机器学习工程师纳米学位毕业项目

基于深度学习的猫狗图像识别（“猫狗大战”）

2018年4月26日

# 问题定义

## 项目概述

“猫狗大战”项目涉及的领域是计算机视觉，具体的说是“图像识别”问题。计算机视觉这个研究领域最早目的是为了研究如何让智能机器人像人类一样拥有高级视觉系统，能“看到和识别”外部世界中的各种物体。之所以选择这个项目是因为我本人对计算机视觉这个研究领域非常感兴趣，希望以后能将相关知识应用于智慧农业，特别是计算机自动识别常见的农业病虫害。

该项目是Kaggle上面的挑战项目之一，该项目自带了与项目有关的数据集，该数据集包含25000张图片组成的训练集和12500张图片组成的测试集。该项目将使用训练集图片训练一个基于深度学习的算法模型，然后在测试集图片上评估其准确率。

## 问题描述

这里将要解决的问题是训练一个算法模型来识别图片中是猫还是狗，是一个“二分类”问题。该问题对于人类而言并不是什么大问题，但对计算机而言却很困难，因为计算机“看到”的是由数字组成的点阵，它要分辨出这样的数字点阵是什么物体是很困难的。然而，像这样的图像识别问题已经有了很多效果不错的解决方案，深度学习就是其中之一。我在这里将结合深度学习中的卷积神经网络提出一个自己的解决方案。

这里准备采取的策略是“迁移学习+模型融合”，因为就卷积神经网络而言，迁移学习是一种很好的策略，特别是应用于图像识别领域，这是由卷积网络的分层架构决定的，网络中的前几层能识别图像中的一些简单的图案，比如边缘等等，这些往往是每个图像识别问题所共有的特征，没必要从头训练，可以复用并节约解决问题的时间，像ImageNet这样包含1000种分类的的超大规模图片数据集已经预训练了几种常见的卷积神经网络模型，可以直接使用。此外，对多个模型的输出结果进行融合，能够博采众长，兼听则明，有效提高预测的准确度，比只使用单个模型要好。最后得到的预测结果为一个概率值p，代表该图片是狗的概率。预期的结果是对于狗的照片，该值接近于1；而对于猫的照片，该值接近于0。

## 评价指标

本项目要得到的最终结果是一个二分类问题，所以这里将用准确率结合二元交叉熵损失函数作为算法性能好坏的评估指标，将根据训练集和测试集的损失函数表现来评估算法性能。如果验证集损失还在下降，那么需要增加模型复杂度或者多训练几代；如果验证集损失上升，则出现过拟合，需要正则化或Dropout防止过拟合；如果验证集的损失出现震荡，则需要减小学习率；如果验证集的损失趋于稳定，则可以减少训练代数。

* C：损失函数（Cost Function）；
* n：数据集数量；
* y：分类为狗（y=1）或猫（y=0）；
* p：模型预测分类为狗的概率。

# 2. 分析

## 2.1 数据的探索

训练集包括25000张图片，都为JPEG格式，是按照“猫/狗.编号”来命名的，比如“cat.0.jpg”。图片包含猫和狗的各种姿态的照片，排在前面的12500张图片全部是猫，剩下的全部是狗。通过对图片的大致浏览，发现某些图片还包含有人类，如图1所示。有的图片则同时出现了狗和猫，但被标记为猫，如图2所示。甚至还发现了异常值，如图3所示，一张人类的照片被标记为了猫，类似这样的异常值肯定会对分类准确性造成一定影响。



图1 人类抱着猫咪的图片



图2 同时出现猫和狗的图片

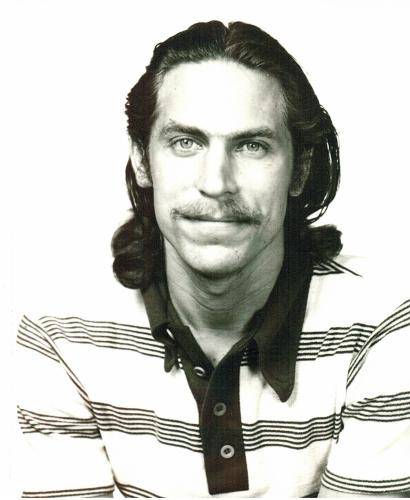


图3 一张人类的图片被标记为猫

另外，图片的尺寸不一致，在输入到神经网络之前，应该调整图片的大小。具体地，需要按照神经网络输入层的要求对图片进行resize操作。训练集数据需要进一步分为训练集和验证集并打乱顺序，这可以提高模型在测试集上的泛化能力。测试集则包含12500张图片，以数字编号，猫狗的出现顺序已随机打乱。

