项目报告

Udacity 机器学习（进阶）毕业项目
之猫狗大战

王光宽

2018.05.23

目录

1. 问题定义

1.1项目概述

猫狗大战的项目要求根据 25000 张带标记的图片（其中 12500 张标记为猫，12500 张标记为狗）训练得到一个能区分猫狗图片的模型，用以识别 12500 张不带标记的图片。这是个典型的计算机视觉方面的问题，近几年计算机视觉随着计算机软、硬件的不断发展，涌现了很多优秀的卷积算法，如 AlexNet [1]、VGGNet[2]、ResNet[4] 等。本项目也将结合一些前辈们的算法，通过迁移学习、模型组合等方法，做相应的预测和实验。希望能找到一个好的预测模型，达到较高的预测准确率，达到项目的毕业要求。

参考 kaggle 项目描述 Dogs vs. Cats Redux: Kernels Edition

1.2问题陈述

使用深度学习方法识别一张图片是猫还是狗，最终将测试集中的 12500 张像素大小不一的图片的预测结果（图片是狗的概率，接近 1 为狗，接近 0 为猫），生成 CVS 文件，提交到 Kaggle。

- 输入：一张彩色图片
- 输出：是猫还是狗

这是个图像分类识别的问题，前人已经总结出了很多优化的卷积神经网络模型，在巨人的肩膀上解决此问题，将是事半功倍的。Keras 开源库集成了许多这样的模型，如 VGG[2]、ResNet50[4]、Xception[5]、InceptionV3[3]、InceptionResNetV2[6]等，而且能够下载到这些模型在 ImageNet 数据集上的预训练权重。ImageNet 是 1000 类图像分类问题，训练数据集 126 万张图像，验证集 5 万张，经过这些数据（其中也含大量的猫、狗图像）训练出来的模型有比较好的通用性。

本项目中，我将选用其中的 ResNet50、Xception、InceptionResNetV2 模型先分别做 fine-tuning，并保存 tuning 过程中的权重文件。然后分别选出各自 tuning 过程中结果最优的权重文件为基础，分别导出三个模型的训练集和测试集特征向量（不含输出层的预测结果），再把三个模型的特征向量合成一条特征向量，做为新模型的输入，做新的训练预测。

另外，此项目的训练和预测对计算机的内存和计算能力都有较高要求。因此，本项目的相关模型训练、预测的运行环境均为 aws 上的 GPU 计算实例，p2.xlarge 或 p3.2xlarge。

1.3评价指标

kaggle 上的排名是以 LogLoss 损失函数的分数做为排序依据的，所以模型的性能评估指标将使用用 LogLoss，下面是 LogLoss 的公式：

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

其中：

- y_i 是第 i 个样本的真实值，y_i=0 表示猫， y_i=1 表示狗。
- $\hat{y}_i$是第 i 个样本的预测为狗的概率。

n 为预测样本的数量。

项目要求模型对测试集的预测结果在 kaggle 上的排名在 10%以内，即测试结果对应真实结果的 logloss 小于等于 0.06127（排行榜第 131 名的 score）。LogLoss 损失函数值越小，说明预测的偏差越小，即模型预测准确率越高。

2 分析

2.1数据的探索

本项目的数据来自 kaggle 项目：Dogs vs. Cats Redux: Kernels Edition，它由两个压缩文件 train.zip 和 test.zip 组成，文件具体内容如下：

- **train.zip**
  1）含 25000 张图片，其中被标记猫和狗的图片各一半，根据文件名标记。被标记为猫的图片文件名为：cat.XXXXX.jpg；被标记为狗的图片文件名为：dog.XXXXX.jpg。其中 XXXXX 为 0~12499 的数字。
  2）图片的像素尺寸大小不一。
  3）观察发现有少量图片标签与实际不符，如：dog.1773.jpg，实际并不是狗，所以训练集是含少量异常数据。训练模型前，如果先排除这些异常图片，应该会有利于提升模型的表现。

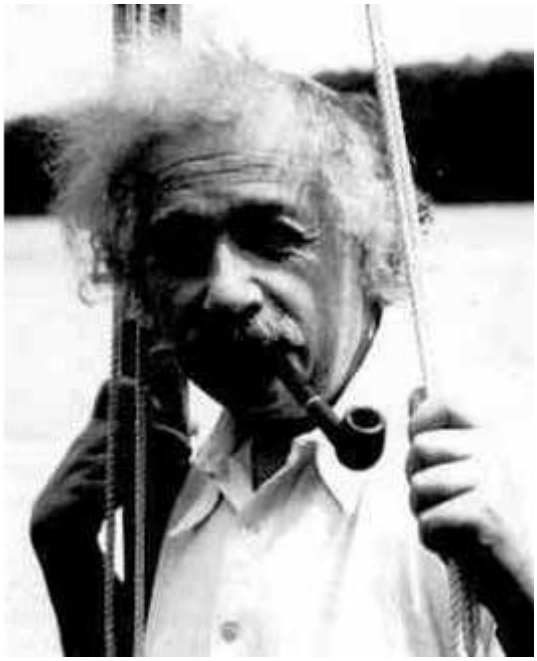

图 1　dog.1773.jpg

- **test.zip**
  1）含 12500 张不带标记的图片，文件名为 XXXXX.jpg，其中 XXXXX 为 1~12500 的数字。
  2）图片的像素尺寸大小不一。
  3）初步观察里面的图片都是猫和狗（不排除有“非猫非狗”或“即有猫又有狗”的图片）。

2.2探索性可视化

训练图片和测试图片都像素尺寸大小不一，它们的图片高度、宽度的散点分布如图 2、图 3 所示。

图 2 训练集图片的宽度、高度散点分布图

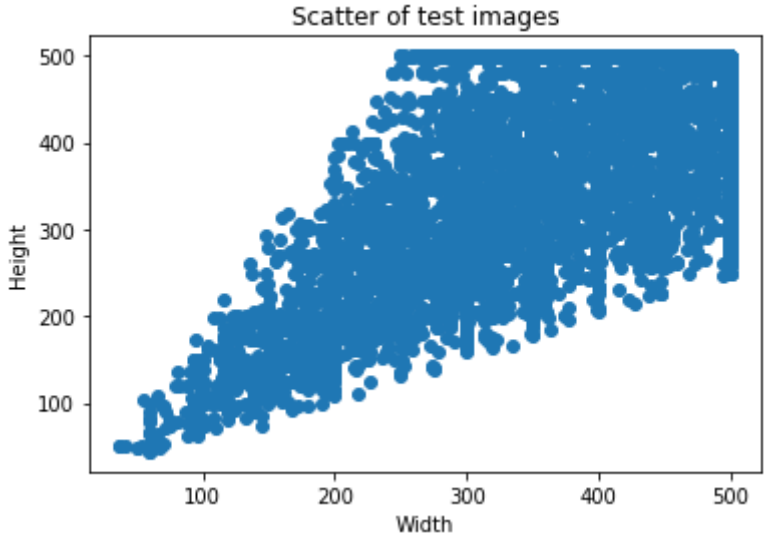

图 3 测试集图片的宽度、高度散点分布图

训练集和测试集图片高度、宽度的均值、标准差、最大值、最小值等如表 1、表 2。

	Height	Width
count	25000	25000
mean	360.47808	404.09904
std	97.019959	109.03793
min	32	42
25%	301	323
50%	374	447
75%	421	499
max	768	1050

