机器学习纳米学位

走神司机检测

行者周

ZNBJTU@126.COM

2018-05-03

1 问题定义

1.1 项目概述

有过开车或者乘车经验的人通常会遇到一种情况就是，在一个交通路口，交通灯变为绿色但是我们前面的车并没有往前走，在后边司机的鸣笛提醒下或者自己后知后觉才向前行驶，这给后边车辆造成了不必要的堵塞。还有一些情况下，我们会发现前车速度突然减了下来，并且有可能伴随左右摇晃的情况。

之所以发生上述这些影响交通安全和效率的状况，很多情况是因为司机在驾驶过程中没有集中注意力，而是一边开始一边做起了诸如发微信、刷社交网络或者拿着手机打电话的事情。据有关报告，在五分之一的汽车事故是由于司机走神引起的，每年大概有 425000 人因此受伤，3000 人因此死亡[1]。因此需要有一些手段对司机的驾驶行为进行监控、判断以及提醒，减少因此发生的交通事故。

该项目旨在通过学习标记好的司机驾驶行为图像数据，对测试集中司机的行为进行检测，属于深度学习中的一个分支——计算机视觉范畴的问题。

之所以选择这样一个项目有以下几个原因：

- 该项目具有很强的实用价值和现实意义，好的解决方案能够减少交通事故的发生
- 该项目相较于其他项目更有挑战性，本人对深度学习十分有兴趣
- 该项目跟自身的业务方向契合

卷积神经网络应用于图像分类和识别已经有很长的历史了，但是直到近几年才得到了广泛的关注和发展。原因在于近些年尤其是移动互联网时代的到来，研究人员可以非常低成本地获得各种各样的数据，加上计算机和存储性能的大幅提高，应用卷积网络进行深度学习的门槛也越来越低。当然，深度学习领域研究的发展和进步也得益于 Kaggle、ImageNet 这样高水平的比赛。很多深度学习界的经典模型都是在这种比赛中被提出来的，这些模型不但在各自的比赛中表现出了卓越的水平，而且被证明在很多其他问题上也能表现的不错，非常值得借鉴。而且很多框架比如 Keras 就集成了一些经典的模型，利用这些预训练模型做迁移学习，可以加快模型构建的速度，提升模型的表型。

AlexNet[2]就是一种经典的网络，是由 Krizhevsky 提出的，他是在 ImageNet 的比赛中提出了这种卷积神经网络，并且取得了非常优异的成绩，这吸引了更多的研究人员在处理大规模图像识别问题时把目光关注在深度神经网络中。Simonyan 提出了一种具有小尺寸卷积核的多层卷积网络（VGG）,参数更少精确率足够高，取得了业界领先的表现[3]。He Kaiming 提出了一种比 VGG 更深的网络 ResNet，尽管层数更多，但是参数反而更少，而且非常易于训练和优化，在 ImageNet 数据集上的表现比 VGG 更好[4]。 谷歌的工作人员提出了一种叫做 Inception 的网络，据说是受到了电影盗梦空间的启发，这种网络的优点是不用人为指定卷积核的尺寸以及是否需要池化，由模型自动选择[5]。

1.2 问题描述

该项目的数据集来源于 Kaggle，训练集是由包含了 10 种司机驾驶行为的图像组成的，而测试集是的图像是所有待分类的图像。很显然这是一个监督学习的多分类问题，我们期望

利用训练集训练得到一个准确的分类器，对测试集中的图像进行归类，归类的范围就是训练集中的 10 种司机行为。每个图像只可能是 10 种分类的一种。

所以我们的目标就是去构建和训练这样一个分类器，由于数据集是大规模的图像，鉴于深度学习模型在大规模图像分类和识别中的成功案例，本文选择卷积神经网络作为模型。

本文选择以及训练卷积神经网路的策略如下：

1. 研究经典神经网络模型论文，掌握经典网络模型的配置，在应用中的数据集特征以及性能表现。
2. 研究 Kaggle 论坛中该项目的一些前人经验，参考一下前人的思路。
3. 结合对经典神经网络的研究，以及对本项目数据集的分析，选择模型，可以选择多个模型。
4. 针对模型所需要的数据集特点，对本项目数据集进行预处理。
5. 利用预训练模型进行单一模型下的训练和调优，直到结果满足。
6. 如果在单一模型下的效果始终不满意，考虑 bagging 以及模型融合。
7. 如果还想进一步提高，可以进一步考虑数据增强以及其他一些改进措施，这取决于对于模型和数据的理解有多深。
8. 最终模型的评价结果是需要在整个测试集上进行预测，然后提交到 Kaggle 上计算 logloss，logloss 值越低，表明所训练的模型越准确。

1.3 评估指标

上文提到了，本文是一个多分类问题，本解决方案采用交叉熵做为评估指标，公式如下：

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

其中 N 是测试集中图像的数量，M 是分类标签的个数，log 是自然对数，y_{ij}当样本 i 标签为 j 是值为 1，否则为 0。p_{ij}为样本 i 归属为标签 j 的预测概率。

2 分析

2.1 数据探索及可视化

数据集是 Kaggle 上提供的，一共有 22424 张照片以及对应的标签信息。每张图像的内容都是一个司机驾驶一辆车伴随有各种各样的行为，比如发信息，吃东西，打电话等。

数据集可能的标签有 10 类，分别是：

- 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

数据集中已经去掉了创建日期等原信息，并且保证了同一个司机不同即出现在训练集又出现在测试集。数据集中 imgs.zip 是所有图像的压缩文件，driver_images_list.csv 提供了图像的的标签信息。训练集样本中各类的分布情况如下：

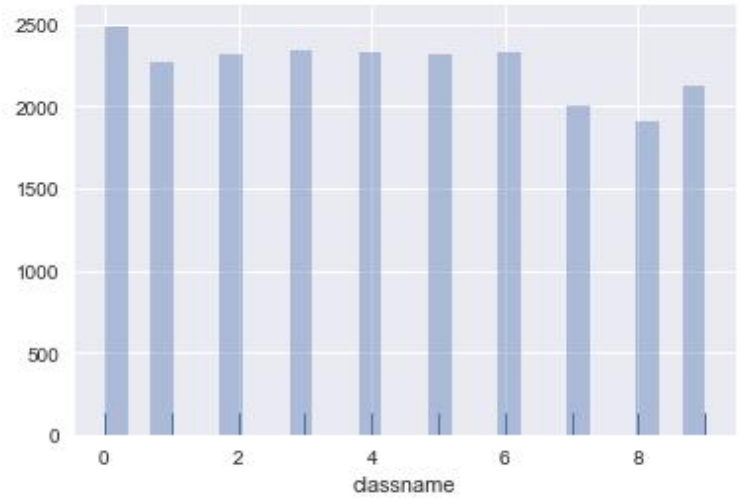


图 1 训练样本分布图-类别

如图 1 所示，训练集数据在每一类的分布较为均匀，不存在某一类特别多、某一类特别少的情况，即不存在数据偏斜类问题。

测试集的数据均位于 test 文件夹下，为了保证模型测试结果的有效性，训练集中出现的司机绝不可能在测试集中出现。这里需要注意的是测试集有 79726 个待分类的图像，几乎是训练集的 4 倍。所以如果出现过拟合严重的情况，可以考虑进行数据增强。