## 2.2 探索性可视化

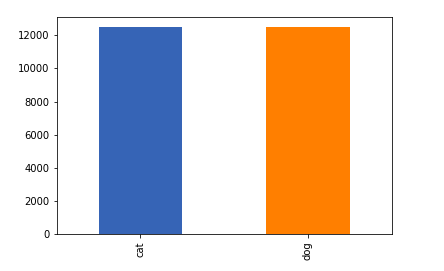


图4 猫和狗的图片各占比例

因为训练集的图片是已经标注过的，所以这里对训练集图片进行可视化，看看猫和狗的图片各占多大的比例，得到的柱状图如图4所示，可以看到训练集中猫和狗的图片各占一半，分别为12500张。但是根据上面的分析，训练集中实际上是存在一些异常值的。在稍后的数据预处理过程中会对异常图片进行删除，所以在预处理过后还需要对清洗过的数据再次进行可视化，观察清洗后的情况。

## 2.3 算法和技术

这里准备采用已在ImageNet上预训练过的4种卷积神经网络模型：VGGNet[1]，ResNet[2]，Inception v3[3]和Xception[4]来实现猫狗识别算法。ImageNet拥有一千多万张图片，对1000种物品进行分类，其中就有118种狗类和7种猫类，由它所导出的特征向量能高度概括图像中包含了哪些内容。首先，先分别去掉这四个模型的顶层（即全连接层），然后分别用25000张训练集图片在4个模型上进行预测，输出并保存得到的特征向量，得到4个特征向量文件；分别读取这些特征向量文件，将它们融合成一个特征向量；添加自己的全连接层，然后对模型进行编译构建；最后使用融合特征向量训练模型，并对测试集进行预测得到预测结果。

一开始我也不是很清楚应该选择哪几种卷积神经网络模型，后来查阅了Keras文档中“Documentation for individual models”[5]，这部分文档介绍了常见的几种模型，如图5所示，它对比了不同模型在大小，Top-1和Top-5准确度，参数数量以及深度之间的区别。我决定选择上述这4种模型的组合，因为它们的准确率都还比较理想。虽然它们不是表现最好的模型，但我采用的解决方案重点在于Ensemble，我想验证的想法是多种模型的融合能够比单个模型的表现更佳，所以我没有考虑其他准确率更高的模型，尽管采用这些模型最后得到的成绩会更好些。

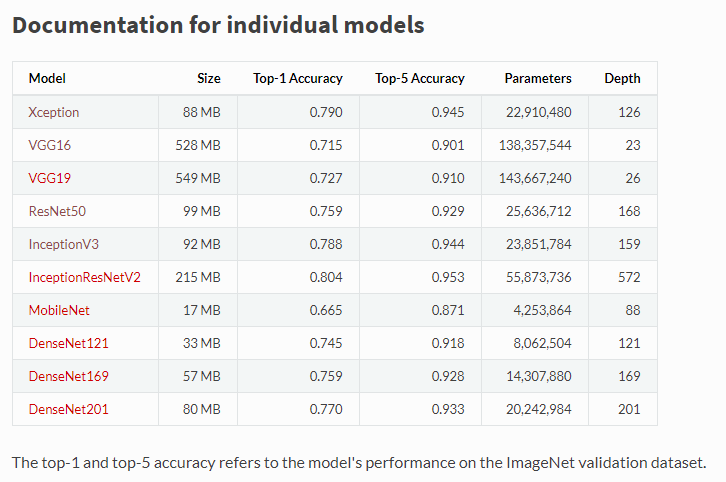


图5 常见卷积神经网络模型介绍（Keras）

[卷积和算法原理]

## 2.4 基准模型

按照毕业项目要求，我最后所得到的模型要能进入Kaggle排行榜前10%，即在Public Leaderboard上LogLoss值低于0.06127。

# 3. 方法

## 3.1 数据预处理

为了提高算法的性能，有必要对上面“数据的探索”部分提到的异常值进行检测和删除。采用的方法是使用ImageNet预训练过的上述4种模型来检测图片中存在的异常值。通过研究清单[5]可以知道ImageNet包含1000种分类，有猫和狗的品种，其中猫有7个品种，狗有118个品种。查阅Keras文档[6]可知准确率最高的模型是Xception，其Top-1和Top-5的准确率分别为0.790和0.945。这里的Top-N准确率是指在预测得到的概率值中，前N个结果包含正确值的占比。这里的关键在于通过不断试验选取一个合适的Top-N值，来尽可能多的检测出异常值并降低误报率。