表 1　训练集图片高度、宽度值的统计数据

	Height	Width
count	12500	12500
mean	359.93072	404.22448
std	96.757411	109.330874
min	44	37
25%	300	329
50%	374	447
75%	418	499
max	500	500

表 2　测试集图片高度、宽度值的统计数据

根据图 2，可以看出训练集有两张图片像素特别大一点，比其他图片大的一倍左右，这两张图片可以视为异常值，也可以不视为异常值，因为我们后面用到算法，都会对输入图片做一次预处理，以把它缩放到算法要求的输入标准大小。本项目处理过程中，没有将它们按异常值处理。

综合图 2、图 3 和表 1、表 2 一起看，训练集和测试集的图片大小分布是一致的。所以从图片大小的角度看，不需要对训练集做特殊处理。

2.3算法和技术

猫狗识别，简单的说是个二分类问题，适合二分类的算法有很多，比如逻辑回归、支持向量机、决策树、神经网络等，可以说大部分监督学习的算法，都能解决二分类问题。但是猫狗项目的输入是图片数据，图片像素大小就是特征空间的维度，假设我们把图片大小缩放成标准为 224X224X3，则它的维度为 178608，这样的维度数量是对普通的监督学习算法是灾难性的，也不可能通过线性分类的方式找一个分类平面。而从目前已经的情况来看，卷积神经网络（Convolutional Neural Network，CNN）是解决此类问题的最佳方法，所以下面大体介绍一下 CNN 的基本原理。

2.3.1卷积神经网络

2.3.1.1卷积

传统的神经网络需要大量的参数，因为每个神经元都和相邻的神经元相连接。但是在图像识别的问题中，要识别的图像主体可能在图片的不同位置，即平移不变性。这时就引入了卷积的概念。举个例子，现在有一个 4*4 的图像，我们设计两个卷积核（也叫滤波器），看看卷积的计算过程是怎样的？

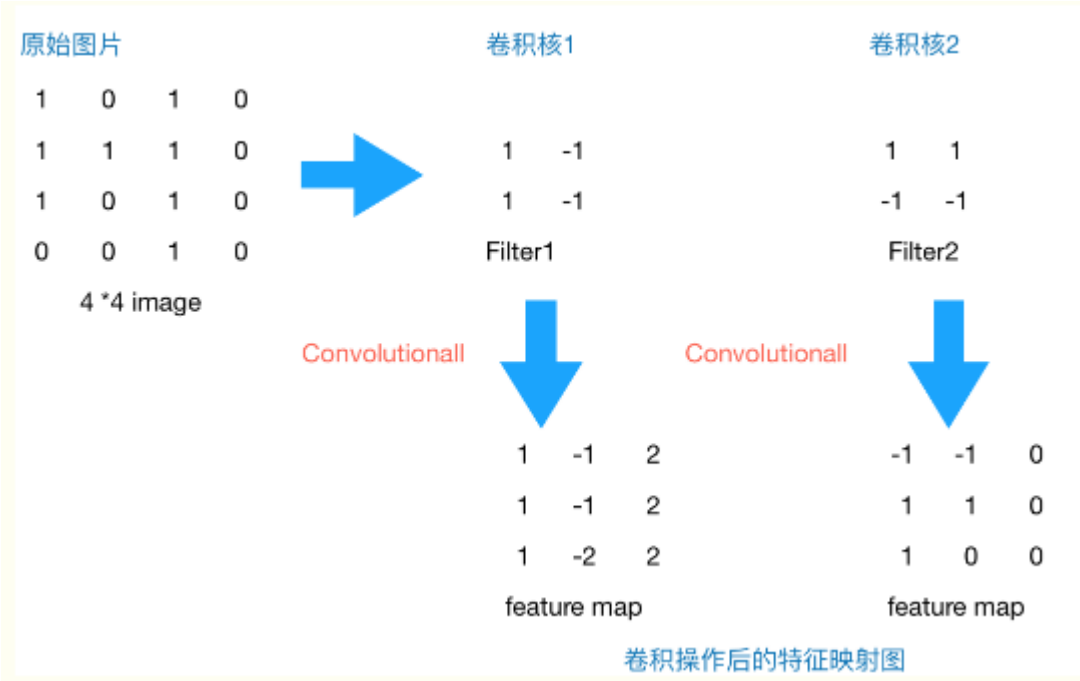

图 4　4*4 image 与两个 2*2 的卷积核操作结果

由上图 4 可以看到，原始图片是一张灰度图片，每个位置表示的是像素值，0 表示白色，1 表示黑色，（0，1）区间的数值表示灰色。对于这个 4*4 的图像，我们采用两个 2*2 的卷积核来计算。设定步长为 1，即每次以 2*2 的固定窗口往右滑动一个单位。以第一个卷积核 filter1 为例，计算过程如下：

```
feature_map1(1,1) = 1*1 + 0*(-1) + 1*1 + 1*(-1) = 1
feature_map1(1,2) = 0*1 + 1*(-1) + 1*1 + 1*(-1) = -1
……
feature_map1(3,3) = 1*1 + 0*(-1) + 1*1 + 0*(-1) = 2
```

feature_map1(1,1)表示在通过第一个卷积核计算完后得到的 feature_map 的第一行第一列的值，随着卷积核的窗口不断的滑动，我们可以计算出一个 3*3 的 feature_map1;同理可以计算通过第二个卷积核进行卷积运算后的 feature_map2，那么这一层卷积操作就完成了。

卷积神经网络与普通神经网络的主要区别就在于，卷积神经网络引入了卷积层和池化层，使得一个神经元只与部分邻层神经元连接。而卷积核是共享权值的，它不仅帮我们实现平移不变性，而且还能大大减少了模型的参数个数。池化（pooling）的方法，通常有均值池化（mean pooling）和最大值池化（max pooling）两种形式。池化可以看作一种特殊的卷积过程。卷积和池化大大简化了模型复杂度，减少了模型的参数。

