

机器学习工程师纳米学位毕业项目

Kaggle 比赛之走神司机侦测

王闻宇

2017 年 9 月 7 日

目录

1. 定义	4
1.1. 项目概述	4
1.2. 问题描述	4
1.3. 输入数据集	4
1.4. 评估指标	5
1.5. 项目目标	5
2. 分析	6
2.1. 数据可视化	6
2.1.1. C0-安全驾驶	7
2.1.2. C1-右手打字	7
2.1.3. C2-右手打电话	8
2.1.4. C3-左手打字	9
2.1.5. C4-左手打	9
2.1.6. C5-操作收音机	10
2.1.7. C6-喝水	10
2.1.8. C7-拿后面的东西	11
2.1.9. C8-梳妆打扮	11
2.1.10. C9-和乘客说话	12
2.1.11. 每个训练数据集的司机的图片是连续的	12
2.1.12. 看看测试集的数据	13
2.1.13. 数据总结	13
2.2. 算法与技术	14
2.2.1. 卷积神经网络	14
2.2.2. Loss 函数的选择	14
2.2.3. 优化器的选择	15
2.2.4. 迁移学习的选择	17
2.2.5. Fine-tune	17
2.2.6. 多模型融合的选择	18
2.2.6.1. Vgg16	18
2.2.6.2. ResNet50	18

2.2.6.3. InceptionV3	19
2.2.6.4. Xception.....	20
2.2.7. CAM 可视化.....	21
2.2.8. 参数优化	21
2.2.8.1. 使用 Dropout 层	22
2.2.8.2. 使用 clip 来防止 logloss 分数过高	22
2.3. 解决问题的步骤和思路	23
3. 实现	23
3.1. 在训练数据中区分训练集和验证集	23
3.2. 数据预处理	24
3.3. 基准模型评估.....	25
3.4. 单模型 Fine-tune 及优化	27
3.4.1. ResNet50 模型 Fine-tune.....	27
3.4.2. InceptionV3 模型	29
3.4.3. Xception 模型.....	31
3.5. 多模型融合	33
3.5.1. 之前模型的汇总.....	33
3.5.2. 新的模型，混合模型.....	33
3.5.3. 结果.....	34
4. 结论	35
4.1. 总结.....	35
4.2. 后续改进.....	35
4.2.1. 全面彻底清理训练数据异常数据.....	35
4.2.2. 可以参考前后几帧图片来判断	35
参考文献.....	36

1. 定义

1.1. 项目概述

我们经常遇到这样的场景：一盏灯变成绿色，你面前的车不走。另外，在没有任何意外发生的情况下，前面的车辆突然减速，或者转弯变道。等等这些现象，给道路安全带来了很大的影响。

那么造成这样现象的原因是什么，主要有因为司机疲劳驾驶，或者走神去做其他事情，想象身边的例子，开车时候犯困，开始时候打电话，发短信，喝水，拿后面东西，整理化妆的都有。这对道路安全和行车效率形成了极大的影响。



据中国安全部门介绍，五分之一的车祸是由司机分心引起的。令人遗憾的是，这样一来，每年有 42.5 万人受伤，3000 人因分心驾驶而死亡。

我们希望通过车内摄像机来自动检测驾驶员走神的行为，来改善这一现象，并更好地保证客户的安全。

1.2. 问题描述

我们要做的事情，就是根据车内摄像机的画面自动检测驾驶员走神的行为。如果是安全驾驶则一切正常，如果有走神行为，给予警报提醒。

驾驶员可能存在的走神的行为，大概有如下几种，左右手用手机打字，左右手用手持方式打电话，调收音机（玩车机），喝饮料，拿后面的东西，整理头发和化妆，和其他乘客说话。

侦测的准确率 accuracy 就是衡量解决这个问题好坏的重要指标

1.3. 输入数据集

输入数据集来自 Kaggle 下载地址如下：

<https://www.kaggle.com/c/state-farm-distracted-driver-detection/data>

下载下来解压后有 3 个文件

- driver_imgs_list.csv.zip (92.89K)
- imgs.zip (4G) 所有的图片数据，解压后
 - train (训练集数据)
 - c0 ~ c9 分别表示不同状态的训练集

- test (测试集数据，用于提交 Kaggle 比赛的测试集)
- sample_submission.csv.zip (206.25K) Kaggle 比赛需要提交的样本

其中 driver_imgs_list.csv.zip 的是对分类标号和人分类编号的 csv 文件。这个 csv 表格有三列

subject : 人的 ID , 不同的人 , 这个值不同

classname : 状态 , c0 ~ c9

img : 图片名称

数据有这些特点

训练集的图片数量是 22424 , 测试集的图片是 79726 , 测试集的数量远大于训练集。

而且训练集司机完全不是测试集司机。

由于数据有这些特点 , 那么过拟合可能是要遇到的最大的问题。

不过 , 对于每个司机而言 , 其数据是连续的 , 是从一个视频里面一帧一帧截取出来的 , 这个特点可能对最后的分析有用。

1.4. 评估指标

这是典型的分类问题 , 评估指标采用

精度 accuracy 来评估结果好坏。

Logloss 的评估方式 , 这也是 kaggle 比赛的评估方式

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

对比这两种方案。Accuracy 对于判断正确和错误的比重是一样的 , 也就是对了就多一个 , 错了就少一个 , 最终看正确的百分比 , 也就是判断正确的占总体测试数据的比例。

而 logloss 的评估方式对判断是不是有明显的方法 , 如果正确了 , $P_{ij}=1 \Rightarrow \log(P_{ij})=0$, 而 $P_{ij}=0.999 \Rightarrow \log(P_{ij})=-0.001$ 。最后增加的 log 差不多。但如果判断错误 , 如 $P_{ij}=0 \Rightarrow \log(P_{ij}) = -\infty$ 。 $P_{ij}=0.001 \Rightarrow \log(P_{ij})=-6.9$ 也就是判断错误一个 , 对得分的影响会非常大 , 所以用 logloss 评估对比 accuracy , 更能反映模型和算法的能力。

1.5. 项目目标

我的目标是： accuracy > 0.93 并且 logloss < 1.0 并且 世界排名在前 1/5。

2. 分析

2.1. 数据可视化

训练数据集中，共有图片 22424 张，测试数据集中，共有图片 79726 张
图片大小都是 640x480。

训练数据集一共有 26 个司机，测试数据集有司机若干，且和训练司机是不同的。

每个司机的图片的数量是：

p002	725
p012	823
p014	876
p015	875
p016	1078
p021	1237
p022	1233
p024	1226
p026	1196
p035	848
p039	651
p041	605
p042	591
p045	724
p047	835
p049	1011
p050	790
p051	920
p052	740
p056	794
p061	809
p064	820
p066	1034
p072	346
p075	814
p081	823

