

机器学习纳米学位

猫狗大战 | 陶子良 | Feb 20th, 2018

I. 问题的定义

项目概述

毕业项目选择Kaggle 竞赛中比较有趣的猫狗大战。项目目标在于从Kaggle提供的数据中训练人工智能模型，来识别图片中的主体是猫还是狗。在人工智能发展初期，让一个代码程序来区分猫狗是一件比较困难的事情。但随着近些年深度学习算法的普及，用程序来识别图片内容逐渐成为可能。

2017年4月Kaggle Team对猫狗大战获胜者Bojan Tunguz采访中得知，Bojan借助Keras构建了VGG16,VGG19,Inception V3,Xception和ResNets 50 卷积神经网络。Bojan优化后的DCNN使他在Kaggle Leaderboard上获得0.35分。



问题陈述

Kaggle的猫狗大战项目是通过监督学习来训练模型，使得模型可以对一个包含猫或者狗的图像做到二分类识别。即对于给定的一张含有猫或者狗的图片，能够正确识别该图片的主体是其中的哪一种。

我们将使用Kaggle官方提供的猫狗图片数据进行学习。训练数据总共包括25,000张jpg图片，其中一半被标记为猫，另一半被标记为狗。一旦我们通过该25,000张图片完成模型训练，就可以使用12,500张未被标记的图片用来测试。我们需要对该12,500张测试数据逐一预测并把结果记录到.csv文件中，通过上传csv文件至Kaggle服务器，来获得Kaggle评分。Kaggle数据可以从 <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data> 下载。

该项目是一个典型的计算机视觉识别问题，因此我们将使用到卷积神经网络作为深度学习模型。时至今日，已有众多成熟的卷积神经网络模型被广泛使用。包括VGG16, ResNet 50, Inception V3, Xception等。上述众多模型更是经过imageNet数据集训练。在此次项目中，我们将不再重复造轮子，而是使用迁移学习的方法，尝试使用各个模型在imageNet中预训练的权重，来作为图像特征抓取的工具。最后将抓取的特征通过自建的输出层进行二分类。除此之外我们将进行更多的探索：包括合并多个预训练模型所抓取的特征来做输出；开放预训练模型最后一个block来微调(Finetuning)整个模型。

评价指标

使用Kaggle的test数据集，设0为猫，1为狗。计算每个样本的二分类预测结果并提交Kaggle验证。

Kaggle将对提交的数据计算log loss, 具体公式为：

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

其中

- n 是测试集图片样本数量
- \hat{y}_i 是模型预测图片为狗的可能性
- y_i 代表正确的结果： 1=狗， 0=猫（非狗）

Logloss得分越低，代表模型预测能力越好。目前Kaggle第10名分数为0.03807。

II. 分析

数据的探索

在「问题陈述」中已经提到，我们使用的是官方猫狗大战的数据集。训练数据总共包括25,000张jpg图片，其中被标记为猫、狗的图片各12,500张。训练集中标记为猫的图片以train/cat.XX.jpg命名，标记为狗的以train/dog.XX.jpg命名，猫狗各1,2500张图片(index从0到12499)。测试集test/X.jpg，共12500张(index从1到12500)，未做标记。猫狗图片数量1:1，单张图片尺寸宽高不超过500px（除了cat.835和dog.2317之外）

以下是数据集的一些样例，第1行为随机抽取的样例，第2~3行为基准模型判断错误的样例。



从样例中我们可以发现Kaggle数据图片有以下这些特点：

1. 照片大多拍摄于生活场景。图片内容以单只主体的猫/狗为主，主要拍摄主体的全身或头部特写。
2. 图片偶有多只主体出现。甚至有与人物的合影，使得主体占据图片篇幅有限。
3. 部分被拍摄主体被牢笼等物体遮挡。
4. 个别图片出现被旋转的情况。
5. 个别图片非真实猫狗主体，而是卡通图像。

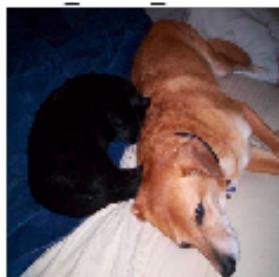
错误与无效训练数据

标签错误

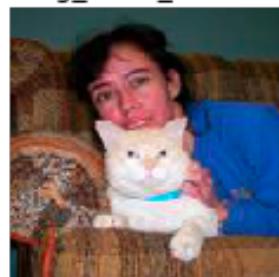
cat_4085_0.9998



cat_7920_0.9998



dog_4334_0.0006



cat_5355_0.9993



猫狗混合

cat_9444_0.9944



cat_7194_0.9943



cat_9882_0.9936



cat_724_0.9931

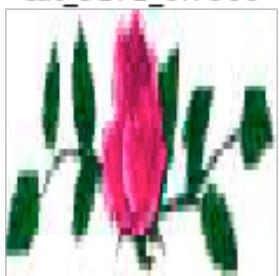


非猫狗图片

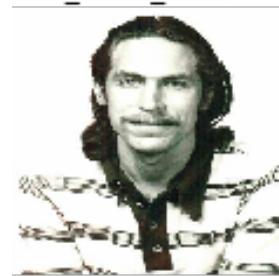
cat_10712_0.8988



cat_9171_0.7906



cat_7377_0.9225



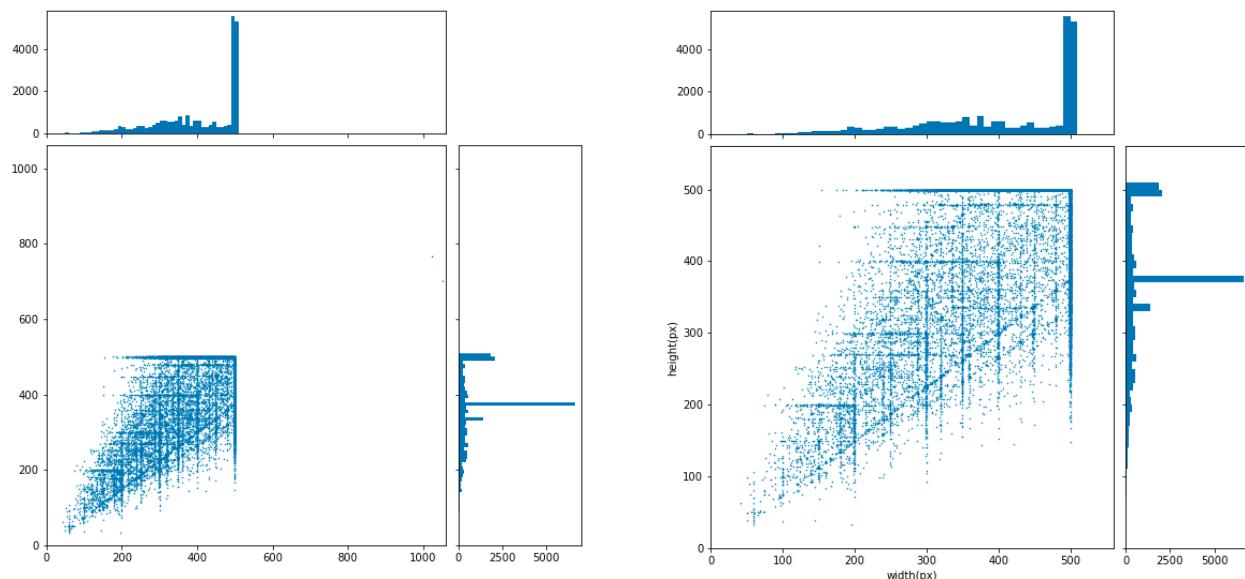
以上错误和无效数据中，图片标题的 cat/dog 代表该图片在训练集中属于哪个分类。中间的数字代表该图片的编号。末尾的数字代表基准模型预测结果。比如 cat_4085_0.9998 代表该图位于训练集 cat.4085.jpg 被基准模型预测为 0.9998（狗）。

我们发现问题数据主要分为三类：

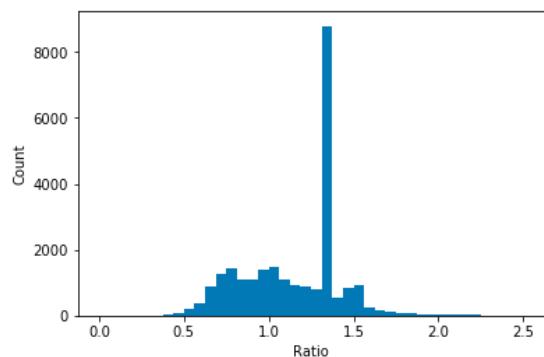
- 标签错误：把狗的图片归为猫，反之亦然
- 猫狗混合：图片中同时包含猫狗，但标签只有一个。这类数据占三类问题中的多数。根据已发现的问题数据，猫狗混合图片多数都被归为「猫」类。
- 非猫狗图片：图片内容与标签无关

探索性可视化

在训练模型之前，我们需要先统计 Kaggle 训练数据集尺寸和长宽比例。下图是数据集的图像尺寸：x 轴为宽度、y 轴为高度，单位都是像素。我们可以发现几乎所有的图片高宽都被限制在了 500px 以下，但有两张图片的宽度超过了 1000px。下右图去除了 2 个异常值后的图像尺寸分布。



右图展示了图片的宽高比分布。x轴代表宽高比，y轴是对应的图片统计数量。我们可以发现绝大多数图片的宽高都在1:2 ~ 2:1之间。其中1.33即4:3宽高比的图片分布最多。根据以上图片尺寸统计，我们可以发现图片高宽比相对还比较集中，在训练模型时，预处理图象时直接把图象拉伸至1:1作为输入是可行的。



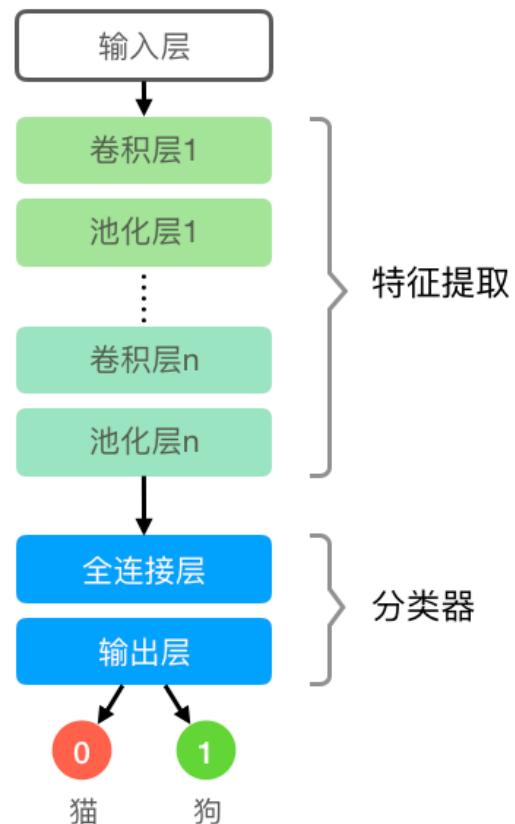
算法和技术

卷积神经网络与图象识别

猫狗大战是一个典型的图象识别二分类项目。在图象识别领域，卷积神经网络技术被广泛使用。我们也将在此项目中使用该技术。总体而言，卷积神经网络可分为两个部分：「特征提取器」与「分类器」。