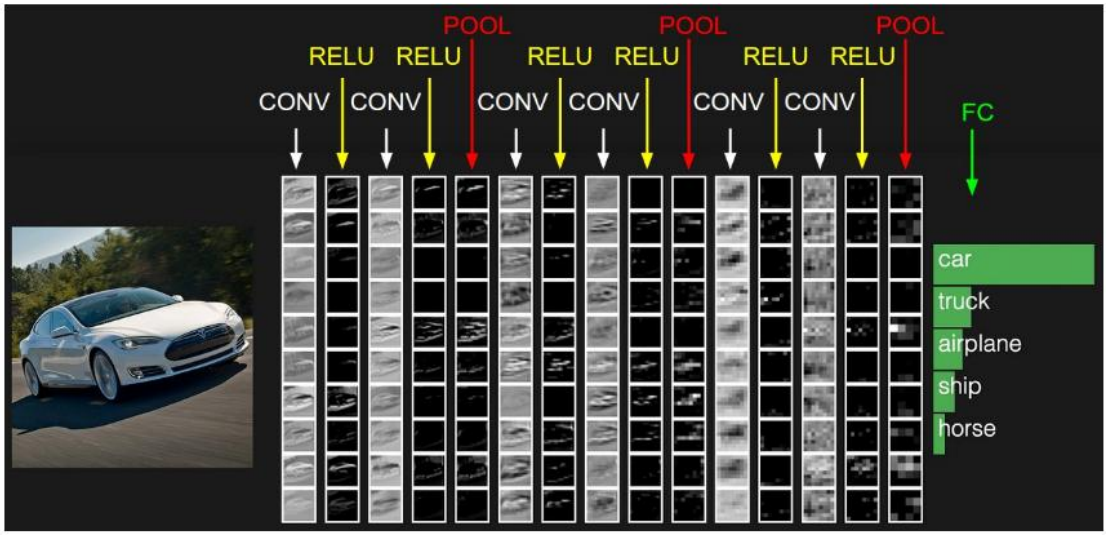

图 5　cs231n 课程里给出的卷积神经网络各个层级结构[8]

卷积神经网络层级结构如上图 5 所示:

- 最左边是：输入层，对数据做一些处理，比如去均值（把输入数据各个维度都中心化为 0，避免数据过多偏差，影响训练效果）、归一化（把所有的数据都归一到同样的范围）、PCA/白化等等。CNN 只对训练集做“去均值”这一步。
- 中间是：
  - CONV：卷积计算层，线性乘积求和。
  - RELU：激励层（激活层），ReLU 是激活函数中的一种。
  - POOL：池化层。
- 最右边是：全连接层

2.3.1.2ReLU 激活函数

ReLU 的公式是$\mathbf{f(x) = max(0, x)}$。激励层的激活函数之所以用 ReLU，而不是 sigmoid、tanh 等函数，是因为相比与 sigmoid 和 tanh，ReLU 有以下优点：

1）**计算高效**：采用 sigmoid 等函数，算激活函数时（指数运算）计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大；而采用 ReLU 激活函数，整个过程的计算量节省很多；

2）**没有饱和和梯度消失问题**：对于深层网络，sigmoid 函数反向传播时，很容易就会出现梯度消失的情况（在 sigmoid 接近饱和区时，变换太缓慢，导数趋于 0，这种情况会造成信息丢失，从而无法完成深层网络的训练）。ReLU 会使一部分神经元的输出为 0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生。

3）**快速收敛**：实践表明 ReLU 收敛速度比 sigmoid 或 tanh 快 6 倍。

但 ReLU 也有缺点，就是训练的时候很“脆弱”，神经元很容易就“die”了，比如：一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了。如果这个情况发生了，那么这个神经元的梯度就永远都会是 0。实际操作中，如果你的 Learning Rate 很大，那么很有可能你网络中的 40%的神经元都“dead”了。使用较小的 Learning Rate 时，可以降低发生这种情况的概率。

2.3.1.3GAP

图 6 的最右边的全连接层，还有一种可替代的方案，在 Network in Network 论文[9]中提出，即全局平均池化（GAP：Global Average Pooling）。比如在 1 个 N 分类的任务，则网络的最后的 feature maps 的个数为 N 个，全连接的做法是将这 N 个 feature maps 合并成一个向量，再输入到 softmax 层中得到对应的每个类别的得分，如图 7 左侧所示。全连接层有两个缺点，就是：1）参数量大，需要大量的计算开销；2）过多的参数会引起过拟合的情况。GAP 的算法则是将每个 feature map 直接求均值，它把 avg pooling 的窗口大小设置成 feature map 的大小，输出长为 N 的向量，再输入到 softmax 层中得到对应的每个类别的得分，如图 6 右侧所示。这么做的真正意义是：对整个网路在结构上做正则化，防止过拟合。它相当于剔除了全连接层黑箱子操作的特征，直接赋予了每个 channel 实际的内别意义。

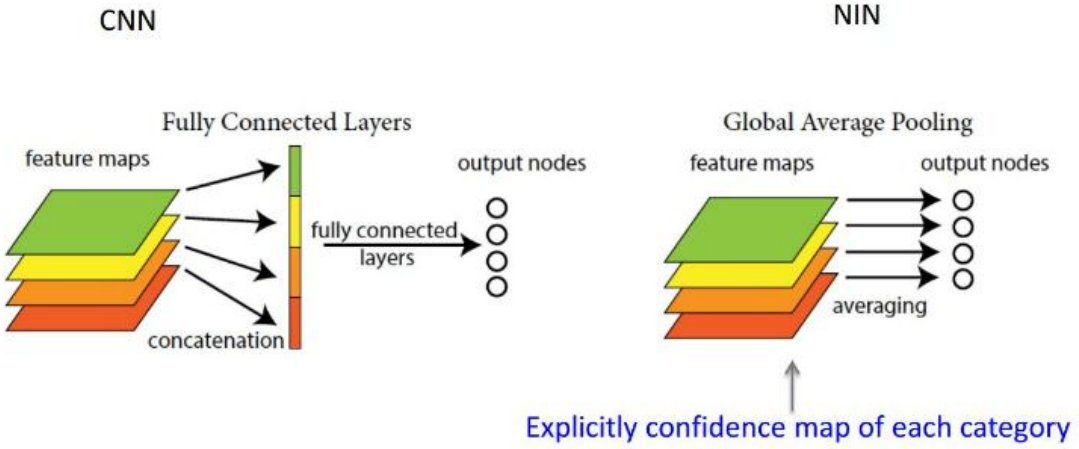

图 6　GAP 与全连接的比较[10]