由于测试集和训练集司机不重合这一特点， 本项目在进行训练集和验证集划分时，需要按照司机来进行划分，这样能够使得在验证集的结果更有参考意义。

训练集图像在不同司机间的分布如图 2 所示：

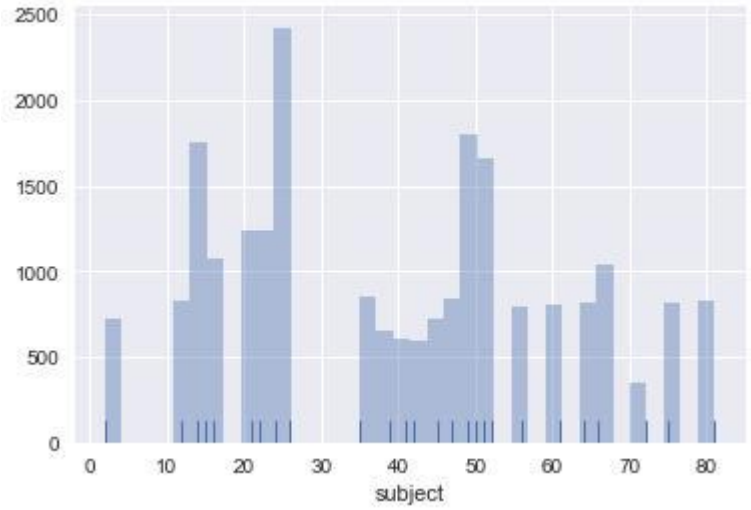


图 2 训练样本分布图-司机

如图所示，训练集在司机间的分布较为不均，因此不同的划分，可能产生较大的差异。

训练集和测试集均为如下图所示的彩色图像，对于彩色图像来说，其特征就是图像上每个像素的 RGB 值，每个图像的 RGB 数据就构成了模型的输入。而输出即模型的预测结果是一个归属于 10 种类别中各个类别的概率。最终期望的结果是测试集中每个图像归属于各个类别的概率值。典型样本图片为图 3 所示的 640*480 像素的彩色图像，图像是连续采集的，因此某些样本的差距非常小，尤其是同一个司机的相似度很高。因此在分割训练集和验证集时要特别注意按照司机进行分割，这样能够保证验证结果的有效性。

图 3 样本示例

2.2 基础理论

2.2.1 深度学习与 CNN

深度学习以数据的原始状态作为算法输入，经过算法层层抽象将原始数据逐层抽象为自身任务所需的最终特征表示，最后以特征到任务目标的映射作为结束[6]。相比传统机器学习算法仅学得模型这一单一“任务模块”而言，深度学习除了模型学习，还有特征学习、特征抽象等任务模块的参与，借助多层任务模块完成最终学习任务，故称其为“深度”学习。深度学习中的一类代表算法是神经网络算法，包括深度置信网络（deep belief network）、递归神经网络（recurrent neural network）和卷积神经网络（Convolution Neural Network，简称 CNN）等。特别是卷积神经网络，目前在计算机视觉、自然语言处理、医学图像处理等领域“一枝独秀”。

CNN 是一种层次模型，其输入是原始数据，如 RGB 图像，原始音频数据等。卷积神经网络通过卷积（convolution）操作、池化(pooling)操作和非线性激活函数（non-linear activation function）映射等一系列操作的层层堆叠，将高层语义信息逐层由原始数据输入层中抽取出来，逐层抽象，这一过程便是“前馈运算”（feed-forward）。其中，不同类型操作在卷积神经网络中一般称作“层”：卷积操作对应“卷积层”，池化操作对应“池化层”等等。最终，卷积神经网络的最后一层将其目标任务（分类、回归等）形式化为目标函数。通过计算预测值与真实值之间的误差或损失（loss），凭借反向传播算法（BP）将误差或损失由最后一层逐层向前反馈（back-forward），更新每层参数，并在参数更新后再次前馈，如此往复，知道网络模型收敛，从而达到模型训练的目的。

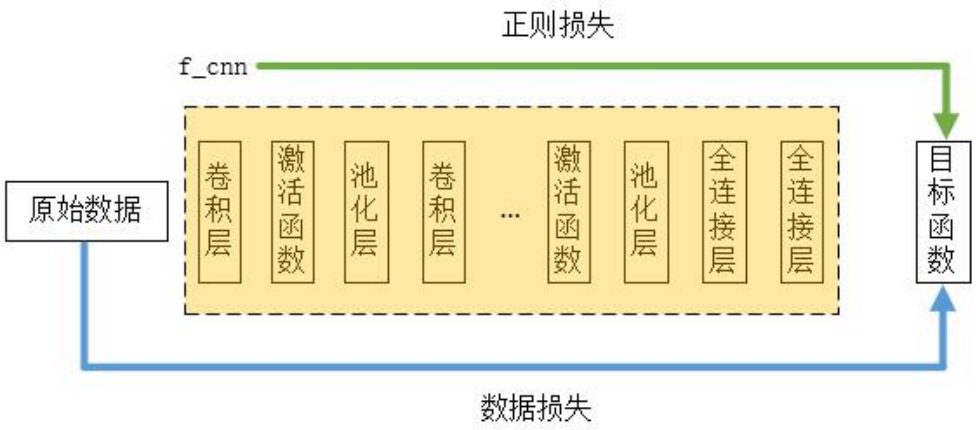


图 4 卷积神经网络基本流程图

如图 4 所示，对深度模型而言，其输入数据是未经任何人为加工的原始样本形式，后续则是堆叠在输入层上的众多操作层。这些操作层整体可看作一个复杂的函数 f_cnn，最终损失函数由数据损失和模型参数的正则化损失共同组成，深度模型的训练则在最终损失驱动下对模型进行参数更新并将误差反向传播至网络各层。模型的训练过程可以简单抽象为从原始数据到最终目标的直接“拟合”，而中间的这些部件正起到了将原始数据映射为特征（即特征学习）随后再映射为样本标记（即目标任务，如分类）的作用。