「特征提取器」的作用是抓取图片数据中的各种图形特征。不同模型的「特征提取器」结构会有较大差异，但最基本的「特征提取器」通常由若干个卷积层、池化层、激活层组成。

「分类器」的作用是将「特征提取器」提取出来的特征数据预测输出，把图片归为项目需要的某一个类别。「分类器」通常由多个全连接层组成，将具有较深维度的特征数据通过Flatten或GlobalAverage的方法降成一维，并连接至深度为n的输出层（n为项目分类数量）。由于该项目为二分类，因此最后输出的深度可以只是1。即把“0”代表猫，“1”代表狗。



迁移学习与预训练模型

为了实现项目目标，我们当然可以自己搭建卷积网络，使用随机正态分布的初始化权重值来训练。然而为了使模型表现优秀，我们会需要有较深的神经网络，使用许多隐藏层来学习捕获图片中不同图层中的不同特征。这会使深度学习网络有大量的权重参数需要学习。然而一旦当需要学习的参数较多时，我们就不得不面对两个问题：

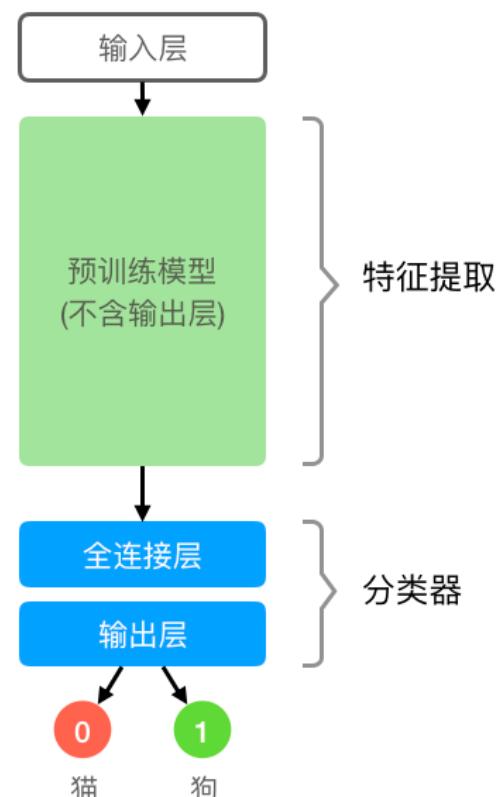
1. 训练卷积网络需要花费大量的时间（数百小时）和算力来优化各个卷积层中的权重参数
2. 需要大量的训练数据（百万级别）来使深度网络中的权重参数收敛到更准确的位置

此次项目中Kaggle给到我们训练用的数据量是2.5万张，对于训练一个深度学习网络来说，即使我们通过Image Augmented虚拟出来更多数据，这个数据量也是很有限的。为了使模型获得更好的预测效果，我们此次将采用迁移学习。

迁移学习是将一个已经被大量数据训练过的模型拿来作为我们深度网络的一部分。我们将这个模型称作「预训练模型」。若预训练模型训练时使用的数据和我们这次项目的数据较类似时，我们可以直接使用预训练模型作为特征提取器。

幸运的是，在ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 中，许多在竞赛中获胜的模型被开源出来供大家使用。并且大赛中使用的训练数据集是ImageNet，一个涵盖2万多个分类的图片数据库。ImageNet中每个细小的分类中拥有平均500张被标记的图片。

下图是imageNet数据库中搜索cat与dog出现的部分结果：

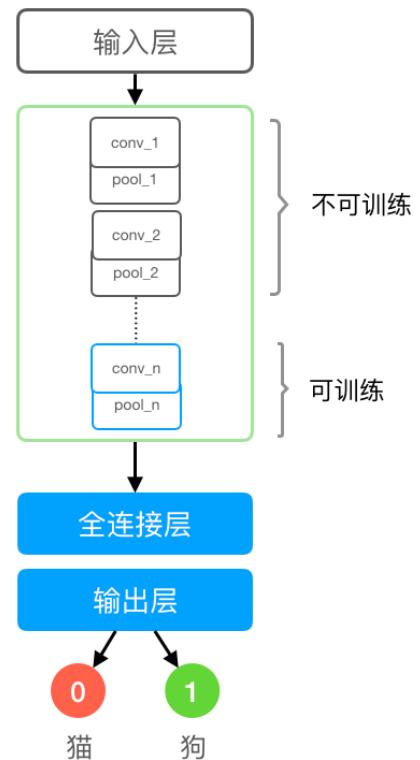




我们可以发现ImageNet中关于猫狗的图片数据和我们该项目的图片数据是比较类似的，因此我们可以直接把某个预训练模型作为特征提取器，作为项目研究的起点。

Fine-Tuning 微调

Fine-tuning 是一种通过微调预训练模型中少数层的参数来针对当前项目优化模型的技术。在卷积网络中，越靠近输入层的卷积层从图片中所提取出来的特征越是通用；相反越是靠近输出层的卷积层从图片中所提取出来特征越是针对特定项目。因此为了在预训练模型基础上针对当前项目优化，我们可以开放靠近输出层的神经网络使其可训练，同时锁定预训练模型中其他层。



合并模型作为特征提取

在ILSVRC中优秀的模型很多，比如有 ResNet, Inception, Xception, VGG等等。我们可以使用任一个模型配合我们自己的分类器来训练我们的模型。然而如果我们同时使用多个预训练模型来为我们提取图片特征，并将每个模型获取的特征向量合并，这将帮助我们的模型获得更好的分类效果。好比原本我们只有一个侦察兵来观察目标，现在我们可以

以有若干个侦察兵相互独立得进行观察，并将观察的结果汇总到司令部（分类器）来做出最后的判断。这将降低单个模型的限制导致错判发生的概率。

图片增强 (Image Augmented)

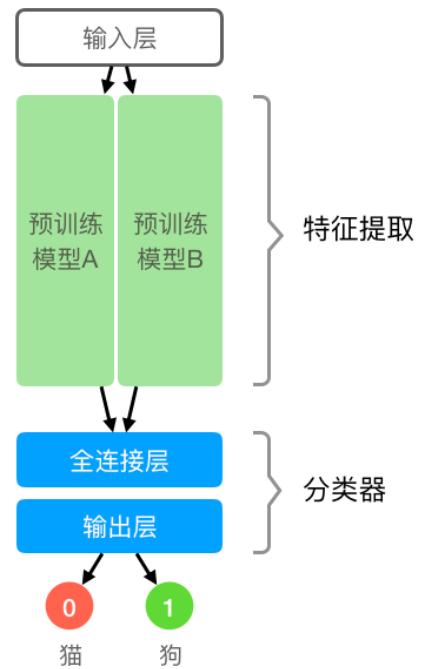
图片增强是一种通过镜像、随机裁剪、放大、旋转等方式来扩充训练数据的方法。通常我们训练模型时，每一代 (epochs)都会把训练集的每张图片作为输入源，也就是说如果我们训练模型时使用了n代，则相同的图片在训练时出现了n次。而图片增强技术就是对同一张图片进行变形调整，使得每代训练输入的图片不会完全相同。这会使模型的鲁棒性得到增强，有助于模型泛化。

我们可以使用多种方法来增强图片，以下是常用的几种方法：

- 镜像翻转
- 随机裁剪
- 缩放
- 平移
- 旋转
- 错切变形(shear mapping)

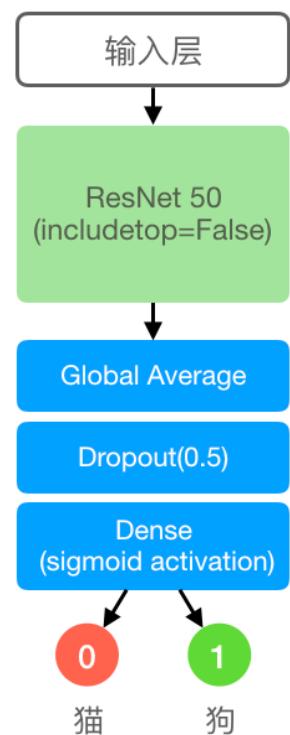


上图是将同一张图片进行图片增强后随机产生的数据。（图片引自<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>）



基准模型

本次项目使用ResNet50作为基准迁移学习的预训练模型，配合Global Average + Dropout (drop rate=0.5) + 分类层 作为输出层。模型使用adamdelta优化器经过10代训练后，validation loss = 0.0323 ; validation accuracy = 0.989 ; 模型预测测试集提交Kaggle后的分数为0.05291。



III. 方法

数据预处理

数据文件结构

由于我们将使用Keras来训练模型，当我们使用imageDataGenerator的flow_from_directory方法来指定包含图片的文件夹时，Keras以该目录下的子目录名作为图片的标签。为了使其正常工作，需要对图片的文件结构做调整。

	原结构	调整后结构 不区分Validation	调整后结构 区分Validation
训练数据	data/ train/ cat.1.jpg ... dog.1.jpg ...	data/ train2/ dog/ dog.1.jpg ... cat/ cat.1.jpg ...	data/ splitted/ train/ dog/ dog.1.jpg ... cat/ cat.1.jpg ...
评估数据		在程序中从训练数据中分割	data/ splitted/ valid/ dog/ dog.21.jpg ... cat/ cat.13.jpg ...
测试数据(不带标签)	data/ test/ 1.jpg ...	data/ test2/ test/ 1.jpg ...	

我们把原数据结构调整成了两种：

	描述	作用
不区分Validation	把所有带标签的数据放在train2文件夹下	使用该结构可以一次性把所有带标签的数据转换成特征向量。在之后的训练中直接读取特征向量，训练时再切分训练和验证数据。
区分Validation	预先切分训练与验证数据集，并分别放在train和valid文件夹下	当使用数据增强技术时，无法预先使用模型抓取特征向量，并且需要对训练图片做变形处理，而对验证图片不需要做变形处理。因此需要区分两组图片，分别使用不同的ImageDataGenerator。

标准化数据

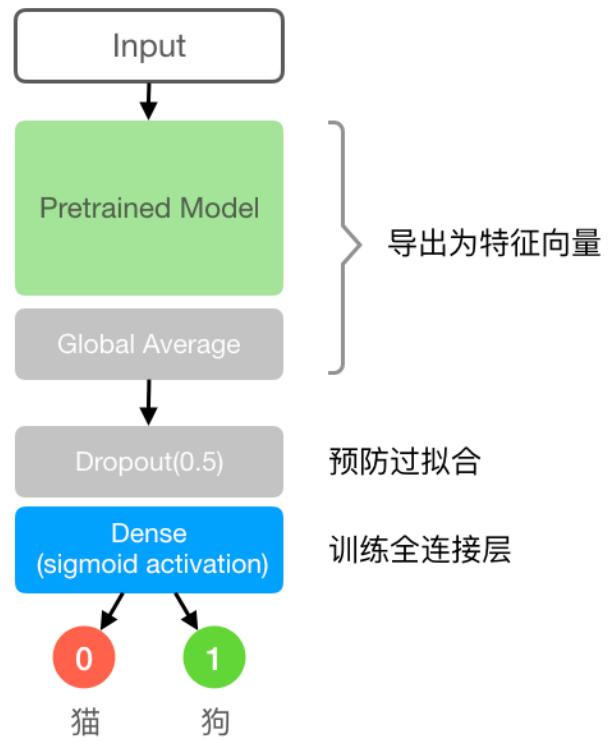
由于我们将使用基于ImageNet图像数据而预训练过的模型来完成迁移学习，因此数据预处理的方法必须与预训练模型在训练ImageNet时使用的预处理方法一致。而该预处理方法已经被定义在了keras.applications.imagenet_utils.py的preprocess_input中。在该函数中我们可以知道图片预处理的方法主要有两种：