2.3.2 技术

项目中最主要用到的技术将是 Keras 库，Keras 是一个用 Python 编写的高级神经网络 API，它能够以 TensorFlow, CNTK, 或者 Theano 作为后端运行。Keras 的开发重点是支持快速的实验。具有用户友好、模块化、易扩展等优点（以下内容来自 keras 中文官网）。

- **用户友好。** Keras 是为人类而不是为机器设计的 API。它把用户体验放在首要和中心位置。Keras 遵循减少认知困难的最佳实践：它提供一致且简单的 API，将常见用例所需的用户操作数量降至最低，并且在用户错误时提供清晰和可操作的反馈。
- **模块化。** 模型被理解为由独立的、完全可配置的模块构成的序列或图。这些模块可以以尽可能少的限制组装在一起。特别是神经网络层、损失函数、优化器、初始化方法、激活函数、正则化方法，它们都是可以结合起来构建新模型的模块。
- **易扩展性。** 新的模块是很容易添加的（作为新的类和函数），现有的模块已经提供了充足的示例。由于能够轻松地创建可以提高表现力的新模块，Keras 更加适合高级研究。
- **基于 Python 实现。** Keras 没有特定格式的单独配置文件。模型定义在 Python 代码中，这些代码紧凑，易于调试，并且易于扩展。

如果你在以下情况下需要深度学习库，请使用 Keras：

- 允许简单而快速的原型设计（由于用户友好，高度模块化，可扩展性）。

- 同时支持卷积神经网络和循环神经网络，以及两者的组合。
- 在 CPU 和 GPU 上无缝运行。

2.4基准模型

根据项目要求，模型对测试集的预测结果在 kaggle 上的排名在 10%以内，即测试结果对应真实结果的 logloss 小于等于 0.06127（排行榜第 131 名的 score，排行榜总人数为 1314）。所以此项目可以不参考基准模型，参考前面的检验基准阈值：logloss=0.06127 即可。logloss 比这个基准阈值小，则认为是符合要求的模型，否则需要继续优化或重建模型。

3 方法

3.1数据预处理

3.1.1目录说明

train 目录下的文件虽然有标记是猫还是狗，但没有分目录存放。keras 的 ImageDataGenerator 读取数据时，需要将不同种类的图片放在不同的目录，所以需要分开存放。

所有数据放 data 目录下，train 目录为原始训练集图片，train2 下为按猫狗分目录后，删除异常值后的图片（连接文件），train3 和 validation 目录下的文件分别为按 20%比例拆分后的训练集和验证集图片，test 目录为测试集数据。具体说明见下面表 3，处理过程参见 prep_data_process.ipynb。

一级目录	二级目录	三级目录	说明
data	train		25000 张原始训练集图片
	train2	dog	训练数据，12479 张图片
		cat	训练数据，12451 张图片
	train3	dog	训练数据，9983 张图片
		cat	训练数据，9961 张图片
	validation	dog	验证数据，2496 张图片
		cat	验证数据，2490 张图片
	test	test	测试集数据，12500 张图片
	anormal	cat	cat 的异常图片
		dog	dog 的异常图片
		top100	top_n=100 时，从上一级 cat 和 dog 目录复制而来
		top50	top_n=50 时，从上一级 cat 和 dog 目录复制而来
		diff	top50 中 cat、dog 目录下的文件分别减去 top100 中 cat、dog 目录下的文件

表 3　数据集目录划分详细情况

3.1.2异常图片检测

train 目录下的文件，有小部分是非猫非狗的图片，需要进行辨别，排除。

检测方法为：使用 keras 的 InceptionResNetV2、InceptionV3、Xception 三个预训练模型，分别对猫和狗的训练集进行预测，并将三个模型的预测结果取并集。之

所以用三个模型，是因为单个模型所学习到的特征可能会有偏向性，结合多个模型可以起到互相补充的做用。

预测时，我先用 100 张图片做前期测试，发现 top_n 取值较小时，预测的准确率不是太高，同时为了减少误判的可能（将标记正确的图片，判断为非异常数据），最后选取 top_n=100 和 top_n=50 做比较。

实现过程参见附件：pick_up_anormal_images_batch_dog.ipynb、pick_up_anormal_images_batch_cat.ipynb 和 prep_data_process.ipynb。

3.1.2.1执行过程

1) 执行过程中先置 top_n=100，运行 pick_up_anormal_images_batch_dog.ipynb 和 pick_up_anormal_images_batch_cat.ipynb，再分别把 anormal/cat 和 anormal/dog 下的选出的图片复制到 top100/cat 和 top100/dog 目录下；
2) 再置 top_n=50，运行 pick_up_anormal_images_batch_dog.ipynb 和 pick_up_anormal_images_batch_cat.ipynb，再分别把 anormal/cat 和 anormal/dog 下的选出的图片复制到 top50/cat 和 top50/dog 目录下；
3) 最后执行 prep_data_process.ipynb 中的 get_diff_image 函数，选出差异图片。