一共有 10 种状态，且每个状态下面的图片数量如下：

c0	2489
c1	2267
c2	2317
c3	2346
c4	2326
c5	2312

c6	2325
c7	2002
c8	1911
c9	2129

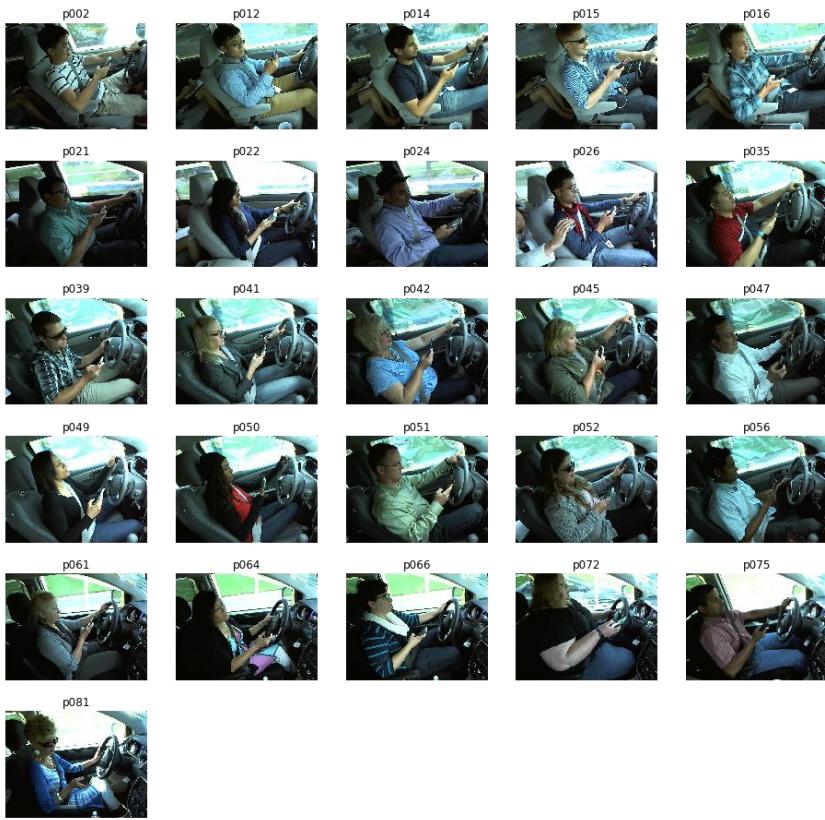
可见，每个状态下面的图片是比较均匀的。

下面是 10 种状态下每个状态下，每一个司机各选了一张照片，pxxx 就是司机的编号。

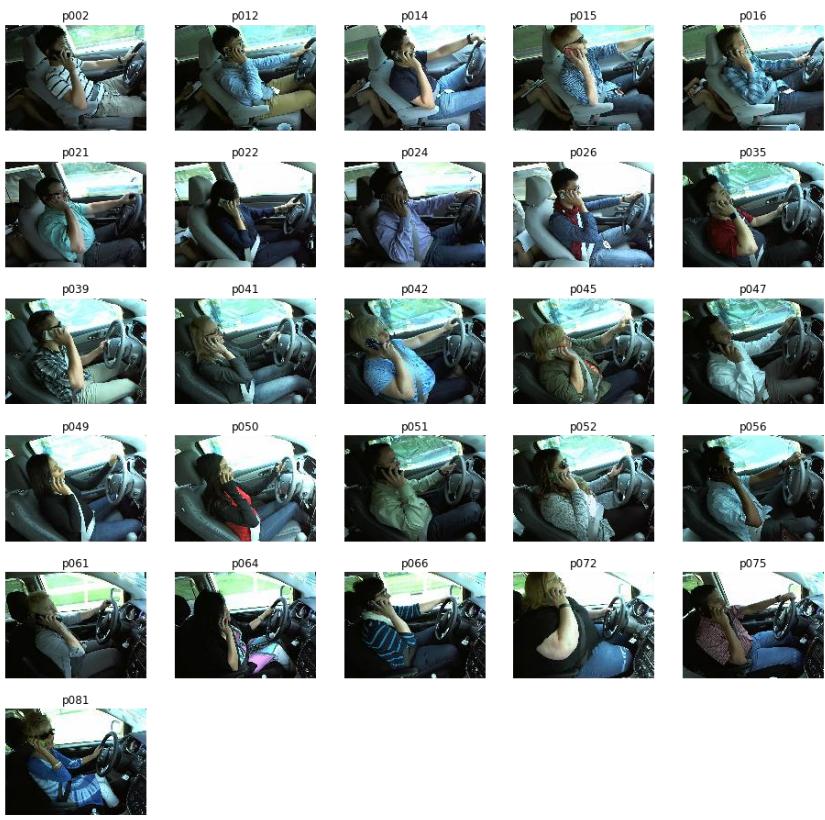
2.1.1. C0-安全驾驶



2.1.2. C1-右手打字



2.1.3. C2-右手打电话



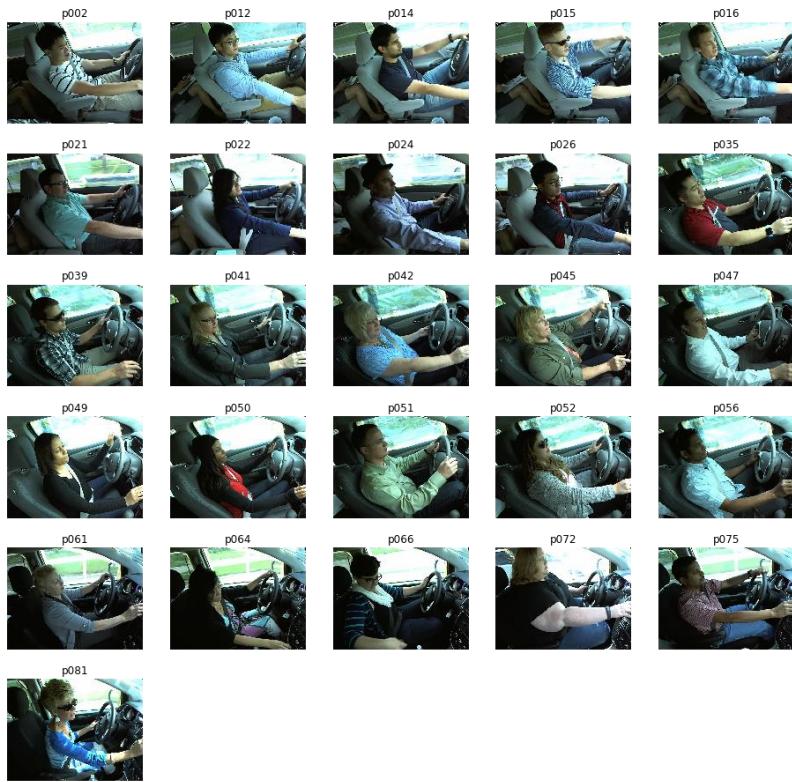
2.1.4. C3-左手打字



2.1.5. C4-左手打



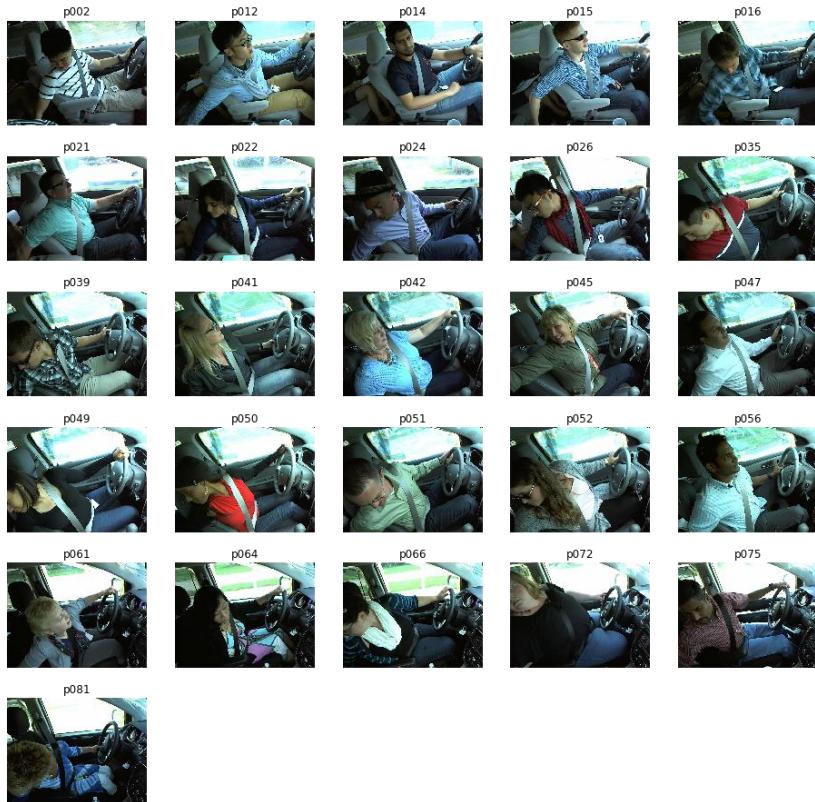
2.1.6. C5-操作收音机



2.1.7. C6-喝水



2.1.8. C7-拿后面的东西



2.1.9. C8-梳妆打扮



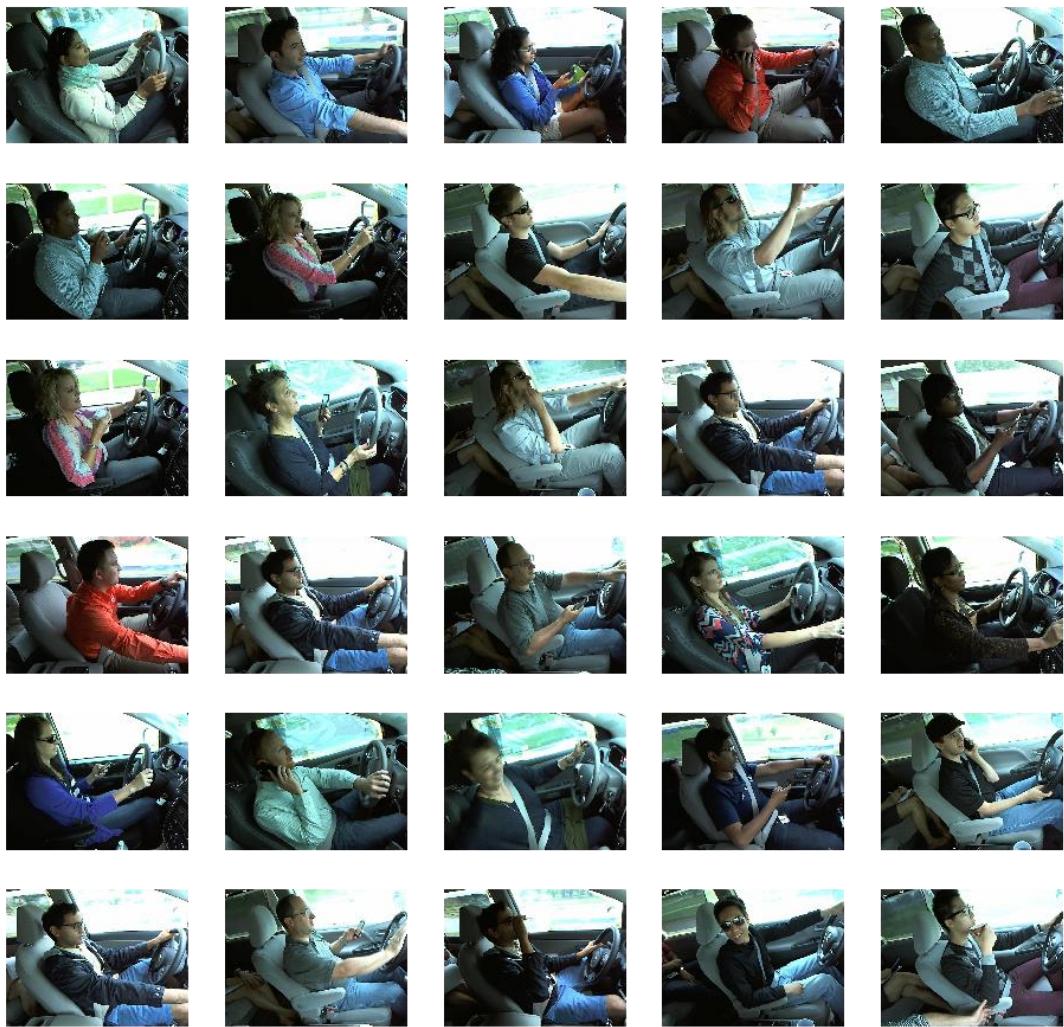
2.1.10. C9-和乘客说话



2.1.11. 每个训练数据集的司机的图片是连续的



2.1.12. 看看测试集的数据



2.1.13. 数据总结

基于以上数据的可视化，可以看出以下特点

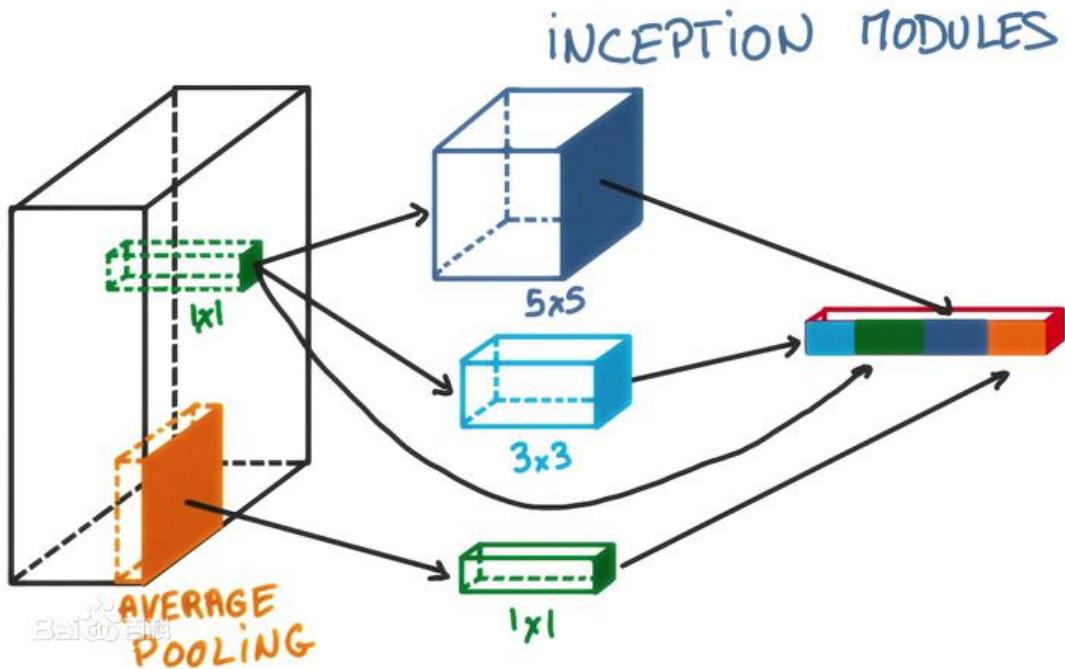
1. 训练数据和测试数据司机不同，但是车应该是一个车，不过摄像头拜访的位置，角度，差不多，但是是有差异的。所有后面要做预处理防止过拟合。
2. 车旁边玻璃上外的画面，是循环类似的，也就是每个人的都是依次操作响应的步骤来获取数据，车的行驶路径差不多，同一个动作的时候，窗外的画面是差不多，这一点可能会导致过拟合。
3. 训练数据的标注存在极为少数错误的，但是我没有好的办法批量筛出错误的，所以，只能简单肉眼基本看一遍，找到部分错误的，并且放到正确的分类下。
4. 训练数据就 26 个司机，测试数据的图片更多，司机更多，场景更多。这样看来训练数据是非常宝贵的。

2.2. 算法与技术

这是一个典型的图片分类器问题，将图片归类为 C0~C9

2.2.1. 卷积神经网络

我整体上采用卷积神经网络 CNN 来解决问题。卷积神经网络是神经网络的一种，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。现在世界上大部分图片处理和计算机视觉问题都是用卷积神经网络 CNN 来解决的。



2.2.2. Loss 函数的选择

我计划采用 categorical_crossentropy 的计算原理来使用 loss，下面介绍一下 categorical_crossentropy 的数学公式

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

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。

而常用的二分分类模型一般采用的是 binary_crossentropy 的方法，下面给出数学公式

$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

而我们面对的问题是有 10 个分类，而不止 2 个分类。应该采用 categorical crossentropy 的 loss 方法

以及它与 binary_crossentropy 的关系，并说明你使用该 loss 的理由

2.2.3. 优化器的选择

我介绍一些优化器的由来以及我为什么最终选择 Adam 来做主优化器，然后再用学习率极低的 RMSprop 来补充优化

首先介绍一下 Adagrad 优化器，Adagrad 其实是对学习率进行了一个约束。

$$n_t = n_{t-1} + g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

Adagrad 的特点是：