2.2.1 卷积

卷基是卷积神经网络中的基本操作，甚至在网络最后起分类作用的全连接层在工程实现时也是由卷基操作替代的。

卷积运算实际是分析数学中的一种运算方式，在卷积神经网络中通常仅涉及离散卷积。下面简单介绍一下二维场景的卷积操作。

假设输入图像为图 5 右侧所示的 5x5 矩阵，其对应的卷积核为一个 3x3 矩阵。同时假定卷积操作没做一次卷积，卷积核移动一个像素位置，即卷积步长为 1。

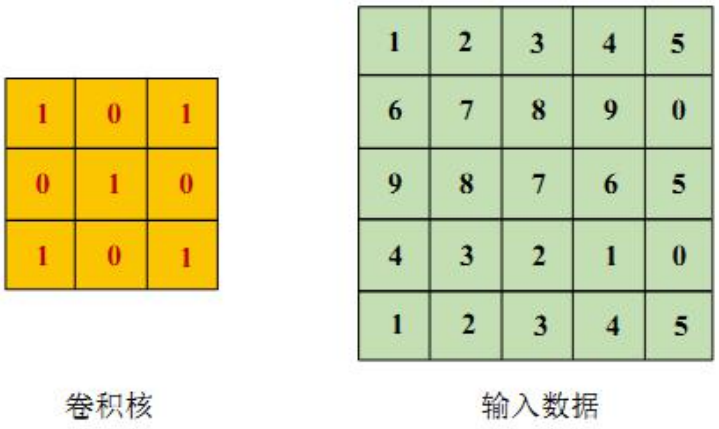
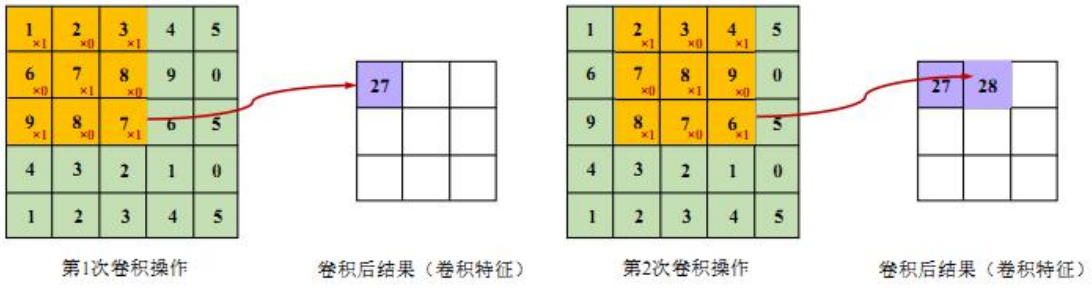


图 5 二维场景下的卷积核与输入数据

第一次卷积操作从图像(0,0)像素开始，由卷积核中参数与对应位置图像像素逐位相乘后累加作为一次卷积操作结果，即 1x1+0x2+1x3+0x6+1x7+0x8+1x9+0x8+1x7=1+3+7+9+7=27。如图 6 中 a 图所示。在步长为 1 时，卷积核按照步长大小在输入图像上从左到右自上而下依次将卷积操作进行下去，最终输出 3x3 大小的卷积特征，同时将结果作为下一层操作的输入。



(a) 第一次卷积操作及得到的卷积特征。　(b) 第二次卷积操作及得到的卷积特征。

(c) 第三次卷积操作及得到的卷积特征。　(d) 第九次卷积操作及得到的卷积特征。

图 6 卷积运算示例

类似地，卷积操作可以应用到三维场景中，本文不进行深入讨论。

(a) 原图。　(b) 整体边缘滤波器 K_{e}。

(c) 横向边缘滤波器 K_{h}。　(d) 纵向边缘滤波器 K_{v}。

图 7 卷积应用示例

从上面的介绍可以看出，卷积是一种局部操作，通过一定大小的卷积核作用于局部图像区域获得图像的局部信息。本节以三种边缘卷积核（亦可称为滤波器）来说明卷积神经网络中卷积操作的作用。如图 7 所示，我们在原图上分别作用整体边缘滤波器、横向边缘滤波器和纵向边缘滤波器，这三种滤波器（卷积核）分别为下边公式中的 3x3 大小卷积核：

$$K_{\mathrm{e}}=\begin{bmatrix}0&-4&0\\-4&16&-4\\0&-4&0\end{bmatrix}\quad K_{\mathrm{h}}=\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}\quad K_{\mathrm{v}}=\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$$

其中 K_{e} 是一个整体边缘滤波器，可消除四周像素值差异小的区域而保留显著差异区域，从而可以检测出物体边缘信息。 K_{h} 和 K_{v} 分别是保留横向和纵向边缘信息的滤波器。

事实上，卷积网络中的卷积核参数是通过网络训练学出的，除了可以学到类似的横向、纵向边缘滤波器，还可以学到任意角度的边缘滤波器。当然，不仅如此，检测颜色、形状、屋里等等众多基本模式的滤波器都可以包含在一个足够复杂的深层卷积神经网络中。通过组合这些滤波器（卷积核）以及随着网络后续操作的进行，基本而一般的模式会逐渐被抽象为具有高层语义的“概念”表示，并以此对应到具体的样本类别。颇有“盲人摸象”后，将各自结果集大成之意。

2.2.2 激活函数

激活函数层又称为非线性映射层，顾名思义，激活函数的引入为的是增加整个网络的表达能力（即非线性）。否则，若干线性操作层的堆叠仍然只能起到线性映射的作用，无法形成复杂的函数。实际应用中，有多达十几种激活函数可供选择，常见的是 Sigmoid 型激活函数和 ReLU(Rectified Linear Unit)型激活函数。本节以这两种激活函数为例进行介绍。

激活函数是对生物神经元特性的模拟：接受一组输入信号并产生输出。在神经科学中，生物神经元通常有一个阈值，当神经元所获得的输入信号累积效果超过了该阈值，神经元就被激活而处于兴奋状态；否则处于抑制状态。在人工神经网络中，因 Sigmoid 型函数可以模拟这一生物过程，从而在神经网络发展历史中处于相当重要的地位。