改进：以上 1、2 两步，由 pick_up_anormal_images.ipynb 一个文件代码完成。

3.1.2.2结果说明

3.1.2.2.1 top_n=100 时

cat 目录中选出的异常图片中如下图 7，共 43 张：

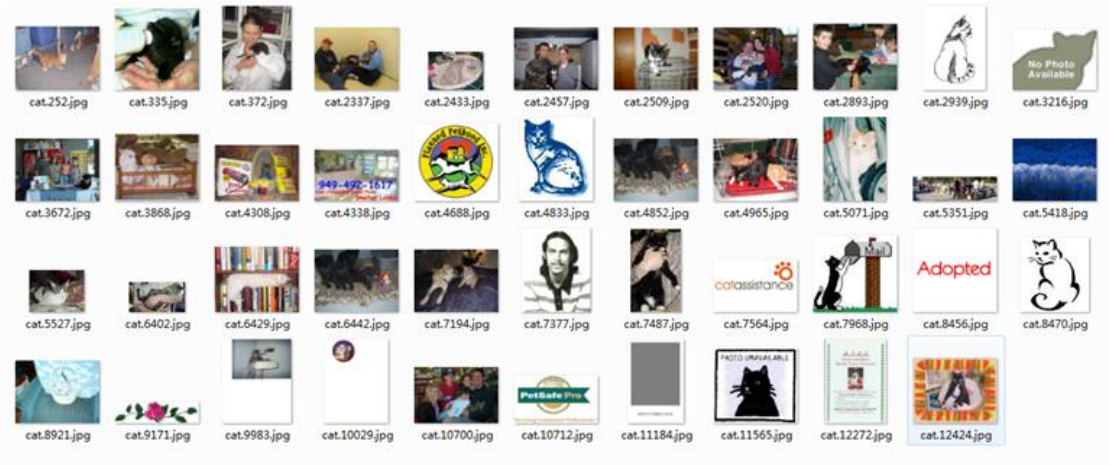

图 7　top_n=100 时，选出的猫的异常图片

dog 目录中选出的异常图片中如下图 8，共 12 张：

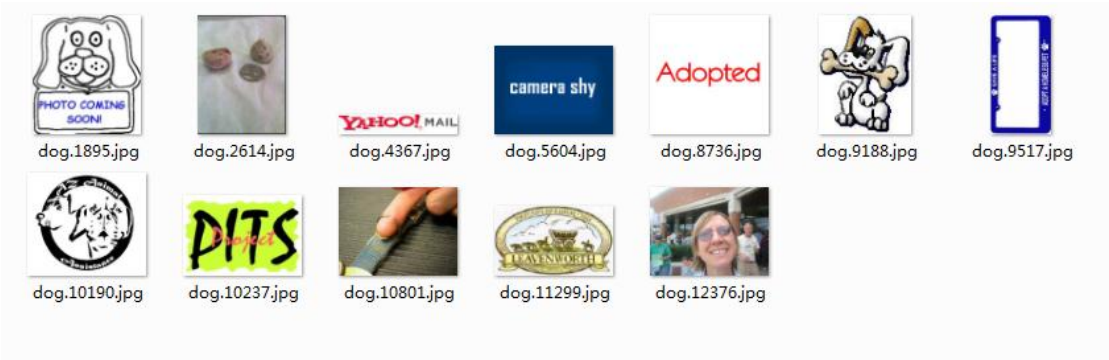

图 8　top_n=100 时，选出的狗的异常图片

3.1.2.2.2　top_n=50 时

由于 top_n=100 时，dog 目录中选出的异常图片不含《开题报告》中的 dog.1773.jpg，所以用 top_n=50 再试一次。

cat 目录中选出的异常图片中如下图 9，共 99 张　：

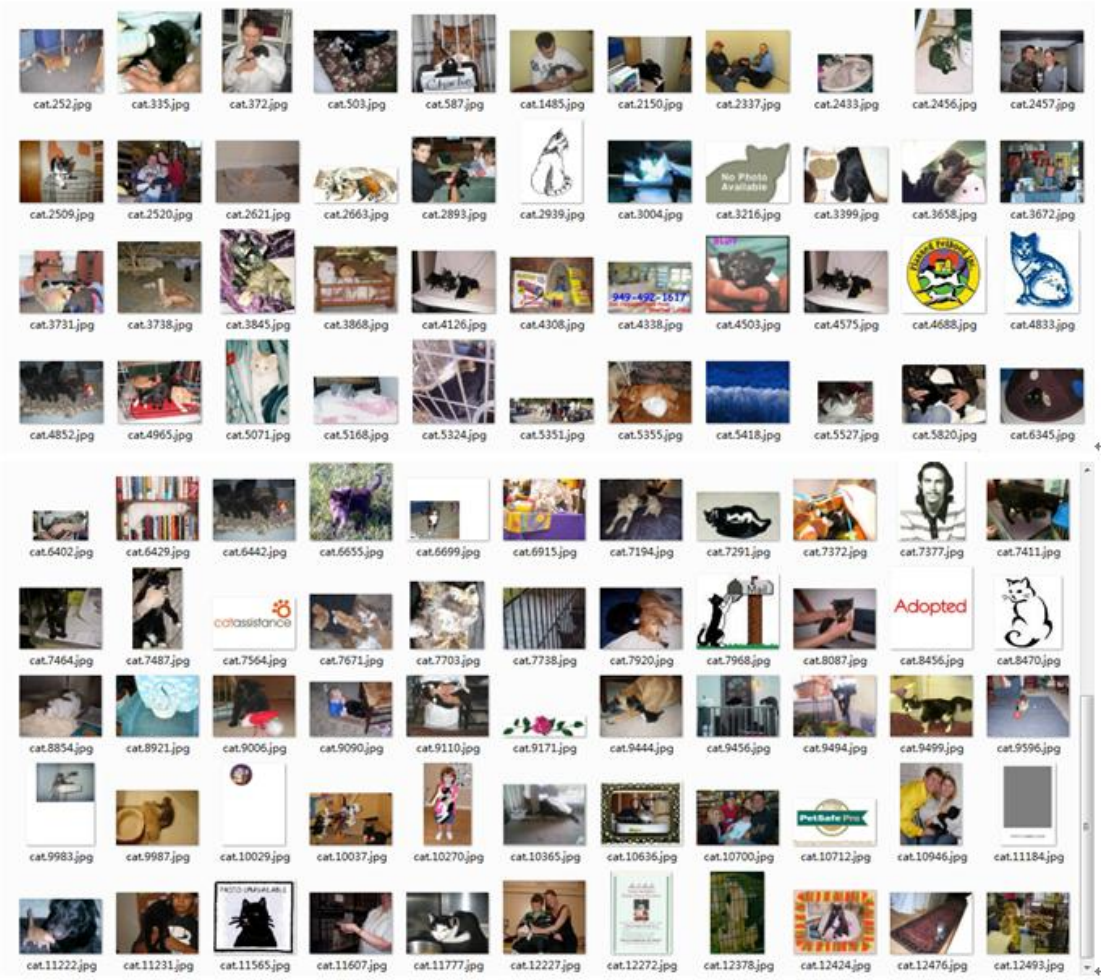

图 9　top_n=100 时，选出的猫的异常图片

dog 目录中选出的异常图片中如下图 10，共 21 张：

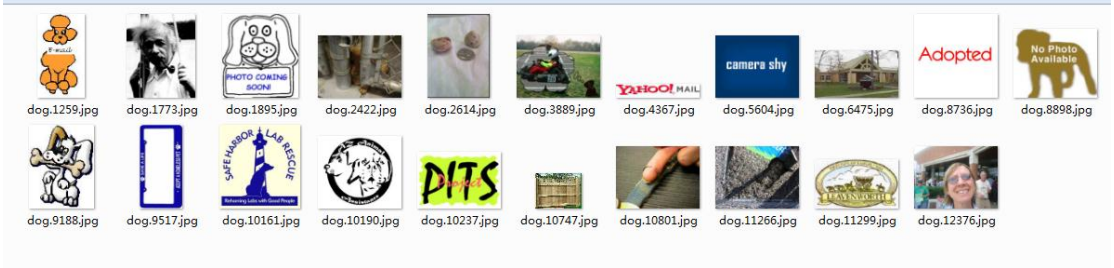

图 10　top_n=100 时，选出的猫的异常图片

3.1.2.2.3　差异

Top50 比 top100 多选出的图片，如下图 11、图 12。

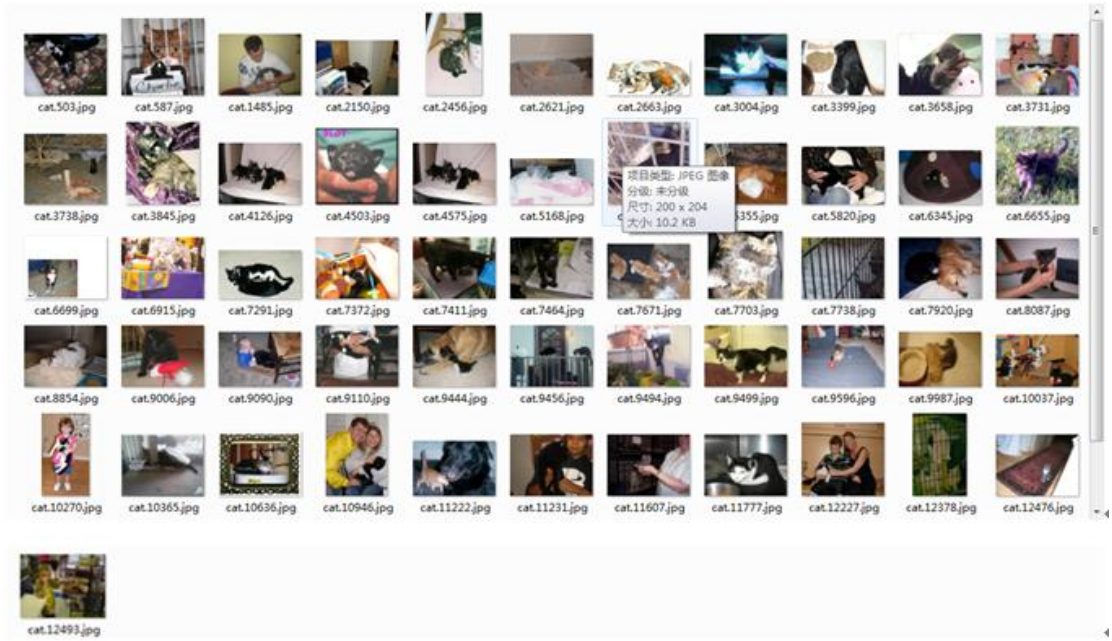

图 11　top_n 为 100 和 50 时，选出的猫异常图片的差异

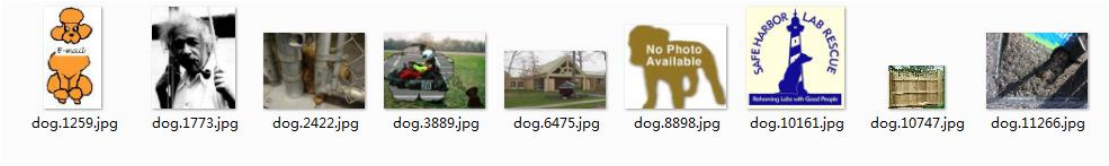

图 12　top_n 为 100 和 50 时，选出的狗异常图片的差异

3.1.2.3结论与决策

可以看出 top50 和 top100 选出的图片，cat 的差异部分基本都是含猫的；但 dog 的差异部分有很多确实是非狗的图片。所以我决定在猫的训练集中删除 top100 挑出的 43 张图片，狗的训练集中删除 top50 挑出的 21 张图片。为了让删除后的数据集总量仍是 10 的倍数（为

了按比例拆分验证集时，不出现小数），在 data/anormal/diff/cat 中再随便选 6 张从训练集中删除，这样一共在训练集中删除 70 张。代码参考附件 prep_data_process 中的 move_anormal_images 函数。

3.2执行过程