前期 g_t 较小的时候，regularizer 较大，能够放大梯度

后期 g_t 较大的时候，regularizer 较小，能够约束梯度

适合处理稀疏梯度

但也有明显的缺点：

由公式可以看出，仍依赖于人工设置一个全局学习率

设置过大时，会使 regularizer 过于敏感，对梯度的调节太大

中后期，分母上梯度平方的累加将会越来越大，使 gradient 趋于 0，使得训练提前结束。

Adadelta 优化器 是 Adagrad 的扩展，最初方案依然对学习率进行自适应约束，但是进行了计算上的简化。Adagrad 会累加之前所有的梯度平方，而 Adadelta 只累加固定大小的项，并且也不直接存储这些项，仅仅是近似计算对应的平均值。即：

$$n_t = \nu * n_{t-1} + (1 - \nu) * g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

在此处 Adadelta 其实还是依赖于全局学习率的，但是作者做了一定处理，经过近似牛顿迭代法之后：

$$E|g^2|_t = \rho * E|g^2|_{t-1} + (1 - \rho) * g_t^2$$

$$\Delta x_t = -\frac{\sqrt{\sum_{r=1}^{t-1} \Delta x_r}}{\sqrt{E|g^2|_t + \epsilon}}$$

其中，E 代表求期望。

此时，可以看出 Adadelta 已经不用依赖于全局学习率了。

Adadelta 的特点是：

训练初中期，加速效果不错，很快

训练后期，反复在局部最小值附近抖动

RMSprop 优化器，可以算作 Adadelta 的一个特例

$$\rho = 0.5 \quad E|g^2|_t = \rho * E|g^2|_{t-1} + (1 - \rho) * g_t^2$$

就变为了求梯度平方和的平均数。.

如果再求根的话，就变成了 RMS(均方根)：

$$RMS|g|_t = \sqrt{E|g^2|_t + \epsilon}$$

此时，这个 RMS 就可以作为学习率\eta 的一个约束：

$$\Delta x_t = -\frac{\eta}{RMS|g|_t} * g_t$$

RMSprop 的特点是：

RMSprop 依然依赖于全局学习率

RMSprop 算是 Adagrad 的一种发展，和 Adadelta 的变体，效果趋于二者之间

适合处理非平稳目标

Adam 优化器(Adaptive Moment Estimation) 本质上是带有动量项的 RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam 的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。公式如下：

$$m_t = \mu * m_{t-1} + (1 - \mu) * g_t$$

$$n_t = \nu * n_{t-1} + (1 - \nu) * g_t^2$$

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

$$\hat{n}_t = \frac{n_t}{1 - \nu^t}$$

$$\Delta\theta_t = -\frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon} * \eta$$

Adam 的特点是

1. 结合了 Adagrad 善于处理稀疏梯度和 RMSprop 善于处理非平稳目标的优点
2. 对内存需求较小
3. 为不同的参数计算不同的自适应学习率
4. 也适用于大多非凸优化 - 适用于大数据集和高维空间

由于 Adam 的这些特点，我选择 Adam 作为主优化器，在优化到差不多的时候，再采用 RMSprop 进行超低学习率的调优。

试验后发现，使用 RMSprop 比起 Adam 优化完成的，要好 0.03~0.05 个 accuracy，LogLoss 要好 0.05~0.2 个点。但是如果全部使用 RMSprop，收敛得比较慢，且也达到开始用 Adam 优化器训练的精度。