首先，从Xception入手，选取Top-5值，并以1000张图片为样本，检测出了15张图片，对这些图片进行人工检查，发现很多正常图片被识别为异常值，效果不是很理想。然后把Top值提高到10，20和30进行试验，并适当提高样本量，开始检测出一些异常图片，例如图5这样的。通过不断地试验和观察，最后发现Top-60是一个比较理想的值。



图6 非猫非狗的图片

找到合适的Top值后，下面就是结合4种模型分别对全体训练集进行检测，综合4种模型输出的结果得到118张图片，通过人工分析发现这些被判定为异常值的图片有以下几种情况：

1. 图片中有猫狗，但尺寸太小，不够清晰；
2. 图片中有猫狗，但图片内容比较复杂；
3. 图片非猫非狗；
4. 图片是猫和狗的卡通形象；

我认为情况1应该保留，因为这里不能假定所有输入模型的图片都是清晰的，总会有一些模糊的图片，模型应当对这样的图片具有一定的健壮性；情况2也应该保留，模型也应该对一些复杂的图片具有一定的健壮性；情况3毫无疑问应该删除；情况4则比较主观，但我训练该模型的目的是为了识别真实世界的猫狗，所以我选择删除。最终被删除的有38张图片。清洗后的训练集图片有24962张，其中猫有12483张，狗有12479张。

## 3.2 执行过程

为了方便之后使用Keras的ImageDataGenerator作为图片数据生成器，在训练模型之前首先要针对训练数据和测试数据重构目录结构。具体做法是创建trains和tests两个文件夹，结构如下：

* trains
* dogs：包含训练集中所有为狗的图片
* cats：包含训练集中所有为猫的图片
* tests
* test：包含测试集中所有的图片

为了避免浪费磁盘空间，新创建的文件夹中的所有图片都为原始图片的符号链接文件（symbol link）。

我采用的方法是迁移学习中的“导出特征向量”法，使用ImageNet预训练过的4种模型，去掉它们的全连接层（include\_top=False），只保留卷积层，使用ImageDataGenerator读取重构过的目录结构获得训练生成器和测试生成器，以batch size等于64为单位进行预测，然后将预测得到的特征向量分别导出为4个h5文件：

* bottleneck\_features\_resnet.h5
* bottleneck\_features\_xception.h5
* bottleneck\_features\_inception.h5
* bottleneck\_features\_vgg16.h5

这种方法基于预训练模型进行“特征工程”，将抽取到的特征向量保存下来，然后再根据特征向量创建自己的全连接层，最后编译构建自己的模型。

紧接着，读入刚才导出的全部特征向量，做合并和随机洗牌，就得到了训练集X\_train（24962\*6656）和y\_train（24962\*1），以及测试集X\_test（12500\*6656）。有了特征向量，构建模型的过程就比较简单了，只需要一层Dense层，激活函数为sigmoid，采用Adadelta作为优化算法，二元交叉熵为损失函数，以准确率为评估指标，然后在训练集上训练100代，其中抽取20%作为验证集。

执行过程中还是遇到了一些困难的。由于对Keras库不熟悉，在导出特征向量的过程中执行predict\_generator()函数提供了错误的steps参数，导致训练非常慢，Resnet50上用了5个多小时，导出的特征向量文件有19G，后来查了些资料并向其他同学请教问题，发现自己的steps设定有误，于是修改了这个bug，后来每个模型导出的时间只需要5分钟左右，且特征向量文件在375M左右。导出特征向量的关键函数如图6所示。