1. 将图片像素信息归一化到[-1,1]之间。使用的模型有：InceptionV3, Xception以及 InceptionResNetV2
2. 将RGB的图片转成BGR然后减 ImageNet 均值。使用的模型有：VGG16、ResNet50

执行过程

使用预训练模型提取并保存特征向量

首先我们使用预训练模型来提取所有训练数据集和测试数据集的图像特征，以避免训练时每次epoch都要重复计算特征向量，这样可以大大加快模型训练和测试速度。我们用来提取特征向量的模型有：ResNet50、VGG16、InceptionV3、Xception 以及 InceptionResNetV2。在实际操作中我们把所有预训练模型（不包含顶层）都添加了一层GlobalAveragePool来输出特征向量，这么做好处在于可极大降低特征向量的数据存储量。而且GlobalAveragePool相对于Flatten可以极大减少全连接层需要训练的参数数量。拿ResNet50来说，不包含顶层的ResNet50得到的特征向量shape是(7,7,2048)，经过GlobalAveragePool处理后是(2048)，数据量减小49倍。

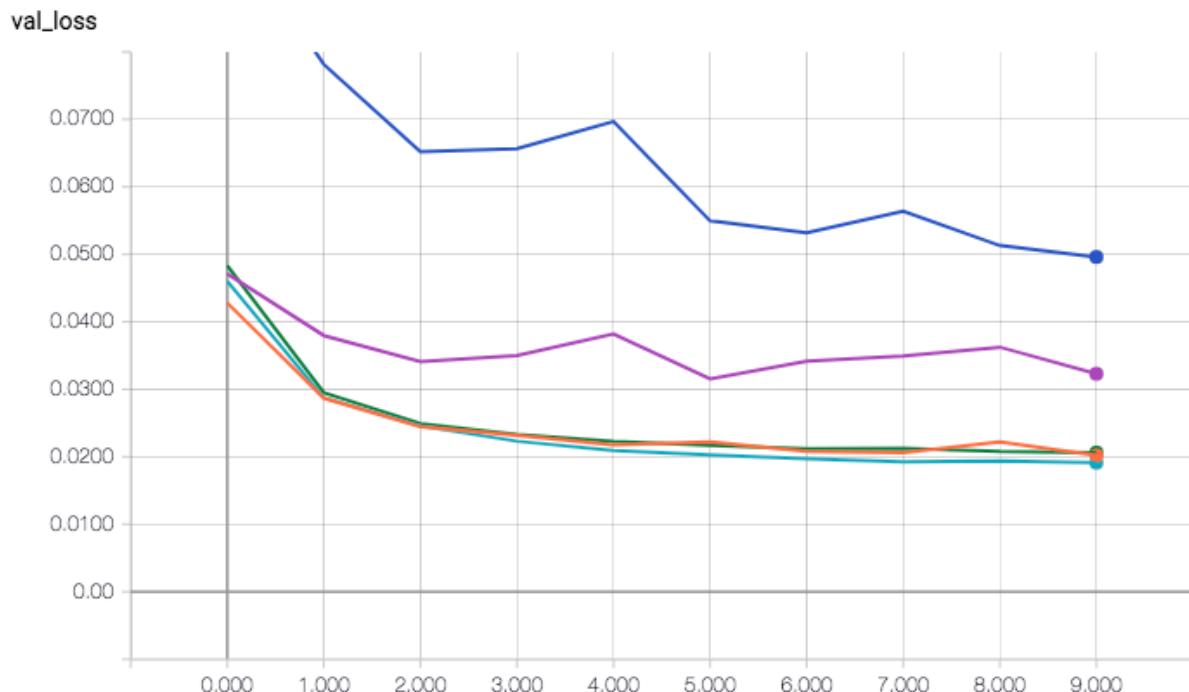


创建输出模型，训练全连接层

由于在提取特征向量时我们已经使用了GlobalAveragePool，因此在输出模型中我们直接使用Dropout(droprate=0.5) 和 全连接层 Dense(1,activation="sigmoid")。所有需要训练

的权重都在全连接层上。模型使用adadelta作为优化器，binary_crossentropy作为损失函数。全局Batchsize设为128。

以下是针对各种预训练模型所提取的特征向量，使输出模型经过10代训练得出的val_loss数据：



Name	Value	Step	Time	Relative
InceptionResNetV2	0.02019	9.000	Tue Mar 20, 15:21:32	6s
InceptionV3	0.01913	9.000	Tue Mar 20, 15:20:54	6s
ResNet50	0.03230	9.000	Fri Mar 23, 12:26:18	11s
VGG16	0.04961	9.000	Tue Mar 20, 15:20:16	6s
Xception	0.02063	9.000	Tue Mar 20, 15:21:13	6s

我们发现InceptionResNetV2、InceptionV3、Xception 表现较好，三个模型之间的loss也比较接近。而ResNet50和VGG16损失值偏大。

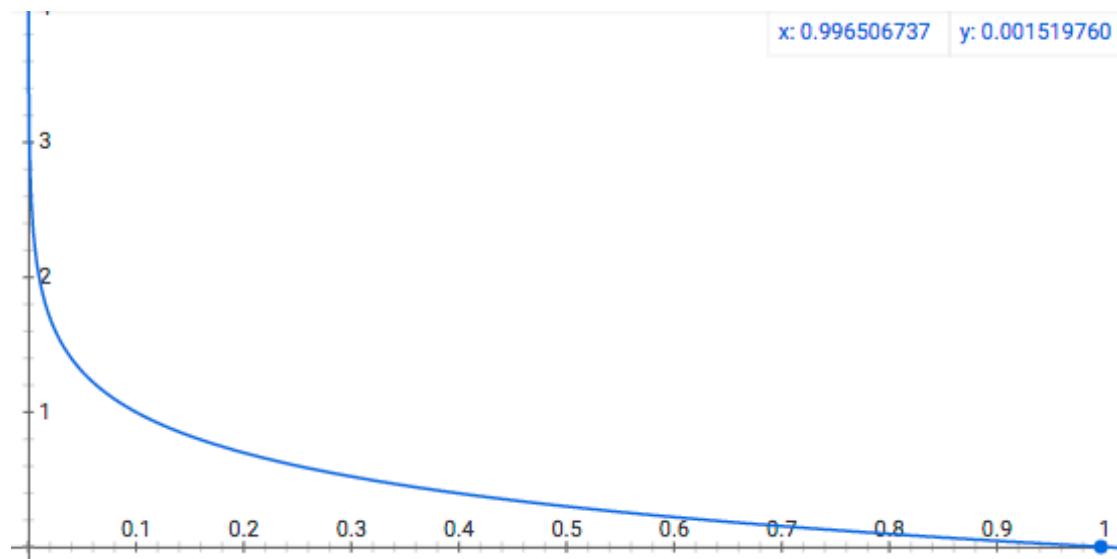
接下来我们通过Kaggle测试数据，把所有训练完的模型对测试数据进行预测。不过在导出测试结果时，我们可以对测试结果进行优化：

由于Kaggle的损失函数计算方法为logloss，公式为：

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

Logloss公式的特点在于：当我们预测的结果正确时，我们对结果越自信（数值越接近0或1），得到的loss值下降越不明显。相反当我们预测的结果错误时，我们对结果越自信，得到的loss值上升越显著。

举例来说如果某图片为狗(即结果为1)，我们的预测值为x，Kaggle针对该图片给我们的评分为 $y = -\log(x)$



由图可知，当我们预测错误且预测值从0.1靠近0时，该图片的loss从1增大到无穷。而当我们预测正确且预测值从0.9接近1时，该图片的loss只是从0.045降至了0。由于logloss这种不对称性，为了优化我们的Kaggle评分，我们统一把所有预测值收缩到[0.005,0.995]之间，来避免由于过分自信错误结果而导致评分波动。

基于以上优化我们把所有模型测试结果上传Kaggle，最终模型关键指标如下：

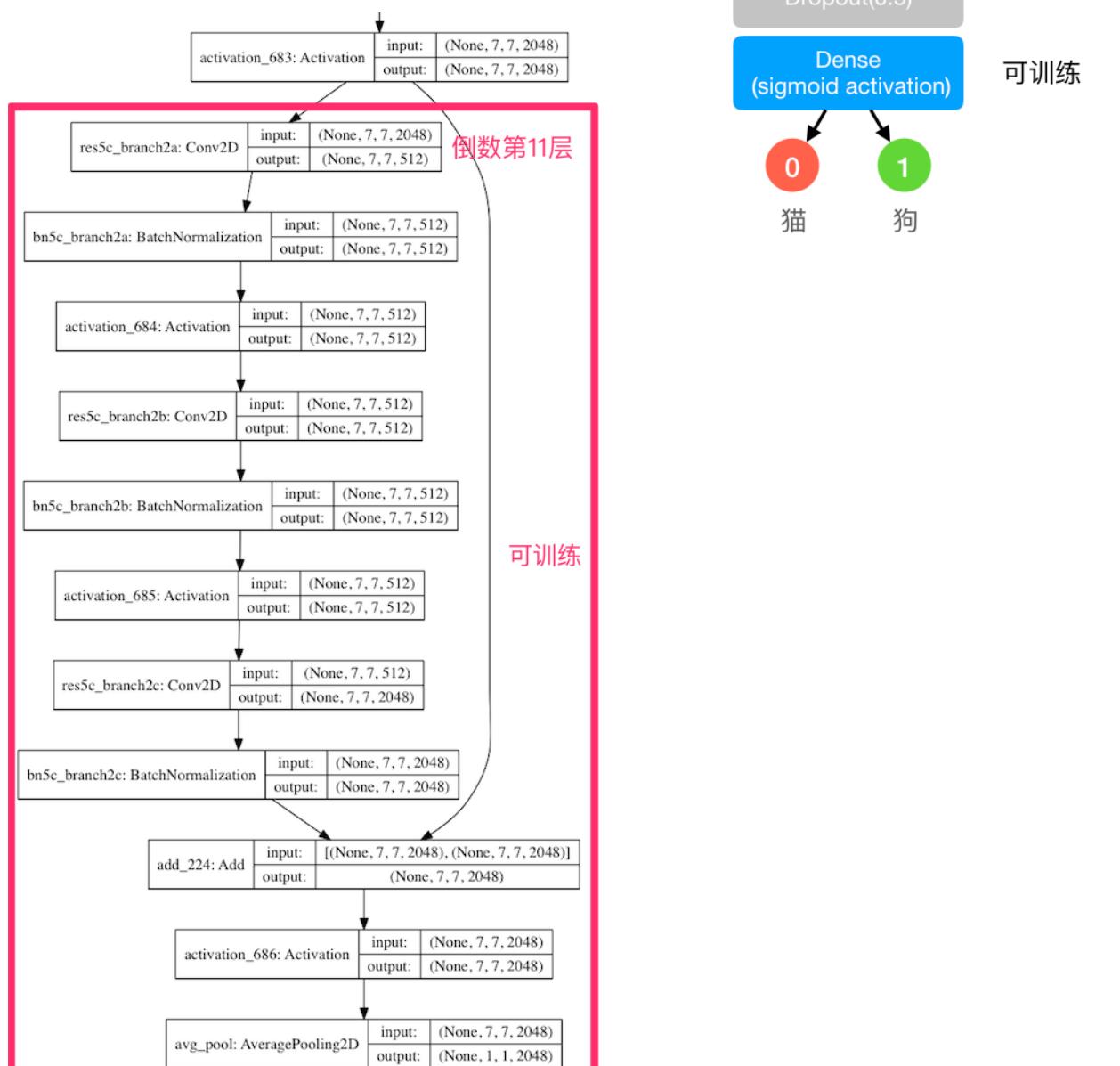
从预训练模型提取特征，训练输出层

模型	val_accuracy	val_loss	Kaggle score
VGG16	0.9816	0.04961	0.06628
ResNet50	0.9892	0.03230	0.05291
Xception	0.9942	0.02063	0.0407
InceptionV3	0.9938	0.01913	0.0407
InceptionResNetV2	0.9950	0.02019	0.0381