2.2.4. 迁移学习的选择

我选择使用迁移学习，迁移学习就是在直接用上 海量数据下已经学习好的成熟模型结果，在从基础上做些小的改造再根据实际问题重新训练从而完成学习的过程。

我选择迁移学习主要是两点原因考虑，首先，题目的数据集比较小，如果从头开始学习很容易学习不该学习的特征，其次就是学习的时间的考虑，我要节约使用 GPU 的时间，因为贵。

由于迁移学习是基于已经学习好的成熟模型的基础之上再次训练，神经网络大部分层次的特征都已经提取了，用较少的迭代次数就能够完成学习。所以迁移学习能够节省不少时间。

2.2.5. Fine-tune

我选择使用 Fine-tune，就是在迁移学习的基础上，不锁定 ImageNet 模型的部分 weights 的情况下训练，也就是 ImageNet 的 weights 部分层次锁定，部分层次重新训

练。之所以有这样考虑，是因为本项目的数据集的特征和 ImageNet 数据集的特征相当大，如果只训练全连接层，很难找到特征，所以要推到全连接层之前的层次去训练。

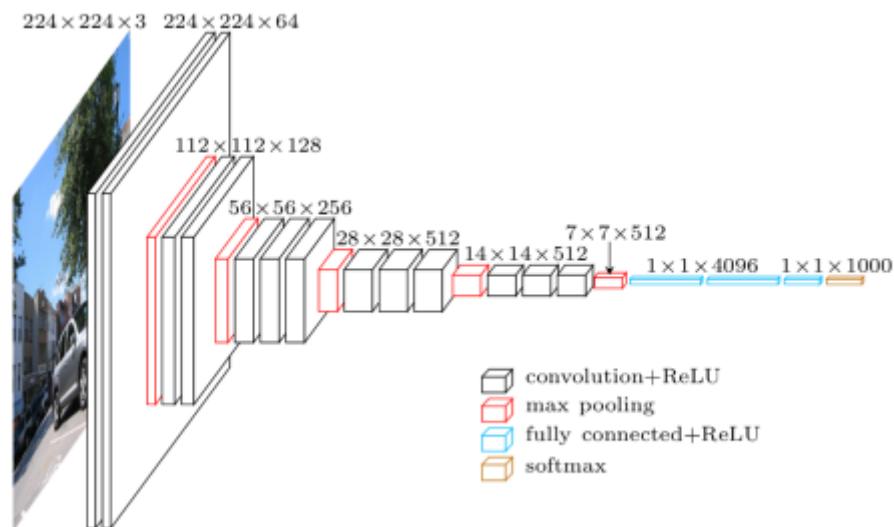
2.2.6. 多模型融合的选择

我选择采用多模型，因为不同模型的设计思路不同，提取图片特征的原理也不同，理论上使用多模型融合的方式能够提高精度，而我采用的融合方式不是求个平均值，而是把各个单模型 Fine-tune 的 weights 去掉全连接前的输出，串联起来，再通过神经网络训练来解决各个模型的权重。

下面我解决大致介绍一下各个单模型的特征和我的理解：

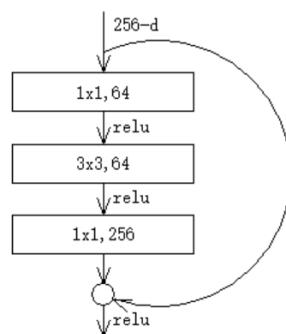
2.2.6.1. Vgg16

Vgg16 和 Vgg19 差不多，都是简单单向神经网络，类似下图所示：

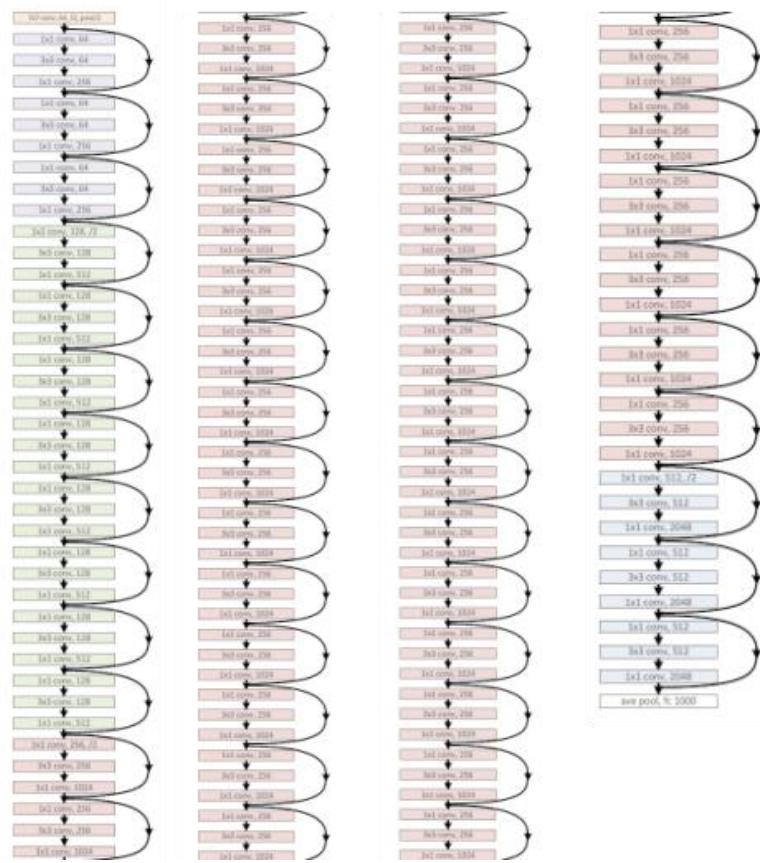


2.2.6.2. ResNet50

Resnet 的大致网络是这样的，简单地说，就是建立一个跨层的连接，这样就能训练很多很多层，Resnet 有各种层次的，ResNet50 是不大不小的一种
ResNet 的跨层单元的细节如下

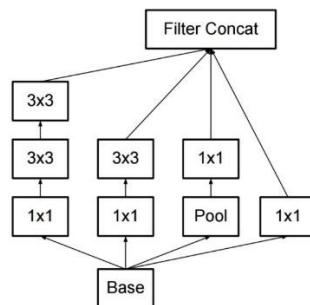


组成的整个网络的结构如下

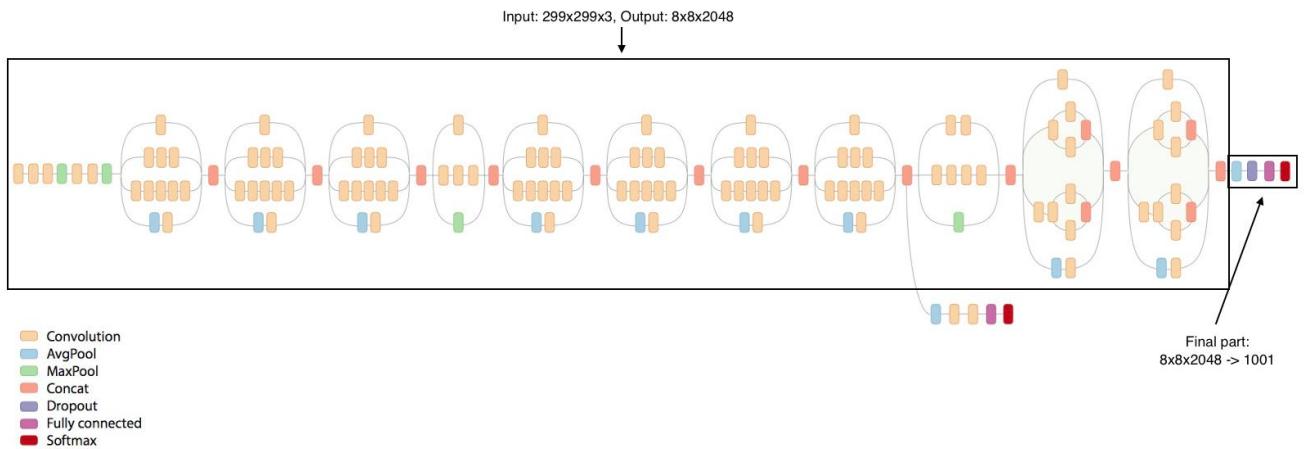


2.2.6.3. InceptionV3

简单来说，就是当你不知道使用 $1 \times 1, 3 \times 3, 5 \times 5$ 来卷积的时候，干脆把他们都用上，最后让神经网络来自适应出结果。