本项目的执行过程总体上分两部分：第一部分是，在 keras 库中选取在 ImageNet 上 Top-1 准确率较高的 3 个模型 Xception、ResNet50、InceptionResNetV2（准确率情况见图 13），分别进行迁移学习。第二部分是通过选取的三个预训练模型，分别导出特征向量，然后以此为输入，搭建一个简单的二分类模型进行训练、预测。

模型概览

模型	大小	Top-1 准确率	Top-5 准确率	参数数量	深度
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

Top-1 准确率和 Top-5 准确率都是在 ImageNet 验证集上的结果。

图 13　keras 预训练模型在 ImageNet 验证集上的结果，来自 keras 中文官网

3.2.1执行结果

先给出项目执行过程中，各个训练场景的结果，后面再针对各个模型的训练过程做说明。

模型	主要参数	放开或冻结层	val_loss	kaggle 得分
Xception	Dropout=0.5 Dense 卷积核使用 l2 正则函数 优化算法：Adam	只放开输出层	0.09292	0.10825
		冻结前 97 层	**0.01005**	**0.03871**
InceptionResNetV2	Dropout=0.25 Dense 不含正则化参数 优化算法：Adam	只训练输出层	0.06561	0.08848
		冻结前 698 层	0.01554	0.03838
		冻结前 618 层	0.01521	0.03878

	VER=4	冻结前 499 层	0.01434	0.03835
	Dropout=0.5 Dense 卷积核使用 l2 正则函数 优化算法：Adam VER=5	只训练输出层	0.07577	0.09462
		冻结前 698 层	**0.02397**	**0.03821**
		冻结前 618 层	0.02235	0.03831
		冻结前 746 层	0.04462	0.04567
ResNet50	Dropout=0.5 Dense 卷积核使用 l2 正则函数 优化算法：Adam	只训练输出层	0.06526	0.07115
		冻结前 164 层	0.04574	0.05814
		冻结前 142 层	0.03366	0.05225
		冻结前 112 层	**0.02828**	**0.04849**
融合模型	优化算法：adadelta	权重文件：ImageNet	0.01628	0.03676
	优化算法：Adam	权重文件：ImageNet	0.01678	0.03707
	优化算法：adadelta	权重文件：fine-tuning	0.00727	0.03668
	优化算法：Adam	权重文件：fine-tuning	**0.00826**	**0.03623**