Sigmoid 型函数也称 Logistic 函数：

$$\sigma(x)=\frac{1}{1+\exp(-x)}$$

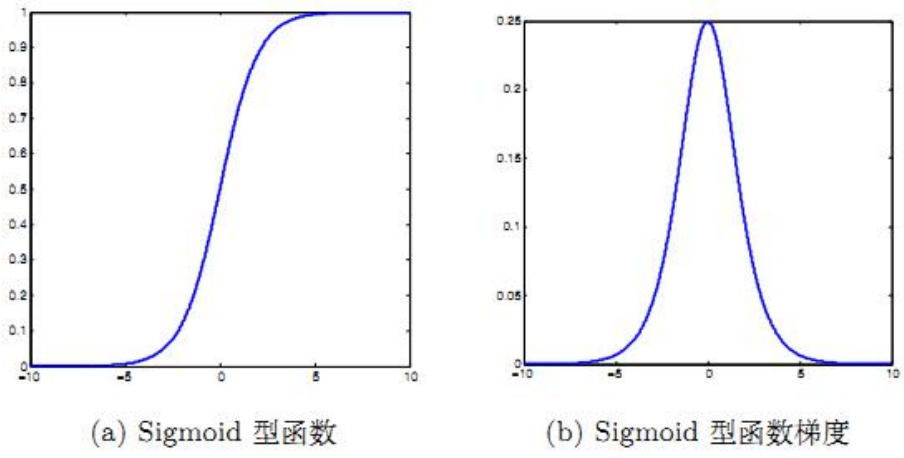


图 8 Sigmoid 型函数及函数梯度

Sigmoid 函数的形状如图 8a 所示。显然，经过 Sigmoid 函数作用后，输出响应的值域被压缩到[0,1]之间，而 0 对应了生物神经元的“抑制状态”，1 则恰好对应了“兴奋状态”。

进一步观察可以发现，在 Sigmoid 函数两端，对于大于 5（或小于-5）的值无论多大（或多小）都会压缩到 1（或 0）。如此便带来一个严重的问题，即梯度的“饱和效应”。对照 Sigmoid 函数的梯度图（图 8b），大于 5（或小于-5）部分的梯度接近 0，这会导致在误差反向传播过程中导数处于该区域的误差将很难甚至根本无法传递至前层，进而导致整个网络无法训练（导数为 0 将无法更新网络参数）。此外，在参数初始化的时候还需特别注意，要避免初始化参数直接将输出值域带入这一区域：一种可能的情形是当初始化参数过大时，将直接引发梯度饱和效应而无法训练。

ReLU 的中文意思是修正线性单元，这种激活函数的优点是可以避免梯度饱和效应的发生，也因此 ReLU 函数是目前深度卷积神经网络中最为常用的激活函数之一。

ReLU 函数实际上是一个分段函数，定义如下：

$$
\begin{aligned}
&rectifier(x) \\
&= \max\{0, x\} \\
&= \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}
\end{aligned}
$$

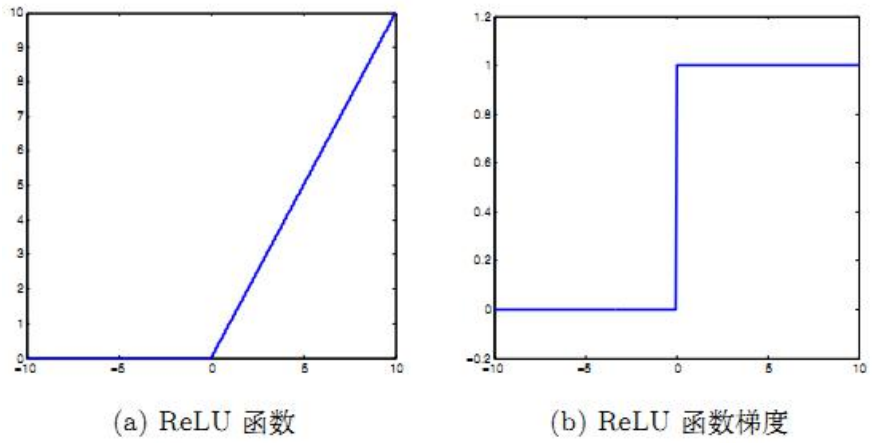


图 9 ReLU 函数及其函数梯度

如图 9 所示，ReLU 函数的梯度在 $x \geq 0$ 时为 1，反之为 0。对于 $x \geq 0$ 部分完全消除了 Sigmoid 型函数的梯度饱和效应。同时，在实验中还发现相比 Sigmoid 函数，ReLU 函数有助于随机梯度下降方法收敛，收敛速度约快 6 倍左右[2]。正是由于 ReLU 函数的这些优质特性，ReLU 函数已经成为目前卷积神经网络以及其他深度学习模型（如递归神经网络 RNN 等）激活函数的首选之一。

2.2.3 池化

图 10 池化运算示例

池化是卷积神经网络中的一个重要概念，最常见的池化为平均池化和最大值池化。通常在连续卷积层之间插入池化层。最大池化操作示意如图 10 所示。

从图 10 中的例子可以发现，池化操作后的结果相比输入维度减小了，其实池化操作的本质上是一种“降采样”操作。池化层的引入是仿照人的视觉系统对视觉输入对象进行降维（降采样）和抽象。池化层的功效如下：

- 特征不变性。池化操作使模型更关注是否存在某些特征而不是特征具体的位置。可看作是一种很强的先验，使特征学习包含某种程度自由度，能容忍一些特征微小的位移。
- 特征降维。由于池化操作的降采样作用，池化结果中的一个元素对应于原输入数据的一个子区域，因此池化相当于在空间范围内做了维度约减，从而使模型可以抽样更广泛的特征。同时减小了下一层输入大小，今儿减小计算量和参数个数。
- 在一定程度防止过拟合，更方便优化。

2.3 基准模型

由于本项目是一个大规模图像数据的多分类问题，所以选择了卷积神经网络这一近些年在处理类似问题表现优异的模型。在众多优秀的基准模型中，选择了 VGG16 作为主要参考模型，VGG19 和 ResNet50 作为基准模型进行对比。其中 VGG16 的模型结构较为简单，尽管需要较多的参数，由于可以得到训练好的权重，所以可以排除计算消耗上的顾虑。而 ResNet50 尽管层数多，但是参数却更少，在某些数据集上的表现超过了 VGG16。而 VGG19 相较于 VGG16 稍复杂一些，通过对比不同模型的表现，能够加深对模型的认识，寻找调优的方向。