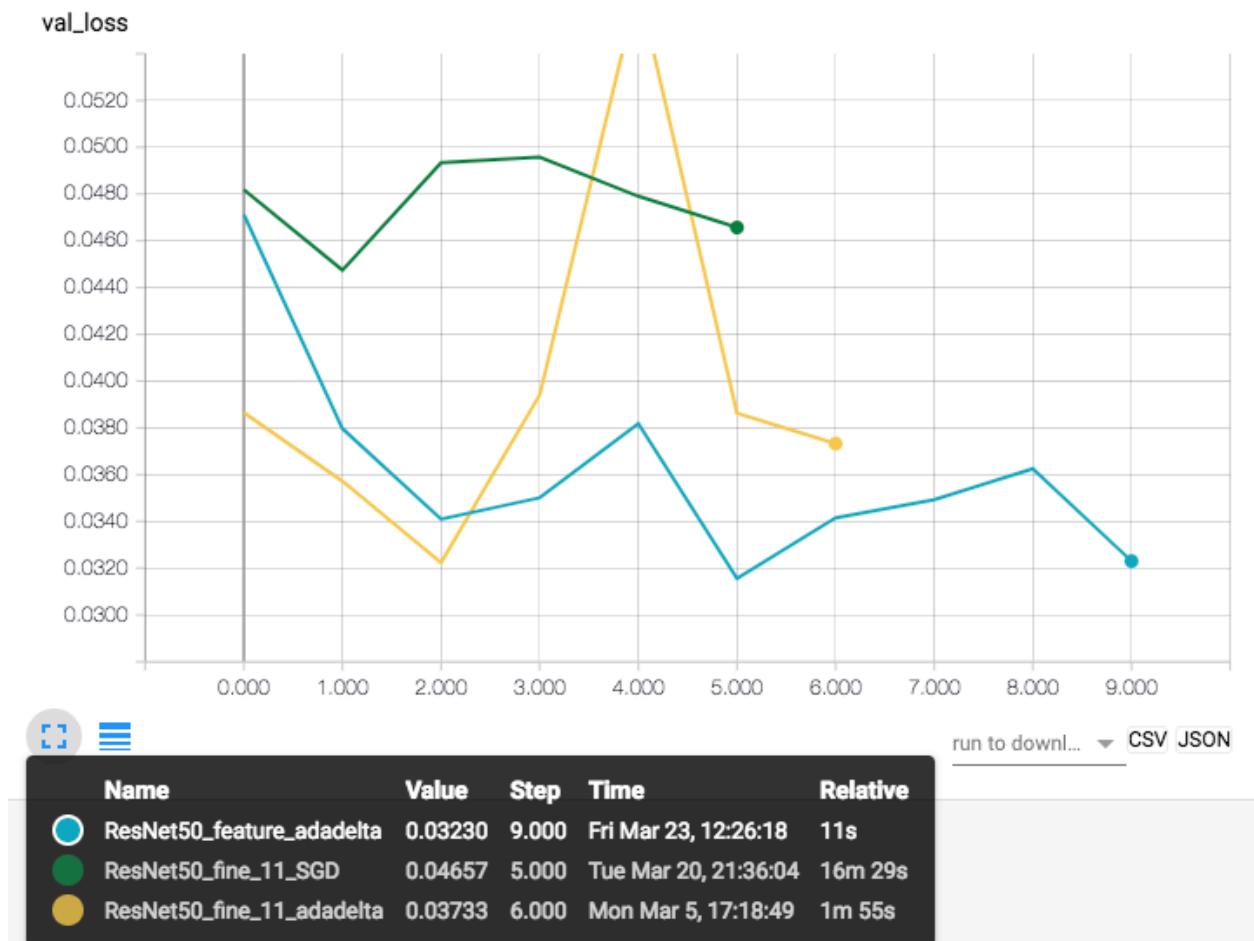
Finetuning - 开放特定层

在这一步我们将使用原图作为输入，把我们自己的输出层添加在预训练模型ResNet50之后。由于在Finetuning中，我们只需要训练ResNet50靠近输出层的若干层，因此我们将其余层锁定。这个模型构架如图。

根据ResNet50模型靠近输出层的模型结构，此次Finetuning我们解锁最后一个block，即解锁倒数第11层至输出层。该block中共有三层卷积层加入Finetuning训练。



我们分别用Adadelta和SGD($lr=0.001$, $momentum=0.9$)作为优化器来finetuning。由于该finetuning计算量较大且训练时间较长，为了防止过拟合整个训练中设置EarlyStopping(patient=4)，即如果连续4次val_loss没有改善，训练就提前终止。测试结果如下：



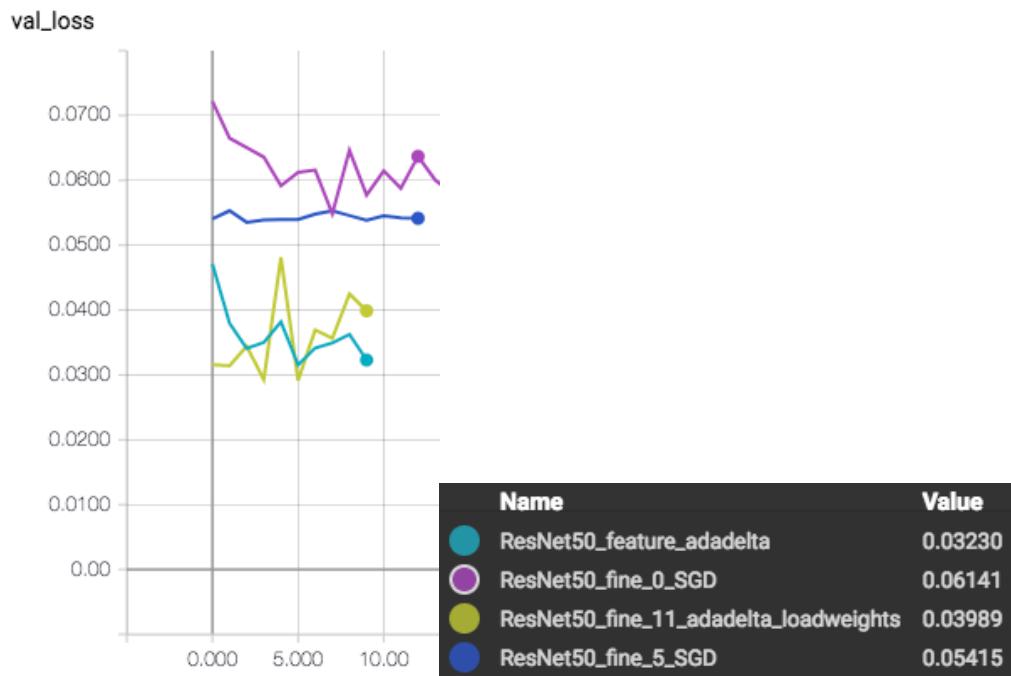
图中蓝色是基准模型。我们发现经使用Finetuning训练的模型val_loss表现并没有提升，反而要弱于基准模型。猜测第一种可能是由于我们创建的输出层是随机权重，而根据这些随机值产生的权重变动却反而较大影响了只需要微调的预训练模型中的权重；也可能是由于可训练的层太多导致finetuning极容易过拟合。基于此猜测我做了以下尝试：

模型A	将基准模型获得的全连接层权重应用在finetuning模型上
模型B	先设置可训练ResNet50层为0（即只训练全连接层），训练5代；然后开放可训练层5（即开放ResNet50最后一个卷积层），训练时设置EarlyStopping

模型C

设置可训练ResNet50层为0，即用SGD只训练全连接层

由此得到以下测试结果：



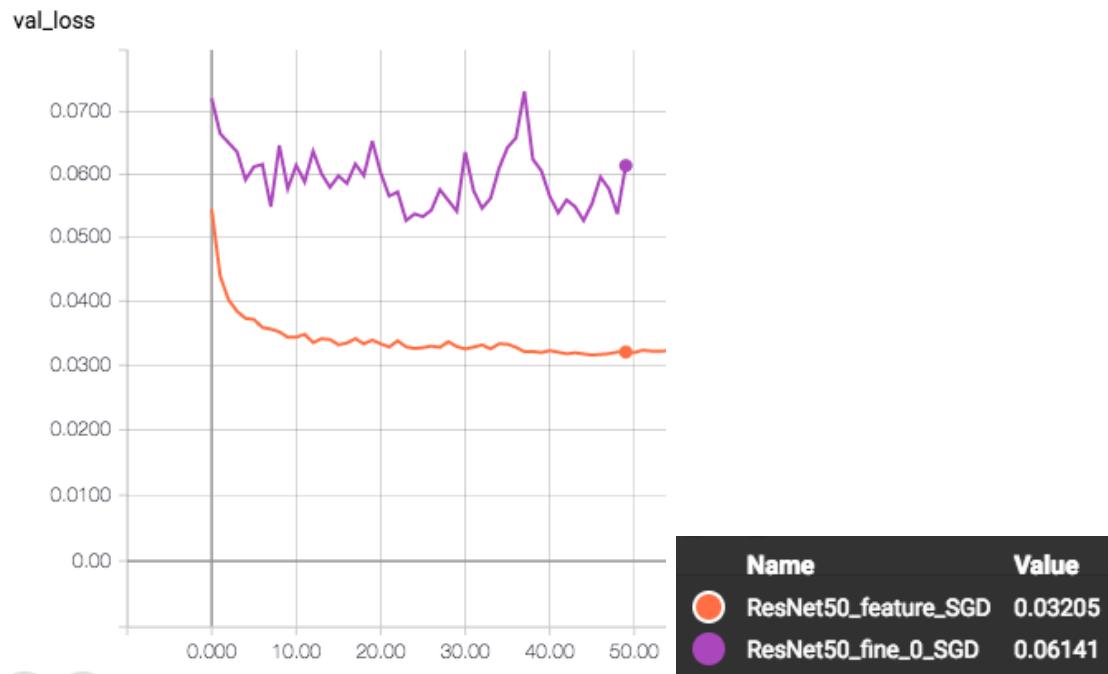
根据图表我们发现，在Finetuning中当我们使用ResNet50来训练整个模型时，不论我们放开1层卷积还是3层卷积，甚至锁定整个ResNet50模型只训练输出层使，val_loss表现都不如基准模型。为了再次确认是否是该训练方法导致的差异，我做了以下试验：