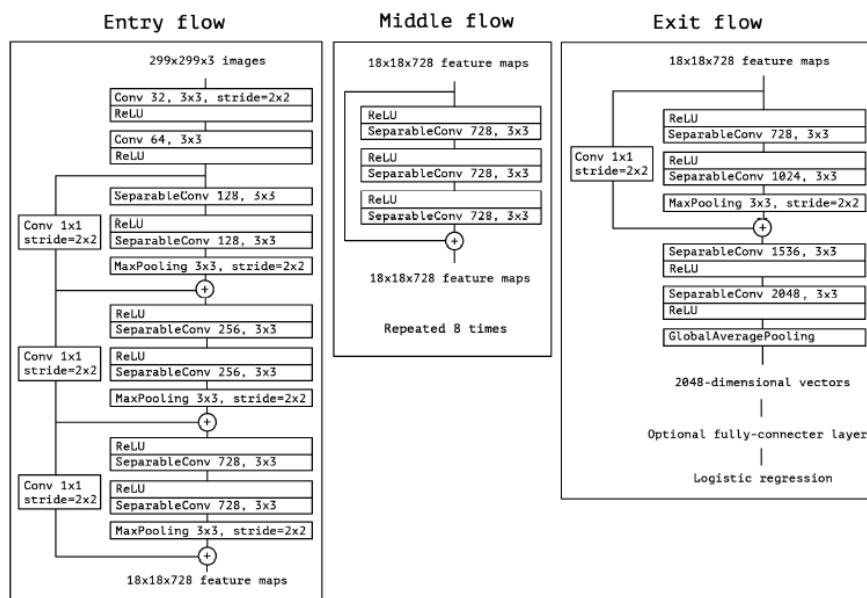


当多个这样的单元就组成了 Inception 的整体网络



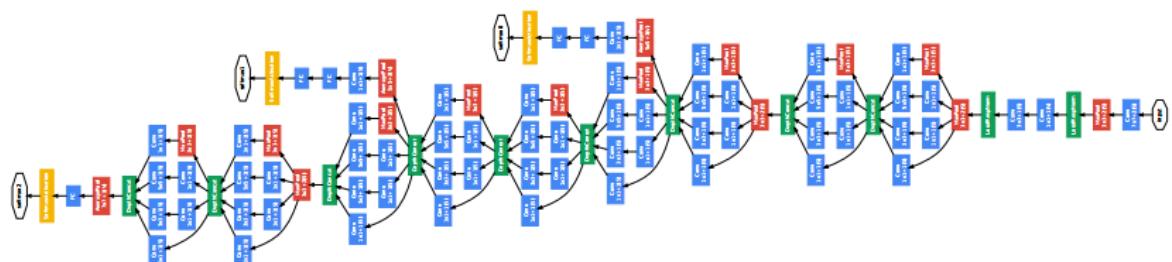
2.2.6.4. Xception

简单来说，Xception 就是有三种流，进入流 Entry flow，中间流 Middle flow，离开流 Exit flow。



<http://blog.csdn.net/xbinworld>

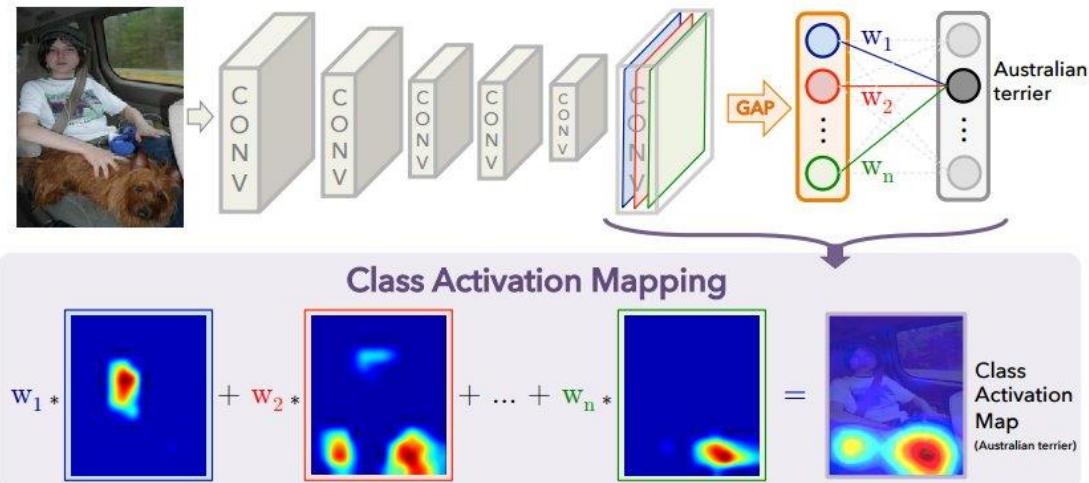
三种流交错进行，就组成了整个 Xception 网络



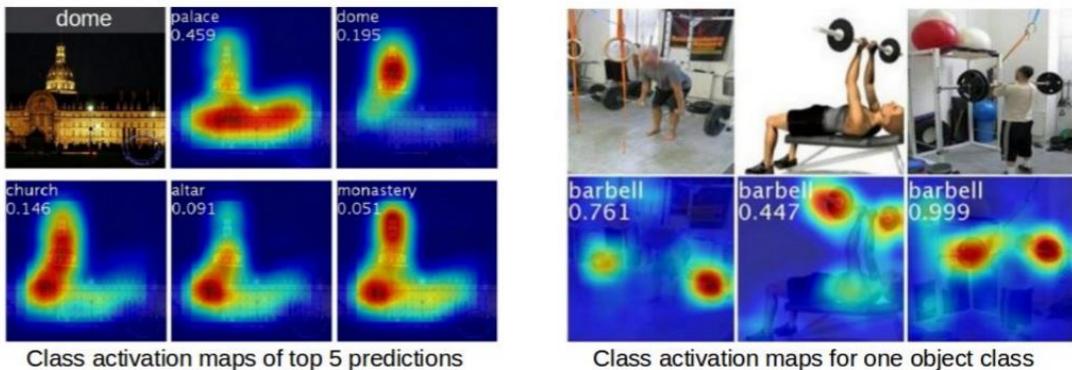
2.2.7. CAM 可视化

CNN 正在寻找什么以及如何改变视频中的注意力，我们采用类激活映射（ Class Activation Mapping ）和类特定显着图（ Class-specific Saliency Map ）

我们提出了一种使用 CNN 中的全局平均池（ GAP ）生成类激活映射的技术。 用于特定类别的类激活图表示 CNN 用于识别该类别的区别图像区域。 生成这些地图的过程如下所示。



类激活图可用于对 CNN 的预测决策进行透视。 左下图分别显示前 5 个预测的类激活图，可以看出，CNN 由图像的不同语义区域触发，用于不同的预测。 下图显示 CNN 学习本地化同一对象类的常见视觉模式。



此外，我们网络的深层功能可以用于通用本地化，使用最新训练的 SVM 的权重来生成类激活映射，然后可以免费获得类特定的显着映射。

随着神经网络训练次数的演化，重要的特征点对应的权重，通过一层层传递，会比较大。反之，不能作为特征的点，就会几乎没有权重，那么通过 CAM 图来说，权重大大的点会从红到蓝分布，越红，说明这个地方越是特征部位。

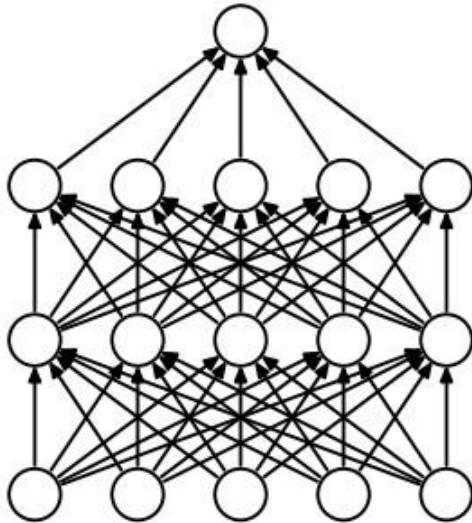
基于 CAM 图的好处，后续我每个单模型都会输出 CAM 以便于分析。

2.2.8. 参数优化

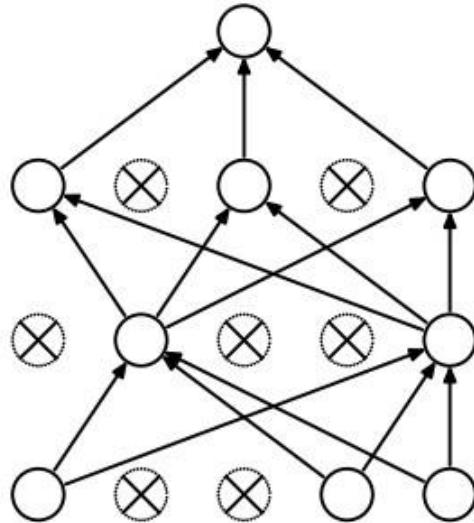
在项目使用了一些参数优化的技巧，大致如下：

2.2.8.1. 使用 Dropout 层

使用 Dropout 层在神经网络中的作用如下



(a) Standard Neural Net



(b) After applying dropout.