VGG16、VGG19 以及 ResNet50 的网络结构如下表所示：

VGG16	VGG19	ResNet50
Input(224x224 RGB image)		
conv3-64 conv3-64	conv3-64 conv3-64	conv7-64 maxpool
conv3-128 conv3-128	conv3-128 conv3-128	conv1-64 conv3-64 conv1-256 conv1-64 conv3-64 conv1-256 conv1-64 conv3-64 conv1-256
conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256	conv1-128 conv3-128 conv1-512 conv1-128 conv3-128 conv1-512 conv1-128 conv3-128

		conv1-512 conv1-128 conv3-128 conv1-512
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512	conv1-256 conv3-256 conv1-1024 conv1-256 conv3-256 conv1-1024 conv1-256 conv3-256 conv1-1024 conv1-256 conv3-256 conv1-1024 conv1-256 conv3-256 conv1-1024 conv1-256 conv3-256 conv1-1024
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512	conv1-512 conv3-512 conv1-2048 conv1-512 conv3-512 conv1-2048 conv1-512 conv3-512 conv1-2048
maxpool		average pool
FC-4096		FC-1000
FC-4096		
FC-1000		
soft-max		

表 1 VGG16、VGG19、ResNet50 网络结构

VGG16 和 VGG19 是 VGG-Nets 的代表，由牛津大学提出，在 2014 年 ImageNet 竞赛中获得了定位人物第一名和分类任务第二名。相比其他网络，VGG-Nets 中普遍使用了小卷积核，韦德是在增加网络深度时确保各层输入大小随深度增加而不极具减小。同时，网络卷积层的通道数也是逐层增加。由于 VGG-Nets 具备良好的泛化性能，其在 ImageNet 数据集上的预训练模型被广泛用于除最常用的特征抽取外的诸多问题。

深度神经网络面临着一个问题，就是随着深度的增加，训练会变得愈加困难。这主要因为在基于随机梯度下降的网络训练过程中，误差信号的多层反向传播非常容易引发梯度“弥

散”（梯度过小使得回传的训练误差极其微弱）或者梯度爆炸（梯度过大导致模型无法更新参数）。目前，一些特殊的权重初始化策略以及批规范化策略等方法使这个问题得到极大改善。但是，随着网络深度的进一步增加，问题仍然存在。残差网络很好的解决了网络深度带来的训练困难，他的网络性能远超传统网络模型，曾在 ILSVRC 2015 和 COCO 2015 竞赛的检测、定位和分割任务中纷纷斩获第一。ResNet50 是残差网络的代表。

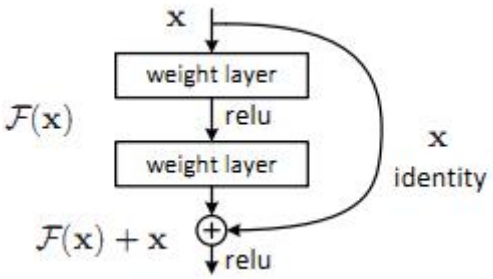


图 11 残差学习模块

残差网络的基本结构是残差学习模块，如图 11 所示。残差学习模块有两个分支，其一是左侧的残差函数，其二是右侧的对输入的恒等映射。这两个分支经过一个简单整合（对应元素的相加）后，再经过一个 ReLU 激活函数，从而形成整个残差学习模块。

图 12 展示了两种残差学习模块，其中左侧的残差模块由两个 3x3 卷积堆叠而成，但是随着网络深度增加，这种结构并不是十分有效。右图所示为“瓶颈残差模块”，一次由 3 个 1x1,3x3 和 1x1 的卷积层构成，这里 1x1 卷积能够起降维或者升维的作用，从而令 3x3 的卷积可以在相对较低维度的输入上进行，已达到提高计算效率的目的。在非常深的网络中，“瓶颈残差模块”可大量减少计算代价。

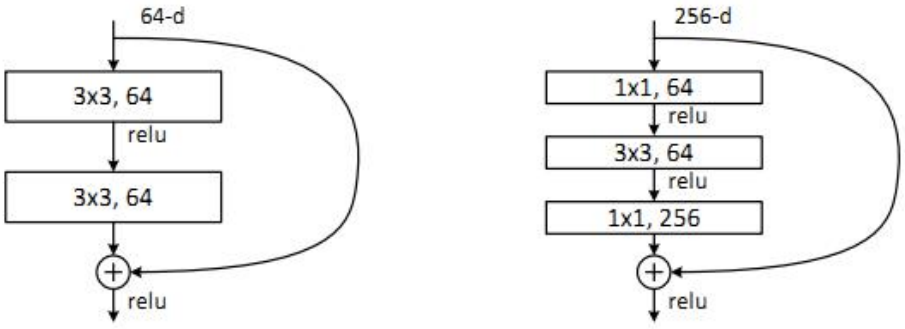


图 12 两种不同的残差学习模块

残差学习模块实现了一种近路连接的效果，可直接通过简单的恒等映射完成。在输入输出维度一致的情况下，残差网络不需要引入额外的参数和计算的负担。通过近路连接的方式，即使面对特别深的网络，也可以通过反向传播端到端的学习，使用简单的随机梯度下降的方法就可以训练。这主要受益于近路连接使梯度信息可以在多个神经网络层之间有效传播。

残差网络与传统 VGG 相比，若如近路连接，残差网络实际上就是更深的 VGG 网络，只不过残差网络以全局平均池化层替代了 VGG 网络的全连接层，一方面使得参数大大减少，另一方面减少了过拟合风险。

Keras 提供了这几种优秀的预训练模型，可以很方便的使用和集成。由于事前不知道哪个模型表现的更好，因此可以互为基准模型，本文将在三个模型下分别进行训练和调优，再对三种训练好的模型进行集成。此外本文还将以 Kaggle Private LeaderBoard 上的排名做为参考，目标是进入前 10%，即排在 1440 个成绩的前 144 名，对应在测试集的 logloss 表现需要比 0.2563 小。

2.4 迁移学习

如果想要做一个计算机视觉应用，相比于从头训练权重，或者随机初始化权重，如果直接下载别人训练好的权重，通常能进展得相当快。用这个作为预训练，然后转换到你感兴趣的任务上去。

计算机视觉领域有很多公开的数据集，如 ImageNet 等。很多人在这些数据集上花了很多时间训练了自己的网络，可以直接下载他们训练好的权重，作为自己网络的初始化权重，用迁移学习把这些数据集上的知识迁移到你自己的问题上。