模型C

使用图片作为Input，锁定所有ResNet50层，即用SGD训练全连接层

模型D

使用之前提取的特征向量作为Input，用SGD训练全连接层

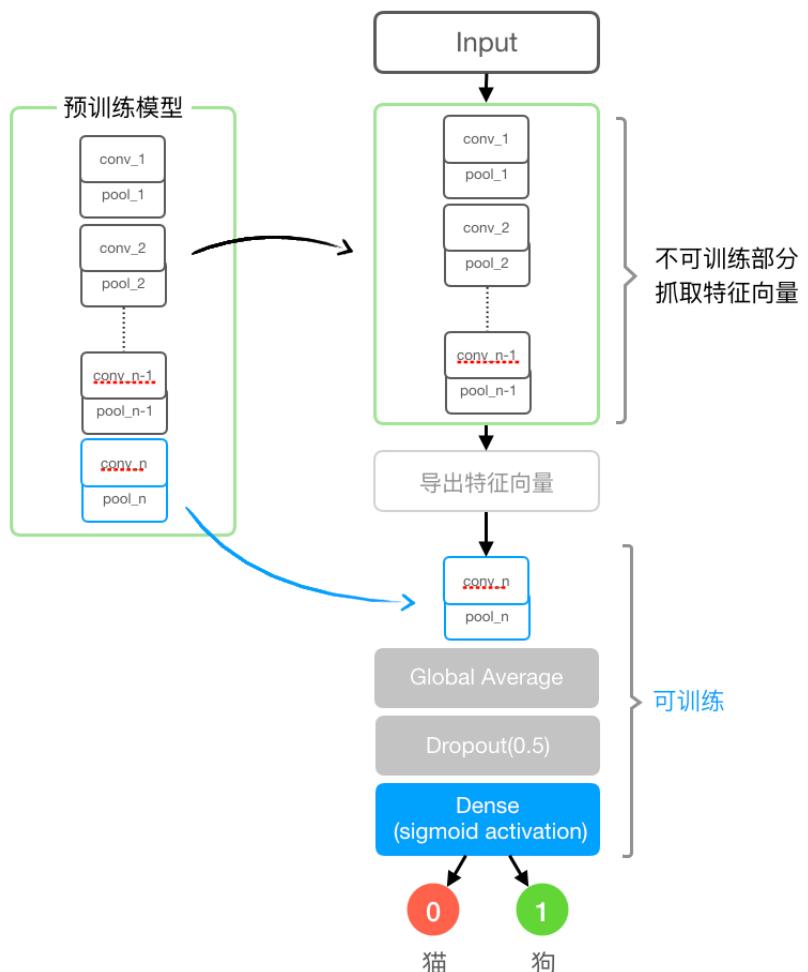


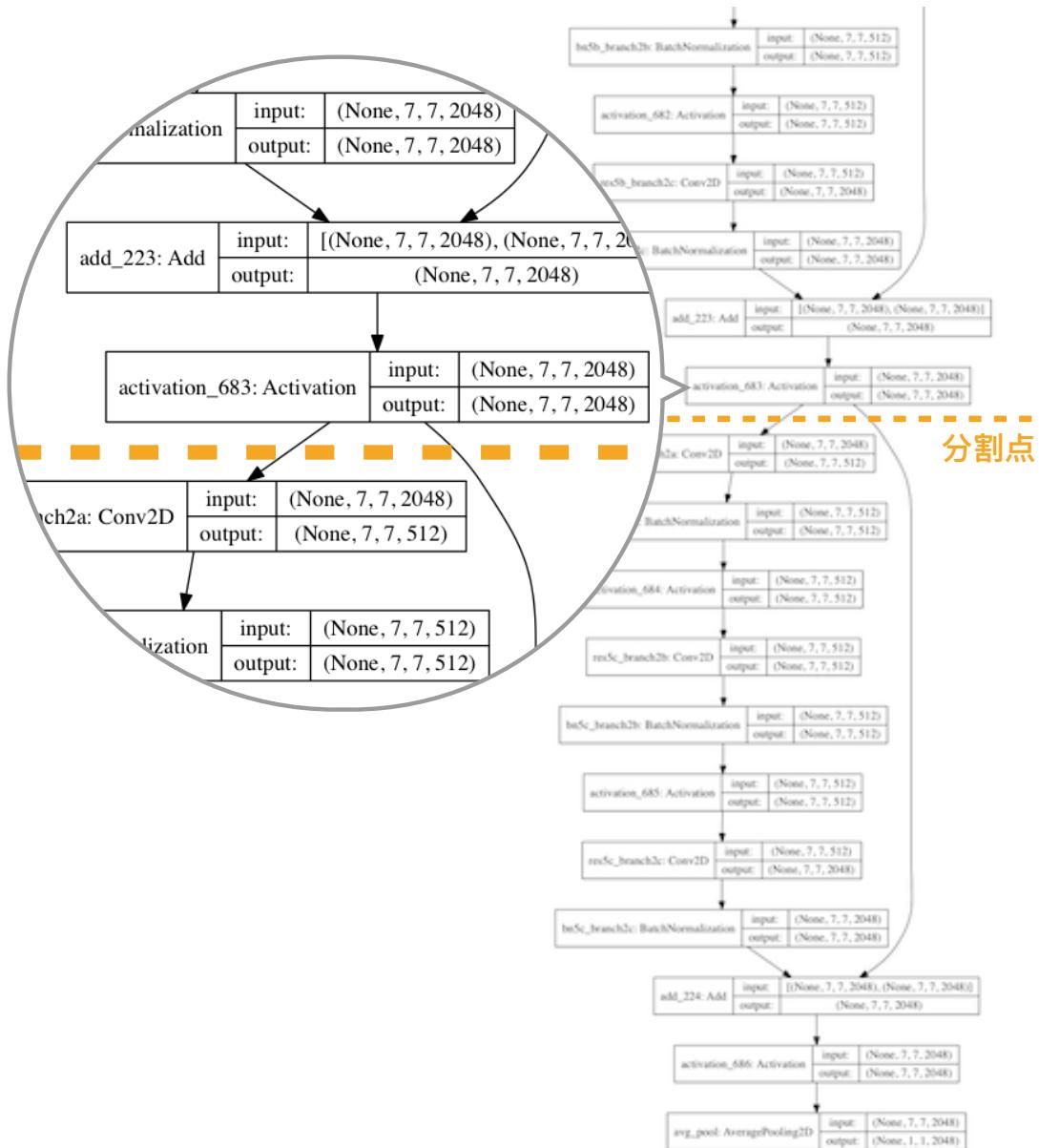
两者SGD优化器参数一致 ($lr=0.001, momentum=0.9$)。我们发现当其他条件一致时，使用提取的特征向量作为Input的训练结果比锁定模型层的训练结果有明显优势。

Finetuning - 分割模型

由于上述直接使用图片作为Input，训练ResNet50部分图层的结果并不理想。我尝试了另一种Finetuning方法：把ResNet50最后一个block分离出来，接上输出层一同训练。而ResNet50其余部分用来抓取特征向量。模型结构如图

具体到ResNet50模型上，我们选择最后一个含有卷积层的区块作为分割点，也就是最后第11层。如图：





分割点以上的模型用来提取特征向量；分割点以下部分衔接输出层用来训练。输出层暂时仍使用GlobalAveragePool+Dropout+Dense。由于预训练模型中所有层之间的关系已经被绑定，在分割点以下的模型中，第一个卷积层和最后一个Add层分别都被分割点以上的激活层绑定。而我们希望这两层应当关联到我们新创建的Input层中，因此我们这里需要一些额外的代码来改变这个绑定关系：

```

# 从切分点开始之后的所有layer中，如果已连接了原模型切分点前一个layer，则改变连接到new_input
last_untrainable_layer = untrainable_layers[-1]
for layer in trainable_layers:
    replace_connect_layer(original_from_layer=last_untrainable_layer,
                          new_from_layer=new_input,
                          to_layer=layer)

trainable_model = Model(new_input, trainable_layers[-1].output)

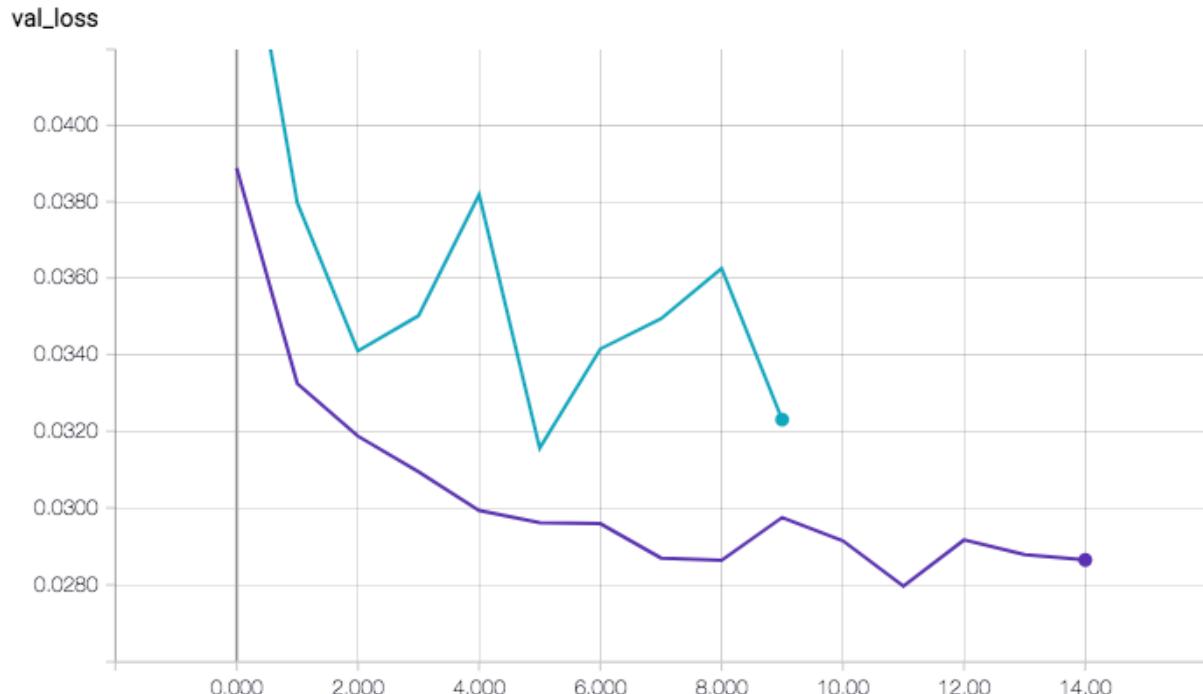
def replace_connect_layer(original_from_layer,new_from_layer,to_layer):
    """
    把original_from_layer -> to_layer的flow变更为new_from_layer -> to_layer。只改layer之间的关系
    :param original_from_layer:
    :param new_from_layer:
    :param to_layer:
    :return:
    """
    for i, each_inbound_layer in enumerate(to_layer._inbound_nodes[0].inbound_layers):
        if each_inbound_layer == original_from_layer:
            _connect_layer(from_layer=new_from_layer, to_layer=to_layer, inbound_layers_index=i)

def _connect_layer(from_layer,to_layer,inbound_layers_index):
    to_layer._inbound_nodes[0].inbound_layers[inbound_layers_index] = from_layer.keras_history[0]
    to_layer._inbound_nodes[0].input_tensors[inbound_layers_index] = from_layer