表 4　各个训练场景的训练、预测结果，标红加粗的数据为各自模型上在 kaggle 上的最好得分

说明：融合模型中的权重文件“fine-tuning”是指 Xcetpion、InceptionResNetV2、ResNet50 三个模型迁移学习后得到的各自的最佳权重文件。

3.2.2 单模型

3.2.2.1Xception

3.2.2.1.1　模型简介

Xception 是在 Inception 基础上演进的一种模型，Inception 模型的核心结构是 Inception 模块，简化的 Inception 模块结构如图 14 所示。Inception 模块首先使用 1*1 的卷积核将特征图的各个通道映射到一个新的空间，在这一过程中学习通道间的相关性；再通过常规的 3*3 或 5*5 的卷积核进行卷积，以同时学习空间上的相关性和通道间的相关性。此时，通道间的相关性和空间相关性仍旧没有完全分离，也即 3*3 或 5*5 的卷积核仍然是多通道输入的。

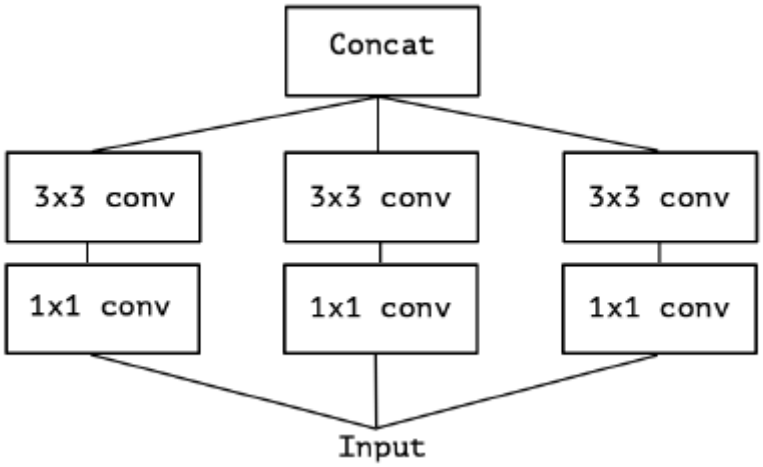

图 14　简化的 Inception 模块结构

Xception 模块则进一步增多 3*3 或 5*5 的卷积分支的数量，使它与 1*1 的卷积的输出通道数相等，如图 15 所示。使得每个 3*3 的卷积核对应一个通道的特征图，达到通道间的相关性和空间上的相关性完全分离的效果。所以 Xception 模块也叫做“极致的 Inception（Extream Inception）”模块。Xception 模型的结构图可以参考附件 model_Xception.png。

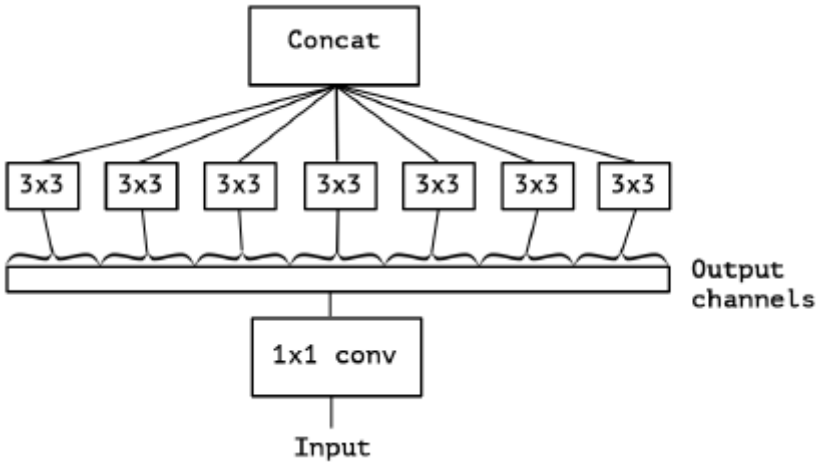

图 15　极致的 Inception 模块结构

3.2.2.1.2 迁移学习

3.2.2.1.2.1 模型搭建

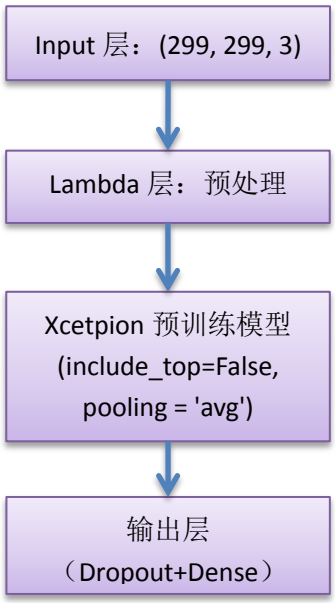

图 16　Xception 迁移学习模型

使用 keras 的 Xception 预训练模型进行 fine-tuning，搭建模型过程如图 16 所示，说明如下：

1) 定义输入层，输入图片需要由 keras 提供的 applications 模块封装的 xception.preprocess_input()函数进行预处理。
2) 加载预训练模型及其权重文件（不含输出层，含 pooling 层），并设置预训练模型的所有层为不可训练。
3) 重新定义输出层，输出层设计为： 1 个 dropout+1 个全连接层，全连接层激活函数为 sigmod
4) 将加载的预训练模型和新定义的输出成组成一个新的模型；只训练最新定义的输出层。

3.2.2.1.2.2 数据读取

使用 keras 库提供的 ImageDataGenerator 生成器读取训练集数据和测试集数据。ImageDataGenerator 同时提供数据增强的功能，即可以将原始图片进行旋转、平移、缩放等操作，产生更多的训练样本。Xception 模型的训练集数据用了增强的功能，代码片段如下图 17，其中 train_data_dir 为 data/train3，valid_data_dir 为 data/validation。

```
gen = ImageDataGenerator(rotation_range=30,  #旋转数据增强
                         width_shift_range=0.2,
                         height_shift_range=0.2,
                         shear_range=0.2,
                         zoom_range=0.2,
                         horizontal_flip=True)
val_gen = ImageDataGenerator()
train_generator = gen.flow_from_directory(train_data_dir, (299, 299), shuffle=True,
                                          batch_size=64,class_mode='binary')
valid_generator = val_gen.flow_from_directory(valid_data_dir, (299, 299), shuffle=True,
                                          batch_size=32,class_mode='binary')
```