假设你已经下载了训练好的权重，并且你自己的训练集数量较少，那么此时可以把除了 softmax 层之外的所有层都冻结（这些参数都不改变），然后把原有的 softmax 改为自己的 softmax，比如一个三分类的 softmax。然后只训练 softmax 层的参数。那么如何冻结呢，所幸很多神经网络架构都支持这样的操作，比如将 trainParameter 设置为 0，或者 freeze 设置为 1，来指定是否训练某一层的权重。

迁移学习还有一个技巧能，就是把前面冻结的所有层视为一个函数，然后将其作用到输入上得到的输出，即特征值或者激活值，将其保存到硬盘。然后整个网络就变成了一个很浅的网络结构，从而加速了整个训练。存储到硬盘或者叫预训练的优点就是不用每次都遍历数据集重新计算激活值了。

如果有更多的训练集，就可以考虑放开更多的层去做训练。如果训练集足够多，可以考虑将所有层放开，然后初始化权重为已经训练好的权重进行训练。

2.5 优化策略

2.5.1 学习率的设定

模型训练的一个关键设定是学习率，一个理想的学习率会促进模型收敛，而不理想的学习率甚至会导致模型直接目标损失值“爆炸”无法完成训练。学习率设定可遵循以下两项原则：

1. 模型训练开始时的初始学习率不宜过大，如果发现刚开始训练没几个批次，模型目标损失值就急剧上升，说明学习率过大，应减小学习率；
2. 训练过程中，学习率应随轮数增加而减缓。

此外，寻找理想学习率或诊断模型训练学习率是否合适可借助模型学习曲线的帮助。

2.5.2 优化算法的选择

深度学习经过多年的发展，出现了一些列有效的网络训练优化算法，如随机梯度下降、基于动量的随机梯度下降、Adagrad、Adam 等。在不同的任务中，不同的优化算法可能起到不同的效果。本文中选择的是自适应学习率的 Adam 算法。

2.5.3 神经网络微调

当目标较少且目标数据与原始数据非常相似时，可仅微调网络靠近目标函数的后几层；

当目标数据重组且相似时，可微调更多层，也可全部微调；当目标数据充足但与原始数据差异较大，此时须多调节一些网络层，直至微调全部；当目标数据极少，同时还与原始数据有较大差异时，这种情形比较麻烦，不过仍可以尝试首先微调网络后几层再微调整个网络。

2.6 集成学习

集成学习时机器学习中的一类学习算法，指训练多个学习器并将他们组合起来使用的方法。尽管深度神经网络模型已经拥有强大的预测能力，但是集成学习的使用仍然能起到“锦上添花”的作用。一般来讲，深度模型的集成多从“数据层面”和“模型层面”两方面着手。

2.6.1 数据层面集成

数据层面的集成分两种：

第一种是对数据应用不同的增强方法训练不同的分类器，然后对不同分类器的预测结果取平均。

第二种是对训练集做不同的划分（采样），训练得到若干个分类器，最后对这些分类器的结果取平均或者投票，这种方法也叫 bagging。

2.6.2 模型层面集成

模型层面的集成也分两种：

第一种是单模型集成。这种集成方法通常应用单个模型不同层的特征进行融合，不同层特征富含的语义信息可以相互补充。对于特征融合应该选取哪些网络层，这里有一个经验是：最好使用靠近目标函数的基层卷积特征，因为愈深层特征包含的高层语义愈强、分辨能力也愈强。

第二种是多模型集成。顾名思义，多模型集成需要构建和训练不同的模型，通常有以下几种生成多模型的策略：

- 同一模型不同初始化
- 同一模型不同训练轮数
- 不同目标函数
- 不同网络结构

多模型的集成方法有以下几种：

- 直接平均
- 加权平均
- 投票法
- 堆叠法

3 方法

3.1 数据预处理

本文是基于预训练模型做迁移学习，所以预处理的方式跟各预训练模型保持一致。其中vgg16、vgg19、resnet50 都是读取 224x224 的 RGB 信息然后转换为 shape 为（224,224,3）的数组，之后运用 keras 预处理模型内置的 preprocess_input 函数做预处理。

除了图像本身的预处理之外，本文还对数据读取的过程进行了特殊处理。首先是通过读取项目提供的 csv 表格，得到按照司机汇总的图像信息，目的在于之后把数据集按照不同的司机做 k 折划分。之所以要进行 k 折划分，是因为训练集数据较少，做一次划分训练得到的模型性能较差。而之所以要按照司机做划分，是因为同一个司机相似度极高，如果训练集和验证集均存在同一个司机，那么就很容易识别出来， 这样得到的模型在测试集会过拟合严重。项目本身为了保证模型有效性，训练集中出现的司机，在测试集中绝不可能出现。所以按照司机进行划分能够使得验证结果更加可信。

3.2 调试过程

在项目执行过程中遇到了很多困难，绕了很多弯路，进行了很多尝试，我对深度学习的理解也越来越深入。刚开始完全没有思路，就参照了知乎提取特征向量进行预测的方法，然后只做了一次训练集划分，也没有按照司机进行划分。在验证集的表现还不错，但是测试集表现一塌糊涂。

然后在划分训练集的时候选择了按照司机进行划分，这次验证集和测试集的表现比较一致了，但是经过多次调节学习率，验证集 loss 始终在 1 以上。

在参考了 Kaggle 论坛上的一些论点和实现，并且在指导导师的指导下，找出了问题所在，并明确了改进的方向。首先是数据预处理的方式没有跟各预训练模型一致，然后是提取特征向量的方式对于本项目可能不是十分适用，然后就是运用 bagging。

在明确了方向之后，还是踩了很多坑，比如 bagging 的时候上来就对训练集做 k 折划分，然后再对每个划分做训练。这种思路受到了内存的限制，直接爆出了内存错误。之后通过改进，每次划分完成就进行模型训练，然后将训练的结果保存起来，从而解决了内存问题。不正确的预处理也可能导致内存问题。因为在很大的数组中存入整数和浮点数占用内存是完全不一样的，所以在读取的时候进行不要做预处理，把预处理放在划分之后训练之前做能缓解内存占用过多的难题。

刚开始运用 bagging 的时候，发现验证集 loss 在后边的划分越变越小，而在测试集上反而更大了。后来才发现没有正确的使用 bagging，每次划分都应有新建一个模型，而不是利用之前划分训练过的模型。

测试集将近 80000 张图片一次性读到内存并做预处理是一件非常占用内存的事儿。仿照 ImageDataGenerator 的做法，分批读入、预处理、预测，最后将预测结果进行汇总和暂存。数据暂存这个思路对于成功训练和预测非常重要，首先能够用硬盘来代替内存存储数据，其次是数据可以重复利用。