```

由于我们每张图的矢量特征shape是 7*7*2048 (最后一层激活层的输出shape)，每张图的矢量特征文件大小为 $7*7*2048*4=382k$ 。因此特征文件总尺寸约为25,000张*0.382M = 9.34G。这相比之前经过GlobalAveragePool处理过的特征向量文件来说庞大很多。

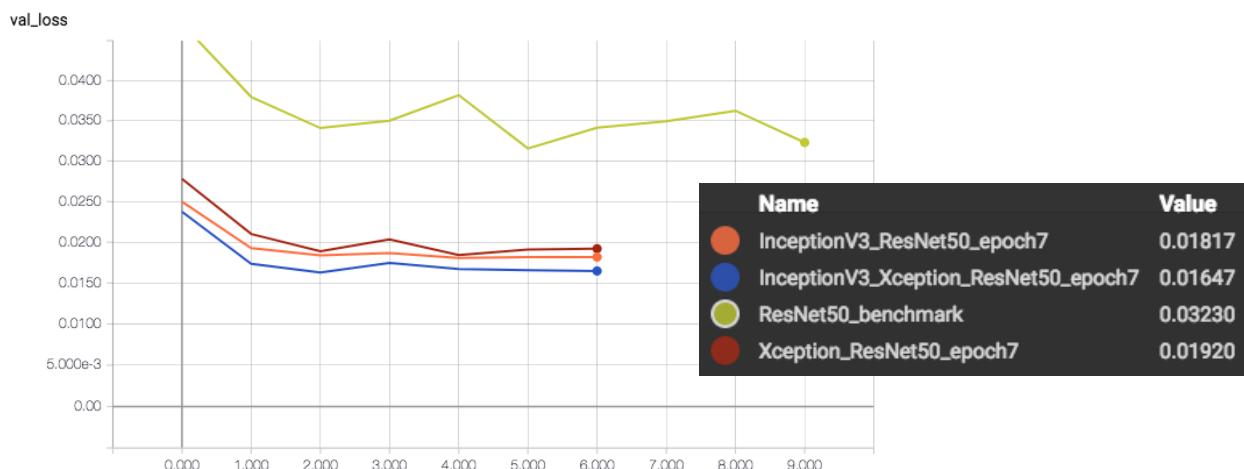
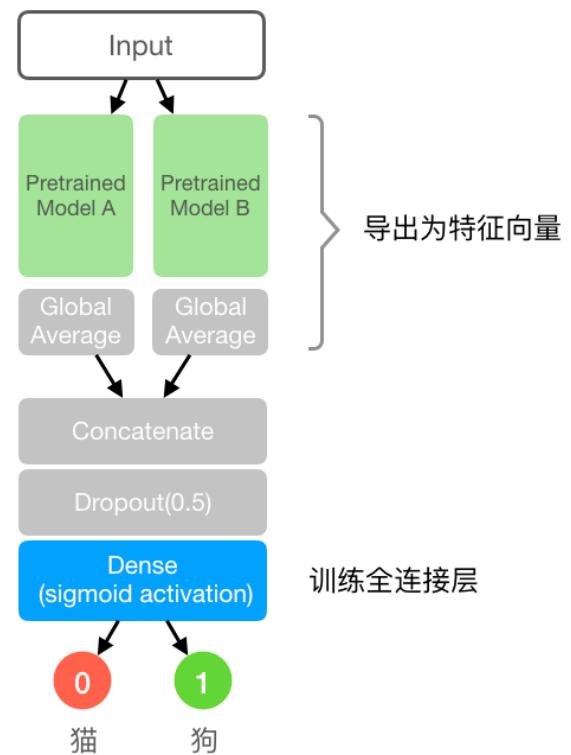
经过此次Finetuning后，模型val_loss有些许改善。蓝色为基准模型，紫色为切分后的模型。



合并模型

我们尝试合并多个预训练模型+GlobalAveragePool导出的特征向量，来训练全连接层分类。合并模型结构如图：

Concatenate层可以帮助我们把多个预训练模型输出的特征向量合并在一起，相当于让不同的预训练模型各自抓取图像特征，然后把所有特征汇聚在一起给全连接层作为分类依据。这样全连接层得到的特征线索更多。我们尝试把ResNet50分别与InceptionV3,Xception两两组合以及三个模型全合并，使用Adadelta训练7代。以下是训练时val_loss图形：



从图中我们可以发现，合并模型后模型的表现得到了明显提升。所有模型的val_loss全部低于0.02分。我们将这些模型进行测试并提交Kaggle评分，最终得到以下结果：

模型	val_loss	Kaggle评分
ResNet50+InceptionV3	0.01817	0.04100
ResNet50+Xception	0.01920	0.03980
InceptionV3+Xception+ResNet50	0.01647	0.03964

我们可以发现InceptionV3,Xception模型与ResNet50合并的三个方案中，三者全合并后的表现相对较好，之后我们就根据该合并模型来改善训练参数。

完善

最后我们对InceptionV3+Xception+ResNet50使用SGD优化器进行调参训练。在该过程中我们准备优化的超参数有：

参数	使用参数的对象	参数选择范围	选择方法
learning rate	SGD 优化器	0.001~0.01	连续随机
decay	SGD 优化器	0.0005~0.05	连续随机
momentum	SGD 优化器	[0.5,0.8,0.9]	离散随机
dropout rate	Dropout层	[0.1,0.2,0.3,0.5]	离散随机

在实现中我们使用HyperOpt来帮助我们循环寻找最优化的参数组合。在HyperOpt中我们需要先定义需要寻找的参数空间：

```
hyper_params = {
    "data": {
        "model_name" : ["InceptionV3", "Xception", "ResNet50"]
    },
    "model": {
        "lr": hp.loguniform("m_lr", np.log(0.001), np.log(0.01)),
        "decay": hp.uniform("m_decay", 0.0005, 0.05),
        "momentum": hp.choice("m_mom", [0.5,0.8,0.9]),
        "dropout": hp.choice("m_do", [ 0.1,0.2,0.3,0.5]),
    },
    "fit": {
        "batch_size": 128,
        "epochs": 200,
        "patience": 8
    }
}
```

参数中需要分别给 data(生成训练数据时相应函数需要传入的参数)、model(生成训练模型时相应函数需要传入的参数)、fit(训练时需要使用的参数)。我们主要通过HyperOpt来随机选择model部分的超参数。

接着就是构建data与model函数。其中重复利用的关键函数比如load_train_data_merge等都在helper.py中定义。HyperOpt每次评估一套超参数时都会根据我们之前定义的选参空间hyper_params来生成一套超参数。

```
def data(model_name):
    Xs, y = load_train_data_merge(model_name)
    return (Xs, y)

def model(train_data, lr=0.001, decay=0, momentum=0.9, dropout=0.5):
    # extract data dimensions
    input_shapes = []
    Xs = train_data[0]
    for feature in Xs:
        input_shapes.append(feature.shape[1:])

    sgd = SGD(lr=lr, decay=decay, momentum=momentum)
    model = create_model_merge_output(input_shapes, optimizer=sgd, drop_rate=dropout)

    return model
```

接下来我们需要compile训练方法，使HyperOpt知道我们的训练数据、模型和评估指标分别是什么。

```
db_name = "merge"
exp_name = "myexp1"
objective = CompileFN(db_name, exp_name,
                      data_fn=data,
                      model_fn=model,
                      loss_metric="loss", # which metric to optimize for
                      loss_metric_mode="min", # try to maximize the metric
                      valid_split=.2, # use 20% of the training data for the validation set
                      save_dir="model/hyperopt/") # place to store the models
```

最后我们创建Trials对象用来保存测试时所有的数据，帮助我们回溯整个过程。max_evals为评估多少次，即HyperOpt会生成多少套超参数来训练，从而得出最佳的模型。

```
trials = Trials()
best = fmin(objective, hyper_params, trials=trials, algo=tpe.suggest, max_evals=100)
```

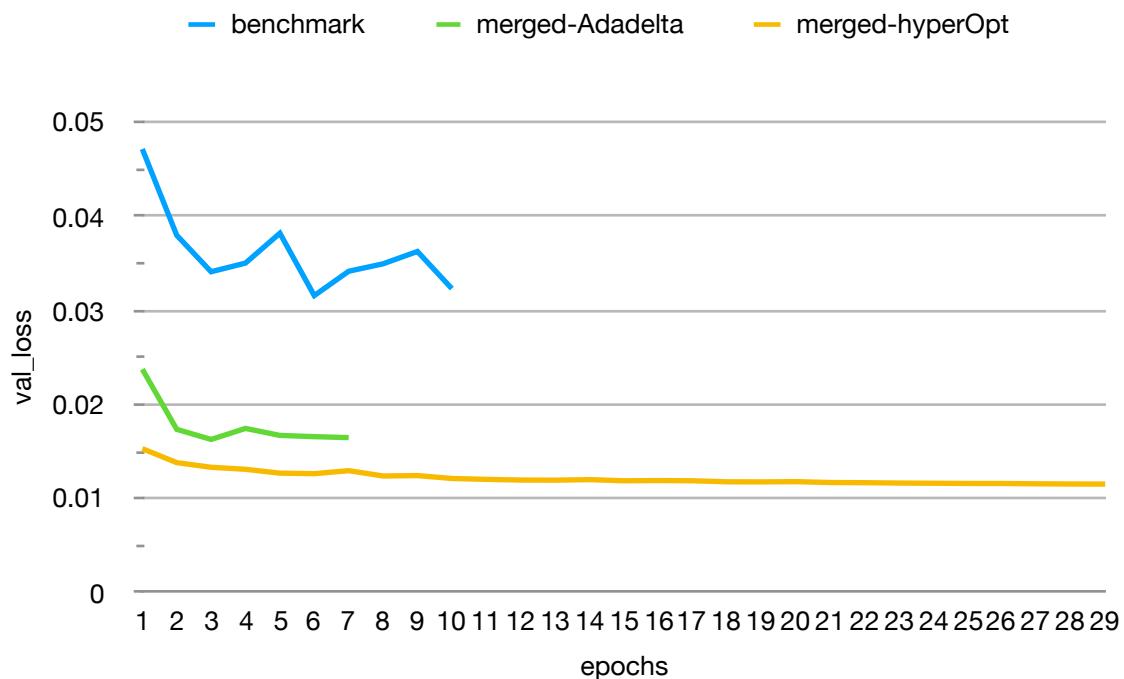
经过100轮循环，每次最多200代的训练后，我们获得最优模型训练的val_loss为：0.01112。对应的超参数为