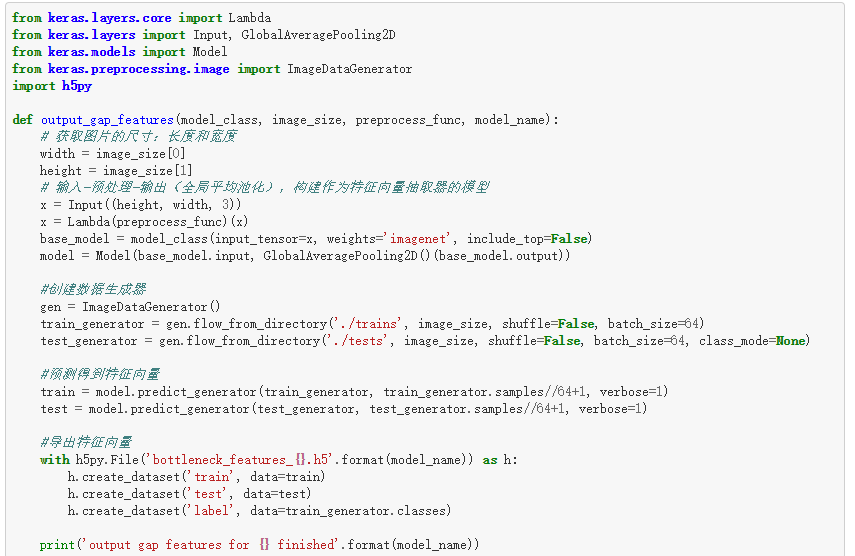


图7导出特征向量的关键函数

## 3.3 完善

最初构建的模型如图7所示。它只是增加了一层Dense层，激活函数为sigmoid，其他没有做任何处理，得到的损失函数曲线如图8所示，其中红色为验证集，蓝色为训练集。

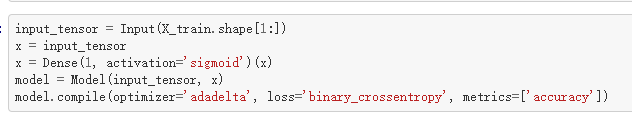


图8最初训练的模型

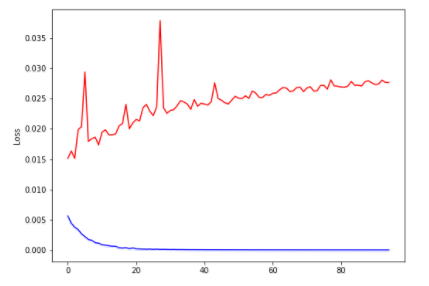


图9 最初训练模型的损失函数曲线

从图9可以看出验证集的损失函数曲线呈明显的上升趋势，说明模型出现了过拟合，需要通过正则化或者Dropout防止过拟合，添加p=0.25的Dropout层进行试验，发现还是过拟合，于是将p增大到0.5再次进行训练。图10展示了核心代码。

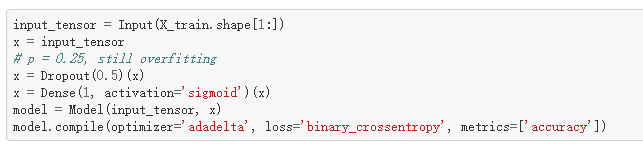


图10 增加Dropout层防止过拟合

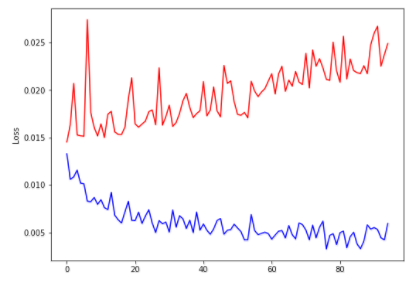


图11 呈锯齿状震荡的损失函数曲线

图11是修改后的模型得到的新的损失函数曲线，可以明显看到验证集损失曲线出现明显的震荡，还需要进一步调参：减少学习率。尝试过学习率的各种值，从1e-3开始，发现训练100代后验证集损失仍在降低，适当增大学习率到5e-3和1e-2，最后定在3e-2。相关代码如图12所示。

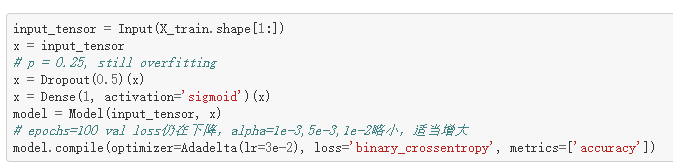


图12 调整学习率之后的模型

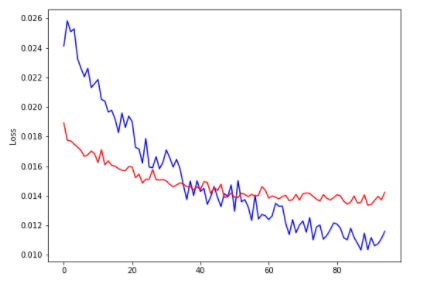


图13 最终优化模型的损失函数曲线

训练100代之后，新模型的验证集损失稳定在0.0140左右，从图13可以看到训练集和验证集损失曲线都在下降，最后收敛到0.01附近（loss=0.010，val\_loss=0.014）。