在明确了数据预处理要和各预训练网络保持一致之后，本文首先是参照论文中提到的，减去平均值，然后显示转置然后在减均值，之后再转置。最终的结果还是不错。偶然看 keras

文档的时候发现预训练模型内置了预处理函数，之后又尝试用这些内置的预处理函数做预处理，结果比自己处理半天的表现要好很多。

整个项目编程部分最复杂的部分就是训练集图像按照司机机型 k 折划分了。应该说这里的实现方式很灵活，但是不同实现可能得到的表现不尽相同。本文首先借助 csv 文件将训练集图像按照司机的视角进行归类，给每个图像打上属于某个司机的标签，这样在后边进行 k 折划分的时候，拿到一张图像，如果是验证集的司机就归入验证集，反之归入训练集。

刚开始构建模型做训练时，首先想到了生成器的方式，觉得可以省内存，而且很方便地做预处理。但是在之后的尝试中发现不能跟 bagging 很好的整合，于是放弃了生成器的方式。但是多多少少运用了生成器分批读取的思路。

3.3 完善

本文完善的思路是首先在单个模型下通过调整超参数（主要是学习率）得到一个较为合理的分数，这里还有一个小技巧，就是每种模型选择在两个划分上进行训练，控制单一变量进行对比，这样能够加快调参的速度，避免跑偏。

本文分别对在同一个 fold 的划分下对比了 vgg16、vgg19、resnet50 在两种学习率 1e-5 和 1e-6 两种情况下的表现，绘制了如图 13-15 所示的学习曲线。通过对比两种学习率下的学习曲线，发现在 1e-5 时能够得到较好的表现，最终选择三种模型 10 折划分下均采用 1e-5 作为学习率。

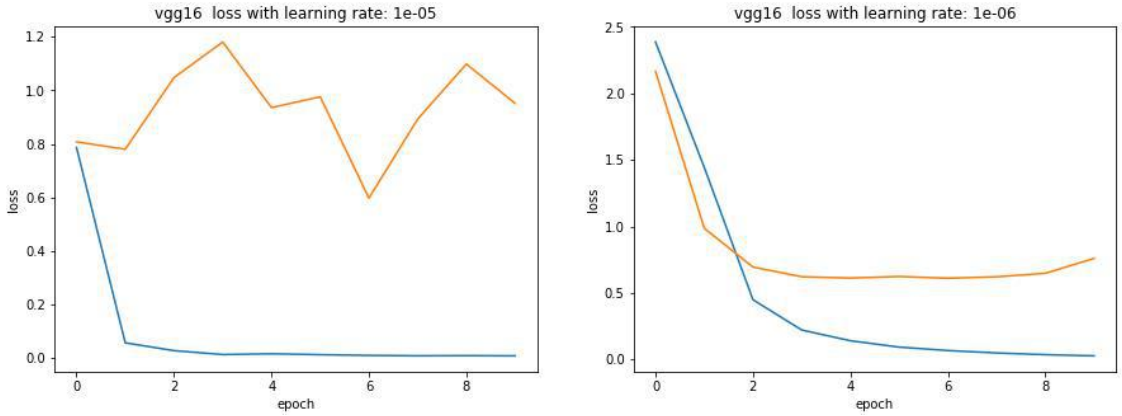


图 13 vgg16 两种学习率下的学习曲线

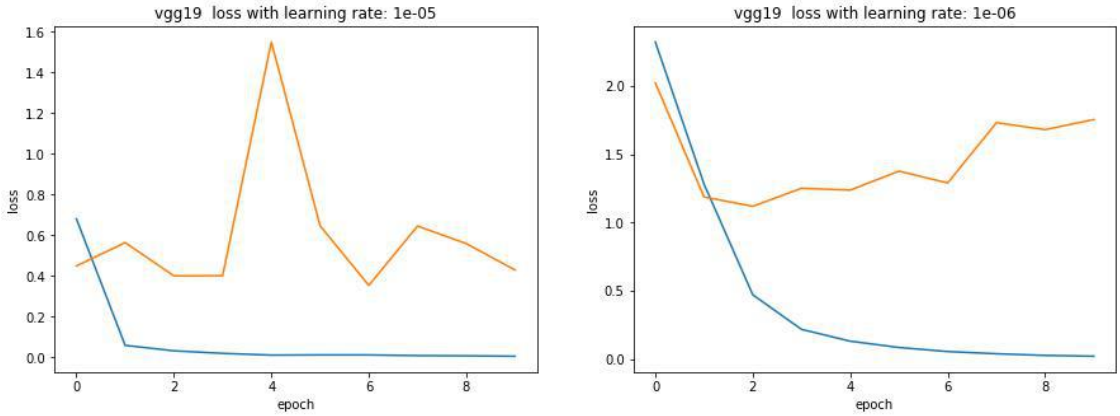


图 14 vgg19 两种学习率下的学习曲线

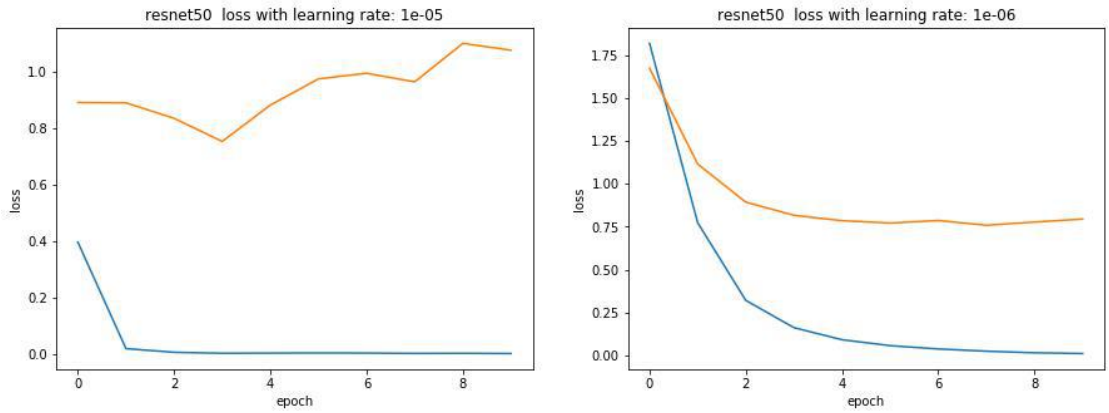


图 15 resnet50 两种学习率下的曲线

最终得到各模型在验证集的表现如下：

	res50-logloss	vgg16-logloss	vgg19-logloss
fold1-val-loss	0.5234	0.4336	0.6124
fold2-val-loss	0.4667	0.3494	0.4797
fold3-val-loss	0.3435	0.3033	0.3473
fold4-val-loss	0.3169	0.3953	0.5224
fold5-val-loss	0.2333	0.2269	0.2292
fold6-val-loss	0.2384	0.3407	0.3150
fold7-val-loss	0.6515	0.5532	0.7193
fold8-val-loss	0.6347	0.8077	0.7732
fold9-val-loss	0.2648	0.3049	0.2354
fold10-val-loss	0.4552	0.3760	0.7363