最优模型(val_loss 0.01112)训练超参数

超参数	值
learning rate	0.0096
decay	0.0084
momentum	0.9

超参数	值
dropout rate	0.5

我们通过Trials回溯训练数据，与基准模型、Adadelta优化的合并模型做对比。（由于最优模型实际训练了200代，若把全部数据都放在图表中会压缩其他两项数据区域。因此只截取最优模型前30代训练记录）

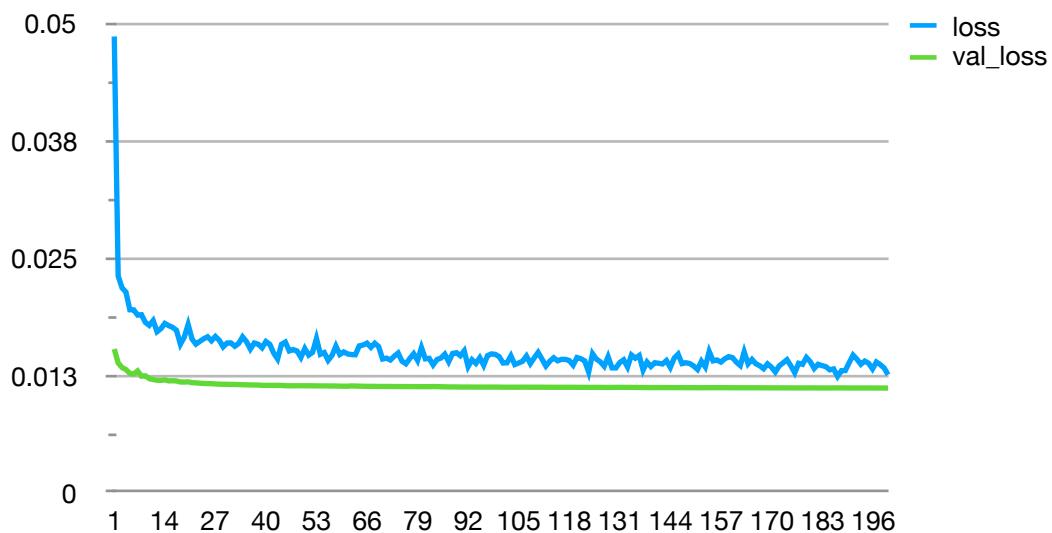


IV. 结果

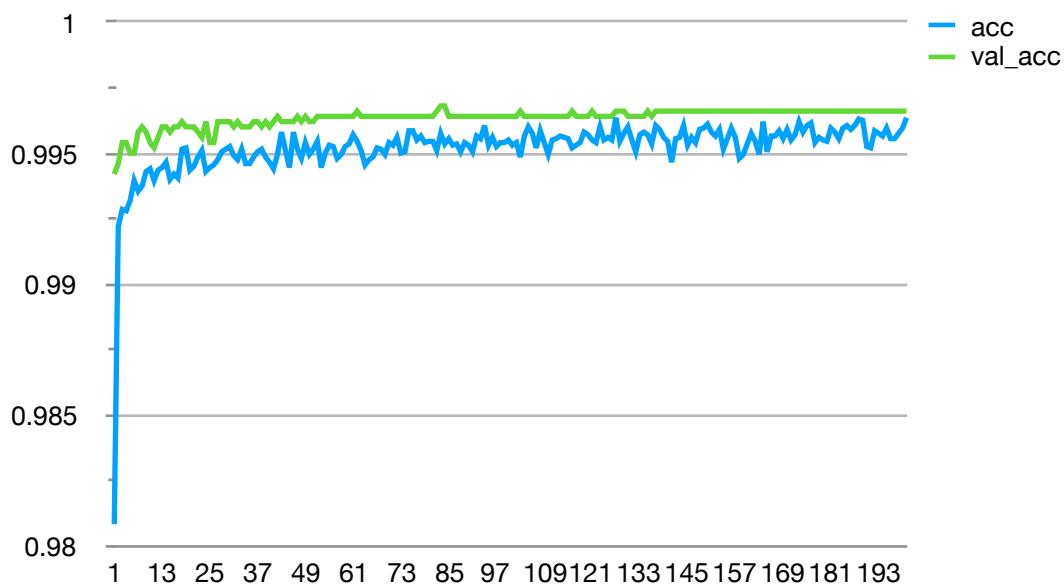
模型的评价与验证

我们最终的模型val_acc为0.9966, val_loss为0.0111。提交Kaggle后的评分为0.03923，位于Kaggle Leadership第13名。模型训练时的历史数据如下：

最终模型训练历史-Loss

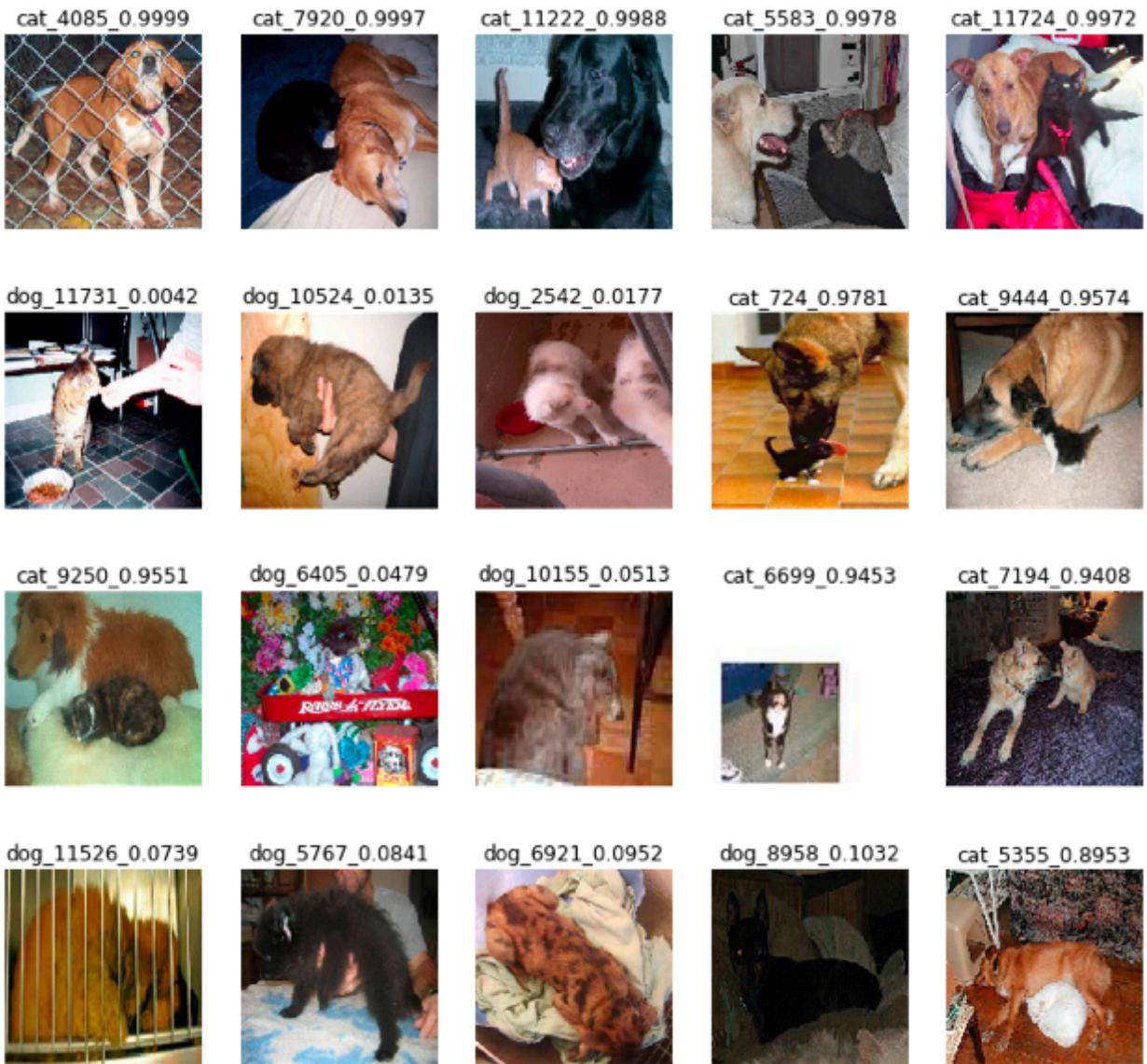


最终模型训练历史-Acc



从图中我们可以发现， val_loss与val_acc在训练后期末产生较大波动，并逐渐趋于平缓。从训练历史数据上看并没有出现过拟合的现象。

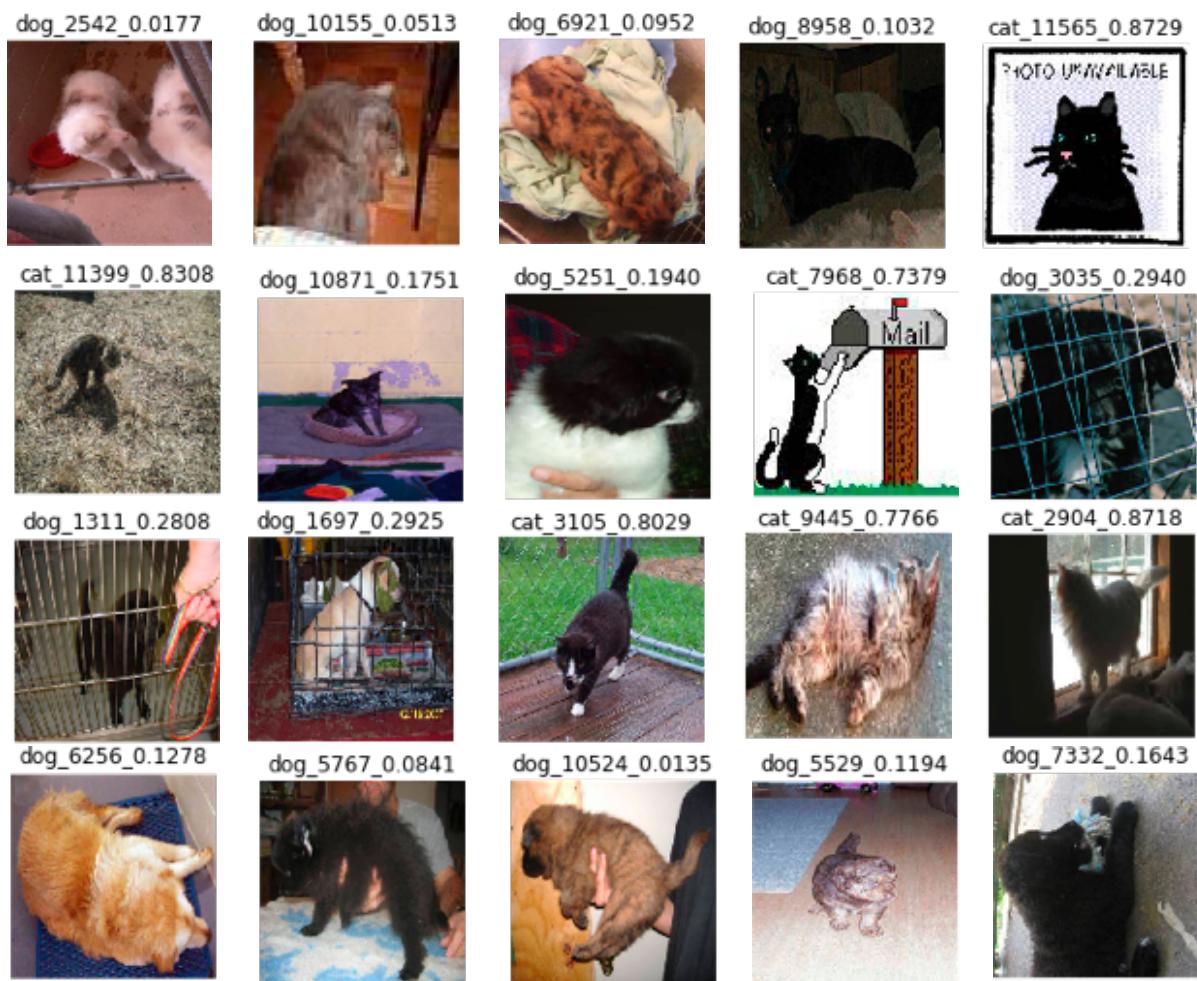
训练完成后，我们使用优化的模型来预测带标签的数据，并且把所有预测结果与标签值相减取绝对值，从大到小排列展示。即预测值越远离正确值，该图片越靠前显示。以下是Top 20张loss最大的图片。标题格式为：标签_编号_预测值