左图为全连接的神经网络，右图为随机丢弃后的神经网络。每次都会随机丢弃一些连接，因为每次都随机，所以实际上没根连接都隐隐约约最终还是起到了作用，

Dropout 能适当放置过拟合的原因大概有 2 点：

- 1 取平均的作用：先回到正常的模型（没有 dropout），我们用相同的训练数据去训练 5 个不同的神经网络，一般会得到 5 个不同的结果，此时我们可以采用“5 个结果取均值”或者“多数取胜的投票策略”去决定最终结果。这种“综合起来取平均”的策略通常可以有效防止过拟合问题。因为不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消。dropout 掉不同的隐藏神经元就类似在训练不同的网络（随机删掉一半隐藏神经元导致网络结构已经不同），整个 dropout 过程就相当于对很多个不同的神经网络取平均。而不同的网络产生不同的过拟合，一些互为“反向”的拟合相互抵消就可以达到整体上减少过拟合。
- 2 减少神经元之间复杂的共适应关系：因为 dropout 程序导致两个神经元不一定每次都在一个 dropout 网络中出现。（这样权值的更新不再依赖于有固定关系的隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况）。迫使网络去学习更加鲁棒的特征（这些特征在其它的神经元的随机子集中也存在）。换句话说假如我们的神经网络是在做出某种预测，它不应该对一些特定的线索片段太过敏感，即使丢失特定的线索，它也应该可以从众多其它线索中学习一些共同的模式（鲁棒性）。（这个角度看 dropout 就有点像 L1, L2 正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高）

基于以上的原因，最终我决定采用 0.5 概率的 Dropout 来最终的 Dense 层之前。

2.2.8.2. 使用 clip 来防止 logloss 分数过高

这里我们用到了一个小技巧，我们将每个预测值限制到了 $[0.005, 0.995]$ 个区间内，这个原因很简单，kaggle 官方的评估标准是 LogLoss，对于预测正确的样本， $\log(0.995) = -0.005$ 和 $\log(1) = 0$ 相差无几，但是对于预测错误的样本， $\log(0) = -\infty$ 和 $\log(0.005) = -5.30$ 的差距非常大。而根据公式

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

限制在 $[0.005, 0.995]$ 之间对最终的 Kaggle 得分是有一定好处的。

2.3. 解决问题的步骤和思路

这是一个典型分类器问题，预测的时候是将图片进行归类 C0~C9

首先，将 train 数据，划分为训练和验证集。由于训练集的司机和测试集的司机不是一个人，所有在划分训练和验证集的时候，要区分司机，把一部分司机的数据做为训练集，把另外部分司机的数据作为验证集。另外由于数据的特性，如 2.1.13，需要做一下数据增强，如做一下旋转，平移，放缩等来增加数据的多样性，从而有效防止过拟合。

第二步，测试基准模型的结果，再一次我选择 VGG16 的 imagenet 迁移学习模型来作为基准模型，然后只重新训练全连接层，从而得到一个基准模型的结果，汇出基本模型的 CAM 类激活图，以便于后续模型于此对比。

第三步，依次尝试 Fine-tune 的其他单模型的 imagenet 迁移学习，如 ResNet50，InceptionV3，Xception 等，得到每个 Fine-tune 单模型的结果，然后针对每个单模型进行优化。把每个模型都优化到差不多了，进入下一步。

第四步，单模型优化完成后，开始做多模型融合，并优化融合模型，得出最终结果。然后生成 pred 文件，提交到 kaggle 上看最后的得分。

3. 实现

3.1. 在训练数据中区分训练集和验证集

由于数据有这个特点：训练集司机完全不是测试集司机，所以在训练数据中区分训练集和验证集的时候，不能简单地随机分离，也要区分司机，不然很容易出现过拟合。

【另外，我试过，如果训练集和测试集，如果不区分司机的化，直接简单随机拆分的化，最后看似精度能做到很高，但是提交到 kaggle 后，分数非常低，也就是验证集得出的验证结果会严重失真】

于是我，抽取了司机编号为 p081 和 p075 这两个司机的数据，作为验证集使用，其他司机都作为训练集使用。

这样，训练集的数据有 20787 条，验证集的数据有 1637 条，这样训练集和验证集的比例为大约 12.6 : 1。

这部分的代码在 `split_valid.py` 中。

3.2. 数据预处理

由于摄像头的画面差异，位置和角度有些差异，司机可能也调整了座椅和方向盘的位置，而且训练数据非常少，为了防止后面的过拟合，我再训练数据之前，要对训练数据做数据增强。也就是随机话处理，也就是每张画面都要做随机的平移，放缩，旋转等，从而增加了数据的多样性。避免后面在神经网络学习的时候，学到不必要的特征，从而产生过拟合。

我的数据预处理，采用 Keras 里面的 `ImageDataGenerator` 来实现。

```
train_gen = ImageDataGenerator(  
    ...,  
    rotation_range=10.,  
    width_shift_range=0.05,  
    height_shift_range=0.05,  
    shear_range=0.1,  
    zoom_range=0.1,  
)
```

做了随机化之后，再进入神经网络，最后实践证明，随机化比起不随机化，要好不少。

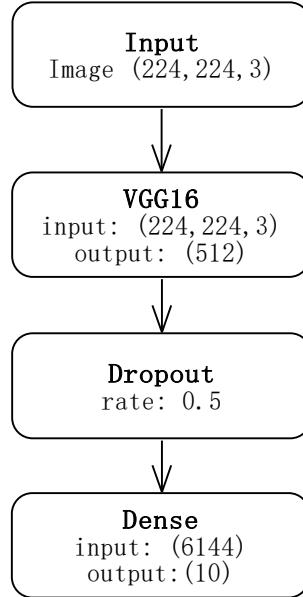
这部分的代码在

`kerasa-vgg16-visual-finetune.ipynb`，`kerasa-resnet50-visual-finetune.ipynb`，`kerasa-xception-visual-finetune.ipynb`，`kerasa-inceptionV3-visual-finetune.ipynb`，中，也在生成融合模型 python 代码 `write_bottleneck_with_fine_tune.py` 中都有。

3.3. 基准模型评估

这部分的代码在 keras-vgg16-visual-finetune.ipynb

为了最终的优化和对比，我采用一个统一的全连接层，我采用的全连接层如下。



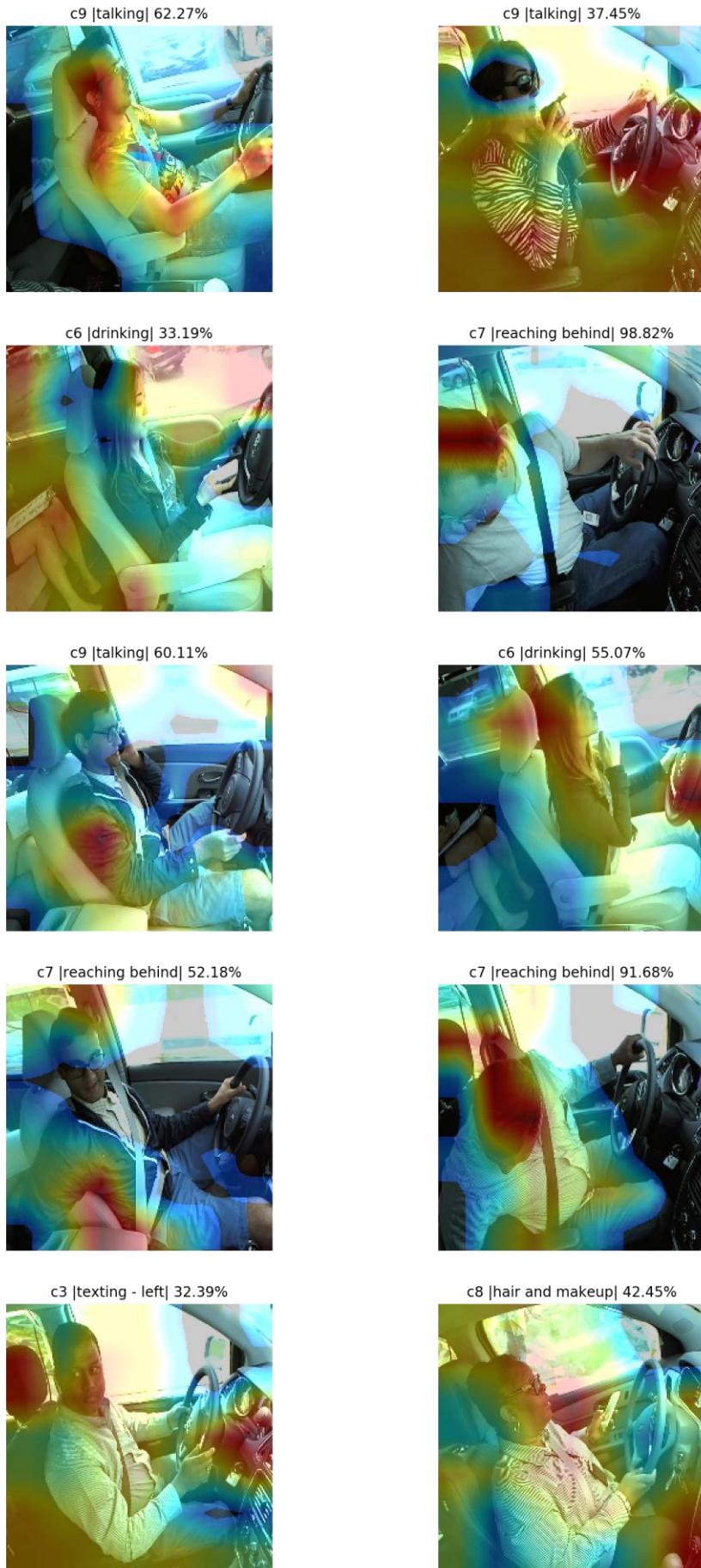
ResNet 的输入模型时，图放缩到大小为 (224, 224, 3)
其去掉全连接层之后的输出是 长度为 2048 的向量。

