [5] STATEFARM. State Farm Distracted Drivers Dataset[EB/OL]. (2017-06-15)[6.15]. <https://www.kaggle.com/c/state-farm-distracted-driver-detection>.
- [6] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.
- [7] CHRISTIANSZEGEDY. Going deeper with convolutions[J]. arXivJournal, 2014.
- [8] HE K. Deep Residual Learning for Image Recognition[J]. arXivJournal, 2015.
- [9] 百 度 百 科 . 卷 积 神 经 网 络 [EB/OL]. [2017]. <https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C/17541100?fr=aladdin>.
- [10] Keras中文文档[EB/OL]. <http://keras-cn.readthedocs.io/en/latest/>.
- [11] ZVEROLOFF. statefarm_facepalm_fork[EB/OL]. <https://www.kaggle.com/zveroloff/statefarm-facepalm-fork>.