我们发现产生loss最大的20张图片中，并非由于模型问题导致预测错误的有以下两种因素：

1. 图片同时包含猫狗两个元素：模型预测为狗，但标签为猫
2. 图片标签错误：例如第1张和第2行第1张

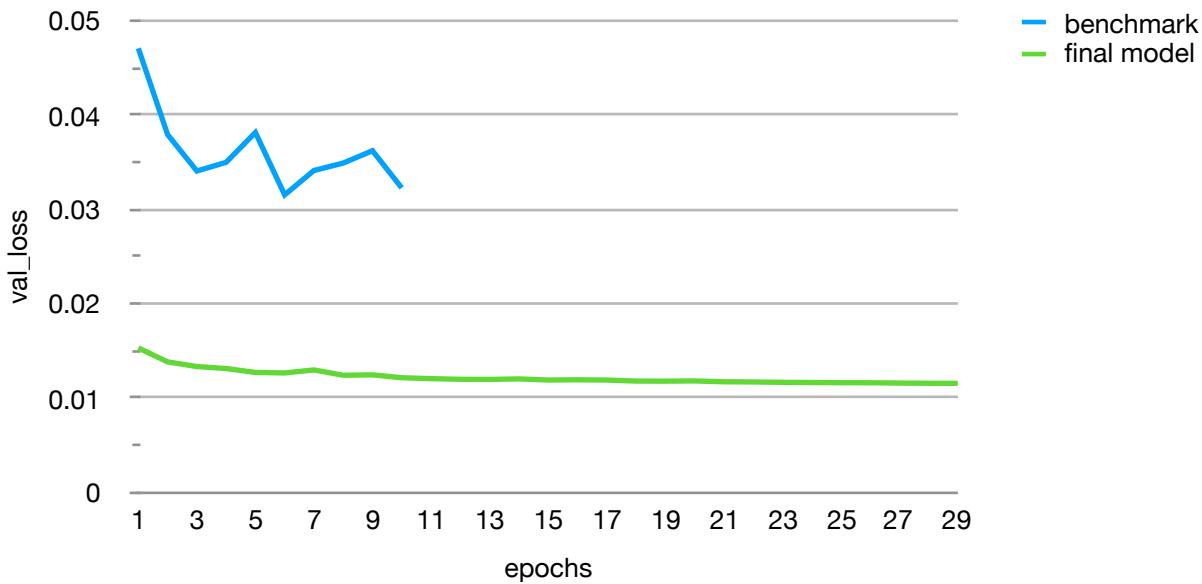
为了避免以上因素干扰，我们从Top 100 张 loss最大的图片中，选出20张非以上原因导致预测不理想的图片：



图中确实有个别图片当缩放至较小尺寸时，会导致我们肉眼也难以观察判断。但其余一些模型预测不理想的图片中，我们肉眼是可以正确判断，这些图片的主要特点总结为：

- 被拍摄物体处于图片一个角落，当预测时由于图片被压缩至更小尺寸，导致主体不清晰，从而模型预测不准确。例如第2行第1张。
- 由于拍摄角度导致主体被旋转超过90度。
- 由于拍摄角度导致主体面部被遮挡（包括俯拍或者被笼子遮挡）
- 卡通形象

基准模型val_loss为0.0323，Kaggle评分为0.0529。相较而言最终模型有更优秀的表现。



模型	优化器	val_loss	Kaggle评分
ResNet50(基准模型)	Adadelta	0.03230	0.05291
ResNet50 切割模型 Finetuning	SGD	0.0295	/
InceptionV3+Xception+ResNet50	Adadelta	0.01647	0.03964
InceptionV3+Xception+ResNet50(最终模型)	SGD(优化参数)	0.01112	0.03923

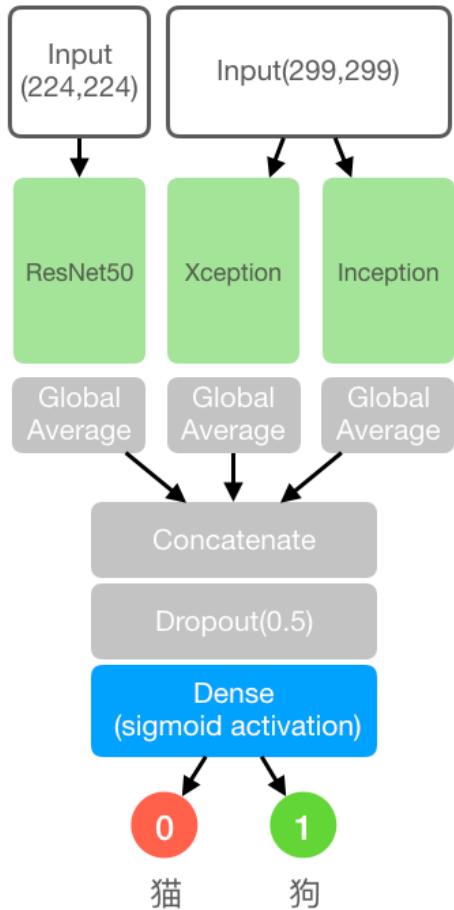
V. 项目结论

结果可视化

相比自建模型从零开始训练，迁移训练给了我们从少量数据中训练出高效模型的可能。这使得图片数据与imagenet数据相似的情况下，可以快速提高模型的预测能力和效率。使得我们的项目从一开始就站来较高的起点。

同时为了进一步提高模型的识别能力，我们对多个预训练模型进行了横向整合。我们从实际训练中可以观察到，任何两个整合的模型的识别能力，都会强于构建该模型的单个预训练模型的识别能力。但当整合的模型个数增加到三个或更多时，一个识别能力较弱的模型可能会对整合的模型产生负面影响。因此我们最后的模型并没有整合VGG16。

我们最终的模型结构如图。为了提高模型识别的准确度，最终模型相较于单个预训练模型更为复杂。由于多个不同的预训练模型使用了不同的输入尺寸，因此模型需要多个输入层，这将导致模型应用时需要前端预先把待识别图片处理多份，增加了应用的复杂程度。



最终模型的Kaggle评分为0.03923,排名第13位。训练时的val_loss为0.0111，相较于基准模型的0.0323，有较大改善。

对项目的思考

猫狗识别是Kaggle上的经典项目，相比自己从零到一构建卷积模型来说，使用迁移训练是一个更有效率的方法。这使得项目的基准模型已经站在了一个较高的起点。

通过阅读迁移学习相关文章和论文，得知训练迁移模型时可以使用到的技术有finetuning、数据增强和模型合并。因此在项目初期规划时，准备最终模型把三者技术结合在一起：即构建一个完整的合并模型网络，开放每个网络最后一个block中的卷积层，并且使用数据增强来训练模型防止过拟合。

在实现过程中，多模型合并使得模型评估Loss得到了显著的改善，好于预期。然而在完成Finetuning的过程中遇到了较大阻碍。在项目中当我们使用完整的模型，通过锁定预训练模型特定层的方法来Finetuning优化时，结果总是不令人满意。最初遇到问题时，以为是开放的层过多，导致模型过快过拟合。所以逐步降低开放层的数量，然而收效并不明显。

之后意识到自建的全连接层由于初始化是随机的，如果直接开放预训练模型层，会导致这些随机初始化的权重过多影响预训练模型。因此最后索性先只开放全连接层做训练。然而效果仍然不理想。

由于之前优化器使用的是Adadelta，所以开始怀疑是自适应learning rate的原因可能导致模型过拟合。所以后期把优化器变更为SGD，并尝试了18组超参数，结果仍然不及预期。

最后当我把预训练模型分割，导出矢量特征作为finetuning模型的输入时，训练结果改善很多。但这导致了一个问题，即该训练方法无法真正使用上数据增强技术。故无法在这次项目中对比数据增强技术可以为模型训练提供多少优化空间，多少有些遗憾。

需要作出的改进

在「模型评价与验证」中我们已经发现目前模型仍然有一些不足：

1. 被拍摄物体处于图片一个角落，当预测时由于图片被压缩至更小尺寸，导致主体不清晰，从而模型预测不准确。
2. 由于拍摄角度导致主体被旋转超过90度。
3. 由于拍摄角度导致主体面部被遮挡（包括俯拍或者被笼子遮挡）
4. 卡通形象

针对第1点，当前预测方法是直接把整张图片压缩至预训练模型使用的尺寸作为输入。因此如果主体在图片中占据的画幅过小，经过压缩后会导致物体过于模糊而无法识别。可以改进的地方是使用物体识别技术得到主体的范围边框，然后把主体裁剪出来并缩放至模型输入的尺寸。

针对第2点，我们可以使用数据增强技术使训练图片产生随机旋转。这样当拍摄角度不在正常范围时，也可以提升识别几率。

对于第3、4点，可以增加训练数据中相类似的图片，增加学习数据。

其次通过这次训练发现Kaggle数据集中有些数据标签和图片内容存在一些问题，之后可以先进行一轮数据清洗或者增加[The Oxford-IIIT Pet Dataset](#)猫狗数据集来使模型获得更有效的训练。

对于模型结果的评估，可以增加可视化的输出。由于此次最终模型使用了多个input，使得当前尚不可使用keras-vis来展示模型的判断依据。下一阶段需研究多个input可视化的方法，来对模型的效果进行更深入的分析。

在应用层面可以尝试制作一个手机App，把已完成训练的模型在手机上运行。这样就可以在现实生活中测试模型的识别效率。并且观察手机上的预测速度。

参考资料

- <http://cs231n.github.io/neural-networks-3/#sgd>
- <https://www.learnopencv.com/keras-tutorial-transfer-learning-using-pre-trained-models>
- <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>
- <https://towardsdatascience.com/neural-network-architectures-156e5bad51ba>
- <http://blog.munhou.com/yong-hyperas-diao-xiao-keras-can-shu>
- <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c>
- https://zh.gluon.ai/chapter_computer-vision/fine-tuning.html
- <https://zhuanlan.zhihu.com/p/25978105>
- <https://raghakot.github.io/keras-vis/visualizations/saliency/>
- <https://www.jianshu.com/p/1deeed4e73b0>