基准模型只训练了上图中添加的全链接层，没有做 Fine-tune。

在迭代了 19 次之后的结果是：

```
loss: 1.4377 - acc: 0.5305  
val_loss: 1.2111 - val_acc: 0.6449
```

查看 CAM 类激活图如下：



可以看出，基准模型的结果并不理想。后面说说我的方案吧。

3.4. 单模型 Fine-tune 及优化

我这里尝试了对 Vgg16 , Vgg19 , ResNet50 , InceptionV3 , Xception 做 Fine-tune 并优化，最后结果 ResNet50 , InceptionV3 , Xception 都成功的，即训练集和测试集的 accuracy 都做到 0.9 左右；但是 Vgg16 和 Vgg19 都没有成功，也就是很长时间的训练，accuracy 最多只能到 0.6 左右，且尝试过不同的层次做 fine-tune，均没有成功。

3.4.1. ResNet50 模型 Fine-tune

具体的实现见代码：keras-resnet50-visual-finetune.ipynb

经过各种尝试和优化，发现以下几点是最优参数

1. 图片缩小到(240, 320, 3)，可能因为很多场景动作很小，缩太小不利于提出特征。且反复尝试，保持图片比例不变进入神经网络的结果却是最好。
2. 现用 Adam 先训练 6 轮，再用 RMSprop 用极小的学习率 0.00001 训练 6 轮。
3. 多次实验，发现在第 152 层的时候开始 tune 效果最好，就是锁定 0-151 层的权重，从 152 层起，权重是可以训练的。

ResNet 的输入模型时，图放缩到大小为 (240, 320, 3)

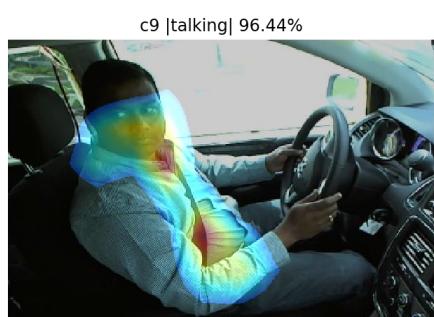
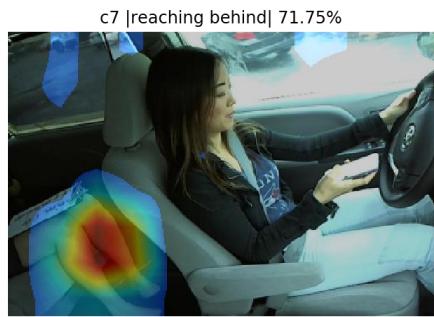
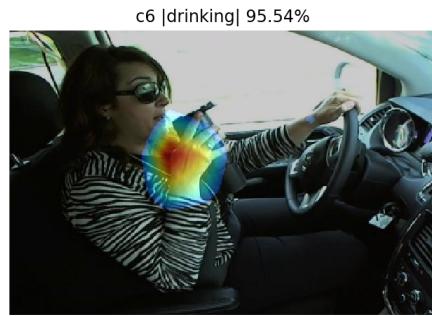
其去掉全连接层之后的输出是 长度为 2048 的向量。

在最后迭代后的结果是：

loss: 0.0038 - acc: 0.9987

val_loss: 0.7260 - val_acc: 0.8596

查看 CAM 类激活图如下：



3.4.2. InceptionV3 模型

具体的实现见代码 : keras-inceptionV3-visual-finetune.ipynb

经过各种尝试和优化 , 发现以下几点是最优参数

1. 图片缩小到(360, 480, 3) , 可能因为很多场景动作很小 , 缩太小不利于提出特征。
且反复尝试 , 保持图片比例不变进入神经网络的结果却是最好。
2. 现用 Adam 先训练 4 轮 , 再用 RMSprop 用极小的学习率 0.00001 训练 6 轮。
3. 多次实验 , 发现在第 172 层的时候开始 tune 效果最好 , 就是锁定 0-171 层的权重 , 从 172 层起 , 权重是可以训练的。

InceptionV3 的输入模型时 , 图放缩到大小为 (360, 480, 3)

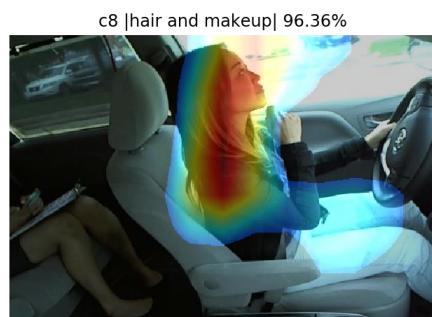
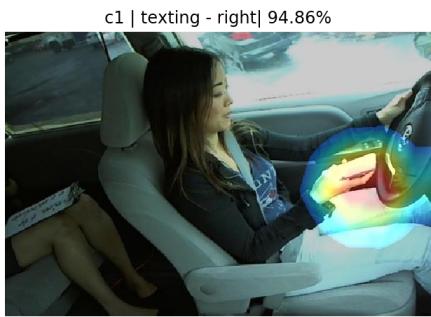
其去掉全连接层之后的输出是 长度为 2048 的向量。

在最后迭代后的结果是 :

loss: 0.0016 - acc: 0.9995

val_loss: 0.3046 - val_acc: 0.9376

查看 CAM 类激活图如下 :



这批数据看起来，特征分析得相对比较准了。

3.4.3. Xception 模型

具体的实现见代码：keras-xception-visual-finetune.ipynb

经过各种尝试和优化，发现以下几点是最优参数

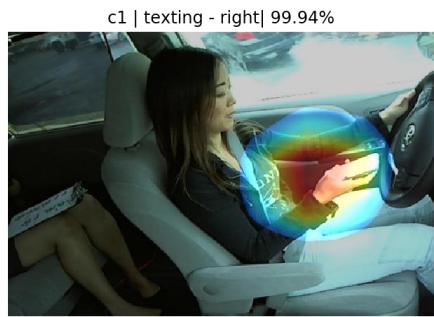
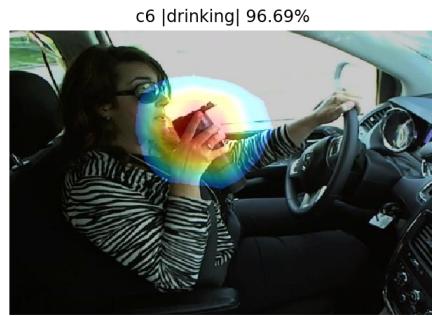
1. 图片缩小到(360, 480, 3)，可能因为很多场景动作很小，缩太小不利于提出特征。
且反复尝试，保持图片比例不变进入神经网络的结果却是最好。
2. 现用 Adam 先训练 4 轮，再用 RMSprop 用极小的学习率 0.00001 训练 6 轮。
3. 多次实验，发现在第 172 层的时候开始 tune 效果最好，就是锁定 0-171 层的权重，从 172 层起，权重是可以训练的。

Xception 的输入模型时，图放缩到大小为 (360, 480, 3)
其去掉全连接层之后的输出是 长度为 2048 的向量。

在最后迭代后的结果是：

```
loss: 0.0095 - acc: 0.9973  
val_loss: 0.4296 - val_acc: 0.9036
```