# 4. 结果

## 4.1 模型的评价与验证

我是通过观察模型训练时的损失函数曲线来得到最终模型的，因为对模型进行调优的过程不能靠凭空猜测，一定要有个感兴趣的指标，围绕这个指标来进行调参优化。最初的模型对应的验证集损失函数曲线（图9）呈上升趋势，且比训练集损失曲线高很多，这是过拟合的表现，因此为了防止过拟合，我使用Dropout。对应得到的图11则显示验证集损失函数曲线呈明显的锯齿形震荡，且模型仍然过拟合，这说明学习率太高，一开始把学习率调低到了1e-3，发现训练了400-500代val\_loss仍然在持续降低，这样训练有点慢，就适当调高学习率，在保证优化的前提下减少训练代数，最后在学习率为3e-2的基础上，训练100代，得到图13的损失函数曲线，训练集和验证集的损失函数曲线都收敛到了一个比较理想的值，因此我选择它为最优模型，是具备一定合理性的。

训练数据或输入的微小改变并不会极大地影响结果，因为在上述数据预处理的过程中，被删除的只是一些非猫非狗的异常值图片，其他容易误判的图片，比如小尺寸模糊不清的图片和内容很多容易造成干扰的图片都得到了保留，这些训练集图片的存在就是为了提高模型的鲁棒性，因为我们不能保证模型以后不会遇到质量不高的图片。综上所述，训练得到的模型是可信的。

## 4.2 合理性分析



图14对测试集进行预测的代码

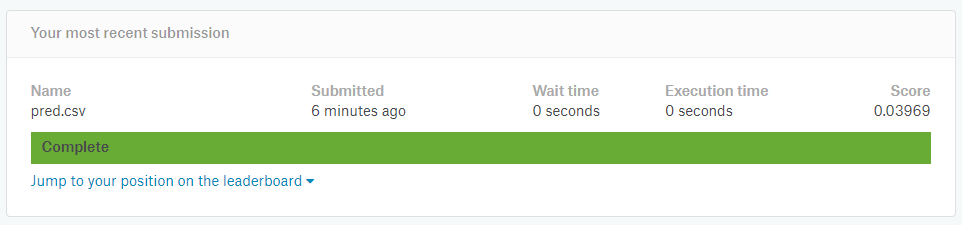


图15 预测结果在Kaggle上的评分

如图14所示的代码展示了将训练好的模型应用于测试集图片，进行预测并得到最终结果导出成csv文件的关键代码。将得到的csv文件上传到Kaggle查看分数，得到如图15所示的结果。可以看到在Public Leaderboard上得分为0.03969，低于基准测试模型要求的0.06127，最终结果比基准模型表现的更好。最终结果确实解决了问题。

# 5. 结论

## 5.1 结果可视化

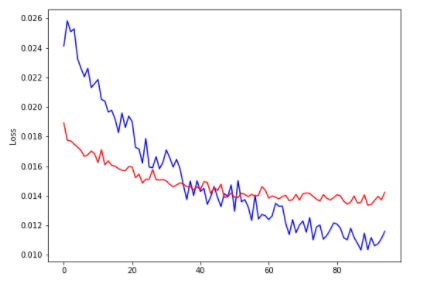


图16最终模型的损失函数曲线

图16再次展示了最终模型的损失函数曲线。要做好这个项目，需要强调的是模型的调优。然而调优并不是漫无目的的，需要有一个指标来指导调优的过程。我们最关心的是模型的泛化性能，即离开了训练集和验证集，面对从未见过的图片，模型是不是能保持较高的预测准确率。因此要避免欠拟合和过拟合问题。损失函数曲线正是观察模型表现的重要工具。如果在训练过程中，验证集损失还在下降，那就说明还有优化的空间，那么需要增加模型复杂度或多训练几代；如果验证集损失函数还在上升，说明出现过拟合，需要通过正则化或者Dropout等策略防止过拟合；如果验证集的损失函数呈锯齿形震荡，说明学习率过大，收敛不到最优参数，需要适当减小学习率；如果训练集和验证集损失基本稳定在一个较低值位置，说明已经得到了一个较好的模型，可以酌情减少训练代数。我就是通过这样的方法来制定调参策略和选定最终模型的。