表 2 三种模型运用内置预处理函数的验证集表现

4 结果

4.1 模型的验证和评价

本文经过大量的调试以及尝试，最终各模型以及组合在测试集上的最佳表现如下：

model	kaggle private score
vgg16	**0.23325**
vgg19	0.26436
resnet50	0.34231
vgg16+vgg19	0.23540
vgg16+resnet50	0.25801
vgg16+vgg19+resnet50	0.24928

表 3 三种模型及组合下的测试集表现

本文在 resnet50、vgg19、vgg16 三个预训练模型上进行训练和调优，对比三者表现以及各种组合下的表现，最终选择了表现最好的 vgg16 作为最后的模型。这个结果在 kaggle Leaderboard private score 上能够排到 110，进入了前 10%（144/1440）。由于对于训练集和测试集的预处理方式保持一致，验证集 loss 表现和测试集 loss 表现也较为接近，因此模型和结果均较为合理，达到了预期的目标。

4.2 Kaggle 表现

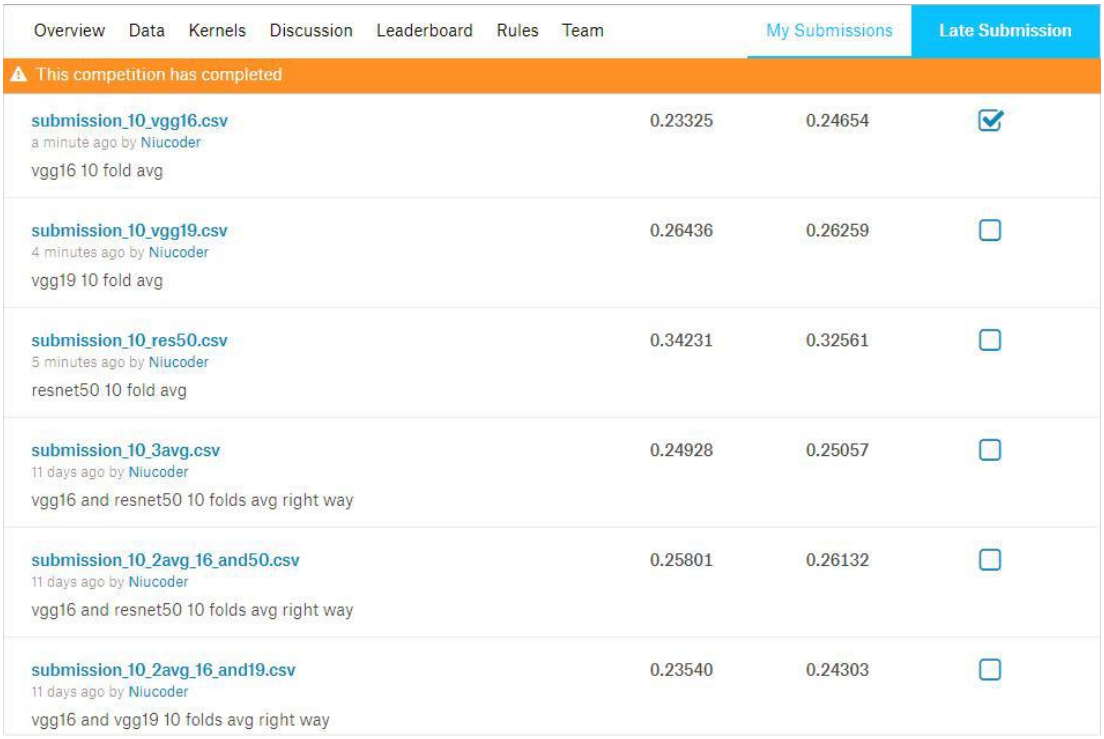


图 16 Kaggle 表现

5 结论与展望

5.1 总结

本文运用三种预训练模型 vgg16、vgg19、resnet50，分别进行模型训练、调优、预测。最终在单个 vgg16 下得到了最佳的表现，达到预期目标的同时，加深了应用 vgg16、vgg19 以及 resnet50 处理大规模图像分类问题的理解。为今后从事深度学习以及图像分类相关研究奠定了坚实的基础。

5.2 思考

本文之所以进展的比较顺利，主要得益于预训练模型的存在以及整个 kaggle 社区和 keras 的帮助。如果自己从头到尾地利用 tensorflow 去构建一各个的卷积层，一个个的池化层，还不知道要进行多少次尝试才能达到现在的表现。但我发现这种从头构建自己网络的能力也正是我缺乏的，需要在今后的学习中进一步加强。当然这需要长时间的积累，通过在解决实际中的问题来加深对网络的理解。

整个项目做下来，对于利用 keras 构建深度学习模型进行大规模图像分类问题有了更进一步的认识。对于数据预处理，模型调优，bagging，模型集成的应用为今后从事相关的研究打下了坚实的基础。

项目最优挑战性的一个地方就是训练集非常少，是测试集的四分之一，所以应用 bagging 可以说直接解决了这个难题。在编程实现过程中着重注意的一点就是编写内存占用少的代码，适当地运用数据暂存，恰当的使用生成器，这样能够尽可能地避免内存耗尽。

5.3 展望

尽管最终的结果还算满意，但是距离最优秀的模型还是有较大的差距。根据对本文问题的认识以及参考的一些文章，提出以下进一步改进的方案：

- 提升硬件水平，比如增加 GPU 和内存，利用并行化的代码实现，加快数据处理和训练速度
- 由于训练集比较少，相信进行数据增强后会有更好的表现
- 尝试更多优秀的模型，鉴于本项目集成效果并不好，所以不推荐集成
- 尝试在单个模型上锁住一部分层进行调优
- 尝试深度浅一些的模型

参考

[1] https://www.cdc.gov/motorvehiclesafety/distracted_driving/

[2] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[3] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:14094.1556, 2014.

[4] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[5] Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 2818-2826.

[6] 魏秀参,解析卷积神经网络,2017