查看 CAM 类激活图如下：



3.5. 多模型融合

3.5.1. 之前模型的汇总

从计算结果上看，大致如下：

首先对比各个模型的运行结果如下，参数都是

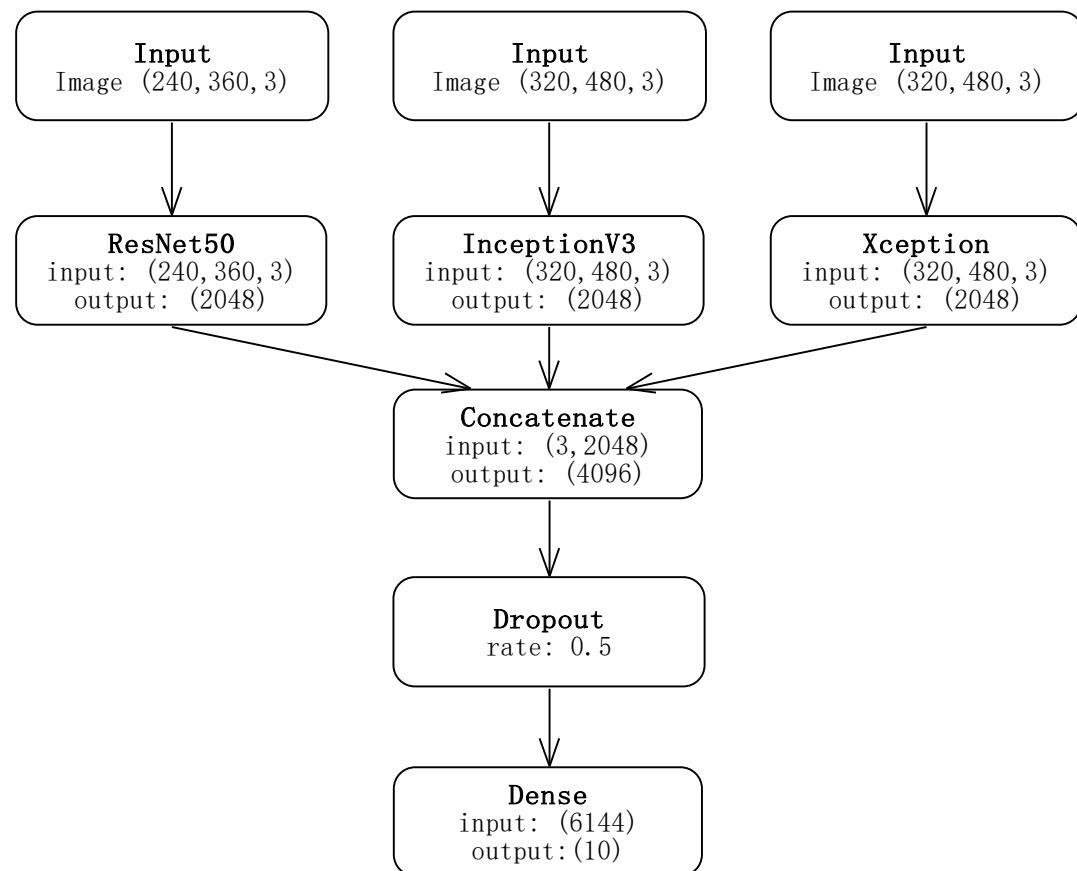
模型	训练 Accuracy	验证 Accuracy	训练 Loss	验证 Loss	Kaggle Logloss Public Score	Kaggle Logloss Private Score
基准模型 VGG16	0.5305	0.6449	1.4377	1.2111	1.92176	1.64561
ResNet50	0.9987	0.8596	0.0038	0.7260	0.45330	0.58270
InceptionV3	0.9995	0.9376	0.0016	0.3046	0.32430	0.35211
Xception	0.9973	0.9036	0.0095	0.4296	0.33950	0.37661

后面把以上三个模型 Merge 在一起

3.5.2. 新的模型，混合模型

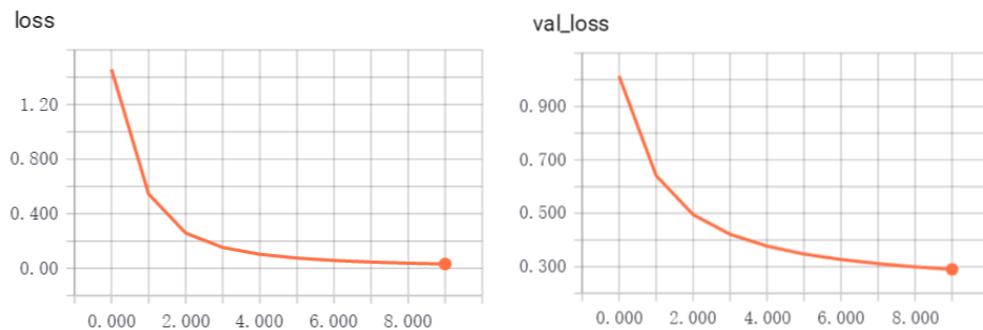
具体的实现见代码：main-finetune.ipynb

新的模型把 ResNet，InceptionV3 和 Xception 混合起来做，从而完成最终模型

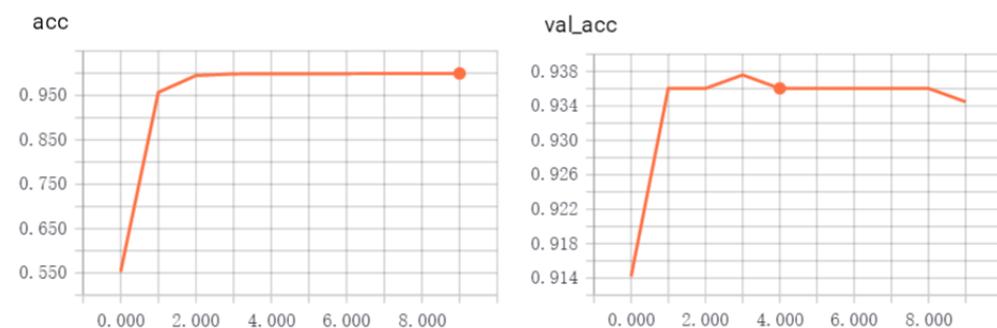


新的模型把 ResNet , InceptionV3 和 Xception 的去掉全链接层的结果混合起来 , 然后重新用神经网络训练全链接层。最后的在 Adam 优化器下迭代 10 轮之后
(使用混合模型运行在 tensorboard 下的图表如下 :)

Loss 图表 :



Accuracy 图表 :



可见 , loss , Accuracy 都处于收敛状态 , 停止时机比较合适

最终结果如下 :

loss: 0.0317 - acc: 0.9995

val_loss: 0.2900 - val_acc: 0.9345

将其结果提交到 Kaggle 后 , 得到的分数如下 :

0.25778

0.29588



3.5.3. 结果

本地验证结果看 :

验证集 Loss : 0.3615

验证集 Accuracy : 0.9314

将结果提交到 Kaggle , 得到的分数是:

Public Score: 0.25778 排名 : 148 / 1440 (top 20%)

Private Score: 0.29588 排名 : 189 / 1440 (top 20%)

其结果对比起基准模型

模型	训练 Accuracy	验证 Accuracy	训练 Loss	验证 Loss	Kaggle Logloss Public Score	Kaggle Logloss Private Score
基准模型 VGG16	0.5305	0.6449	1.4377	1.2111	1.92176	1.64561
ResNet50	0.9987	0.8596	0.0038	0.7260	0.45330	0.58270
InceptionV3	0.9995	0.9376	0.0016	0.3046	0.32430	0.35211
Xception	0.9973	0.9036	0.0095	0.4296	0.33950	0.37661
Mix 混合模型	0.9995	0.9345	0.0317	0.2900	0.25778	0.29588

4. 结论

4.1. 总结

最终模型的结果达到了我之前设定的 Accuracy > 0.93, Loss < 1.0, 世界排名在前 1/5 的目标。

但是由于我自身机器性能问题，和业务学习不多没有足够时间训练数据，没有做得更好。 不过我会继续改进算法，后续改进思路大概如下。

4.2. 后续改进

关于后续改进有几点想法：

4.2.1. 全面彻底清理训练数据异常数据

训练数据集中有些错误标记，对结果有些影响，要想办法清理掉这些。

4.2.2. 可以参考前后几帧图片来判断

我现有的方式，是把每一帧数据当作完全独立的图片来处理的，但是如果参考前后每帧的画面内容，想办法用起来，对结果应该是有些帮助的。而且如果最终在车上实地部署的化，新数据预测的时候也可以采用关联前后帧的方式来做，这样应该会有更高的精度。

后续我有更多时间，还会继续优化和尝试，把分数做得更高。

参考文献

Very Deep Convolutional Networks for Large-Scale Image Recognition [Karen Simonyan, Andrew Zisserman] (Submitted on 4 Sep 2014 (v1), last revised 10 Apr 2015 (this version, v6)) <https://arxiv.org/abs/1409.1556>

Deep Residual Learning for Image Recognition [Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun] (Submitted on 10 Dec 2015) <https://arxiv.org/abs/1512.03385>

Rethinking the Inception Architecture for Computer Vision [Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna] (Submitted on 2 Dec 2015 (v1), last revised 11 Dec 2015 (this version, v3)) <https://arxiv.org/abs/1512.00567>

Going Deeper with Convolutions [Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich] (Submitted on 17 Sep 2014) <https://arxiv.org/abs/1409.4842>

Learning Deep Features for Discriminative Localization [Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba] (Submitted on 14 Dec 2015) <https://arxiv.org/abs/1512.04150>

ADADELTA: AN ADAPTIVE LEARNING RATE METHOD
[Matthew D. Zeiler] (Submitted on 22 Dec 2012) <https://arxiv.org/abs/1212.5701>

手把手教你如何在 Kaggle 猫狗大战冲到 Top2% [杨培文] (发表于 2017-03-18)
<https://ypw.io/dogs-vs-cats-2/>

使用 Keras 来破解 captcha 验证码 [杨培文] (发表于 2017-03-07)
<https://ypw.io/captcha/>

Kaggle 求生：亚马逊热带雨林篇 [刘思聪] (发表于 2017 年 7 月 31 日)
https://zhuanlan.zhihu.com/p/28084438?utm_source=itdadao&utm_medium=refer_ral