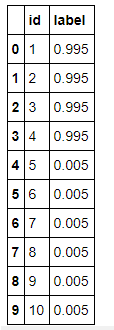


图17 限制置信度后的预测概率值

另外，这个项目还有个特点，训练好的模型在输出测试集的预测结果时，会尽可能输出非常接近0或者非常接近1的极端的置信度。Kaggle官方的评估标准是LogLoss，对于预测正确的样本，0.995和1相差无几，但对于预测错误的样本，0和0.005的差距很大。如果不做处理，Kaggle评分系统的惩罚很大导致分数不高，所以图13中的代码对结果进行了clip处理，限制模型置信度在[0.005，0.995]之间，如图17所示。这样在Kaggle上就能得到一个比较好的分数。

## 5.2 思考

总的来说，整个项目的流程分为4个部分：数据探索，数据预处理，模型训练和模型优化。

作为机器学习工程师，首先要熟悉项目提供的数据集，对它进行基本的探索，就本项目而言，应该大致浏览一遍提供的图片，统计一下猫和狗的图片数量分别有多少，进而发现一些可能影响模型性能的异常值图片。

其次，对数据进行预处理，选取合适的Top值，采用ImageNet预训练过的4种模型对训练集图片进行检测，从检测出的结果中找出非猫非狗的图片删除掉。

然后就是模型的训练，还是采用ImageNet预训练模型，去掉它们的全连接层，把它们变为特征选择器，对训练集进行预测，导出特征向量，利用4种模型得到的特征向量进行融合，然后根据问题构建自己的全连接层，直接使用特征向量作为输入进行训练，绘制模型的损失函数曲线。

最后就是根据损失函数曲线进行调参和优化，用优化的模型预测测试集得到结果，上传Kaggle查看最终得分，评估是否超过基准模型。

项目中一个比较困难的地方是数据预处理，一开始我发现训练集的有些图片是人类，想使用OpenCV中训练好的人脸检测模型来识别异常值，后来发现有些图片是人类抱着猫和狗的图片，因此这种方案不会起到很好效果，后来通过查资料发现更好的方法就是通过预训练模型来检测异常值，因为ImageNet的1000种分类中本来就有猫和狗，只要选好合适的Top值就行。最后采用这种方法完成了数据预处理的任务。项目中另一个比较困难的地方就是导出特征向量的环节，一开始我写错了steps参数，导致训练非常慢，要好几个小时，后来修改了这个bug后训练就非常快，4个模型加起来不到30分钟。我还专门去查了相关资料，看深度学习是怎么做到模型融合的，结果发现只要将导出的特征向量结合（concatenate）在一起即可 ，没有想的那么复杂。

项目中比较有意思的地方是模型的调参和优化，这完全是以损失函数图像为导向的，通过观察函数曲线，制定优化策略各个击破，首先解决过拟合问题，然后解决震荡问题，最后看着不断变化的曲线表明模型正朝着更优化的方向发展，是一件很有成就感的事情。最终得到的模型符合我当初的期望，从训练结果来看，它在验证集上准确率最高达到了99.56%，且对于测试集中的12500张图片而言，它得到的LogLoss为0.03969，远低于基准测试模型要求的0.06127。它可以在通用场景中解决问题。

## 5.3 后续改进

可以从“数据增强”的角度考虑对模型进行改进，即在ImageDataGenerator上下功夫，对图片进行各种翻转，裁剪，缩放和调节，增大训练集的样本量，使模型得到更充分的训练；也可以进一步加入更多的预训练模型，导出更多的特征向量，这就相当于从不同的角度做特征工程，然后进行融合，博采众长。这两种方法都能从某种程度上进一步提高预测准确度和模型的鲁棒性。因此以我得到的最终模型为新基准，我认为还有更多更好的解决方案。

# 6. 参考文献

[1] Karen Simonyan & Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. arXiv:1409.1556v6 [cs.CV] 10 Apr 2015

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385v1 [cs.CV] 10 Dec 2015

[3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567v3 [cs.CV] 11 Dec 2015

[4] Francois Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357v3 [cs.CV] 4 Apr 2017

[5] Keras. Documentation for individual models. https://keras.io/applications/