图 17　Xception 模型使用 ImageDataGenerator 生成数据的代码

3.2.2.1.2.3 训练

用 fit_generator 函数完成训练过程，Xception 模型的训练代码如图 18 所示，训练参数说明：

1）训练 10 代；

2）只有验证集 loss 下降时才保存模型权重文件；

3）steps_per_epoch=277 (nb_train_samples=19944, batch_size=72)

4）validation_steps=4986//72=69

```
#训练模型并保存在验证集上损失函数最小的权重
filepath="xception-best_weight_freeze.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min',save_weights_only=True)
callbacks_list = [checkpoint]

history=model.fit_generator(
        train_generator,
        steps_per_epoch=nb_train_samples//batch_size,
        epochs=10,
        validation_data=valid_generator,
        validation_steps=nb_validation_samples//batch_size,
        callbacks = callbacks_list)
```

图 18　Xception 模型的训练代码

3.2.2.1.2.4 放开输出层训练结果

训练输出中间结果，如下图 19：

```
Epoch 00006: val_loss improved from 0.12011 to 0.11273, saving model to xception-best_weight_freeze.h5
Epoch 7/10
277/277 [==============================] - 452s 2s/step - loss: 0.1186 - acc: 0.9713 - val_loss: 0.1058 - val_acc: 0.9878

Epoch 00007: val_loss improved from 0.11273 to 0.10581, saving model to xception-best_weight_freeze.h5
Epoch 8/10
277/277 [==============================] - 452s 2s/step - loss: 0.1136 - acc: 0.9682 - val_loss: 0.1042 - val_acc: 0.9864

Epoch 00008: val_loss improved from 0.10581 to 0.10415, saving model to xception-best_weight_freeze.h5
Epoch 9/10
277/277 [==============================] - 453s 2s/step - loss: 0.1037 - acc: 0.9716 - val_loss: 0.0948 - val_acc: 0.9878

Epoch 00009: val_loss improved from 0.10415 to 0.09484, saving model to xception-best_weight_freeze.h5
Epoch 10/10
277/277 [==============================] - 464s 2s/step - loss: 0.0985 - acc: 0.9736 - val_loss: 0.0929 - val_acc: 0.9882

Epoch 00010: val_loss improved from 0.09484 to 0.09292, saving model to xception-best_weight_freeze.h5
```

图 19　Xception 模型只放开输出层时的训练部分结果

学习曲线，如图 20 所示：

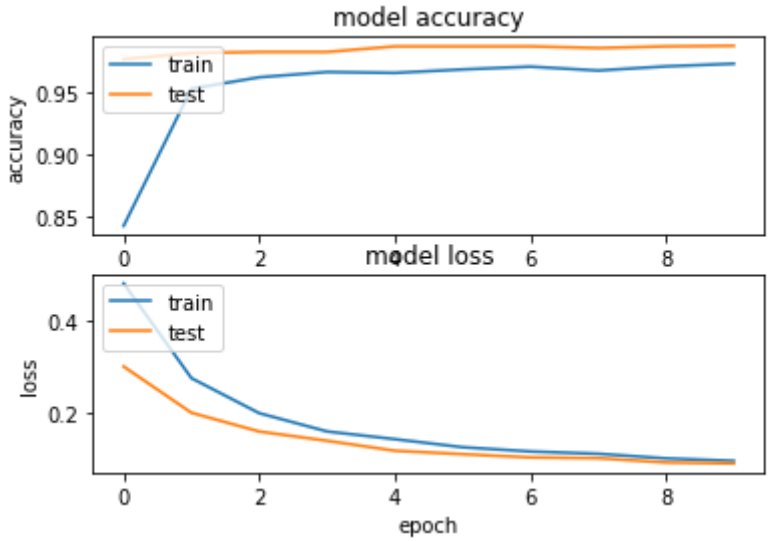

图 20　Xception 模型只放开输出层时的学习曲线

从训练结果看，每代训练 val_loss 都在下降，其实可以继续增加训练代数；从 loss 的学习曲线看，训练集 loss 和验证集 loss 已经比较接近了，更多的训练代数，不一定能提高太多 val_loss，反而可能会出现过拟合，所以没有重新训练。

3.2.2.1.2.5 预测

项目要求输出的测试结果存到 csv 文件中，第一列为与预测图片文件名相对应的从 1 到 12500 的 id 编号，第二列为'label'，即预测结果。所以读入测试文件时，也按文件名从 1 到 12500 顺序读入，再做预测。同时为了降低提交到 kaggle 时的 Logloss，将每个预测值限制到[0.005, 0.995]，参考培神的思路。具体参见如下图 21 的代码：

```
def predict_on_model(n, width, heigth, test_data_dir, model, weight, output_name):
    x_test = np.zeros((n,width,heigth,3),dtype=np.uint8)

    for i in tqdm(range(n)):
        img = load_img(test_data_dir+"/test/"+'/%d.jpg' % (i+1))
        x_test[i,:,:,:] = img_to_array(img.resize((width,heigth),Image.ANTIALIAS))

    model.load_weights(weight)
    y_test = model.predict(x_test, verbose=1)
    y_test = y_test.clip(min=0.005, max=0.995)

    df = pd.read_csv("sample_submission.csv")
    for i in tqdm(range(n)):
        df.set_value(i, 'label', y_test[i])
    df.to_csv(output_name, index=None)
    df.head(10)
```

```
predict_on_model(12500, 299, 299, test_data_dir, model, "xception-best_weight_freeze.h5", "pred-xception-freeze-2.csv")
```

图 21　Xception 模型的预测代码

3.2.2.1.2.6 改进-开放部分层 fine-tuning

以前面训练保存的 xception-best_weight_freeze.h5 权重文件为基础，放开 97 以上层重新训练、预测。只训练 5 代，得到 val_loss 为 0.01005，可以说比是个很不错的分数了。预测结果提交到 kaggle 的得分为 0.03871，已经比基准模型要求的 0.06127 低了。

实验的具体结果看表 4 中的统计。

3.2.2.2 InceptionResNetV2

3.2.2.2.1 模型简介

InceptionResNetV2 模型是在 InceptionV3 模型基础上结合残差网络（ResNet）的思路，变化而来的一种模型，结合残差连接（Residual connections）可以使得我们能够训练更深层的网络，从而使模型获得更好的表现。我们此次实验用到的 keras 库中的 InceptionResNetV2 模型有 780 多层，比论文（Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning）中介绍的要多出将近 1 倍，主要几个 Inception-ResNet 的模块都扩展了一倍。下面图 22、图 23 分别是论文中的 InceptionResNetV2 结构示意图和本次项目使用的 InceptionResNetV2 结构示意图。更多 InceptionResNetV2 模型的介绍可参考论文[6]

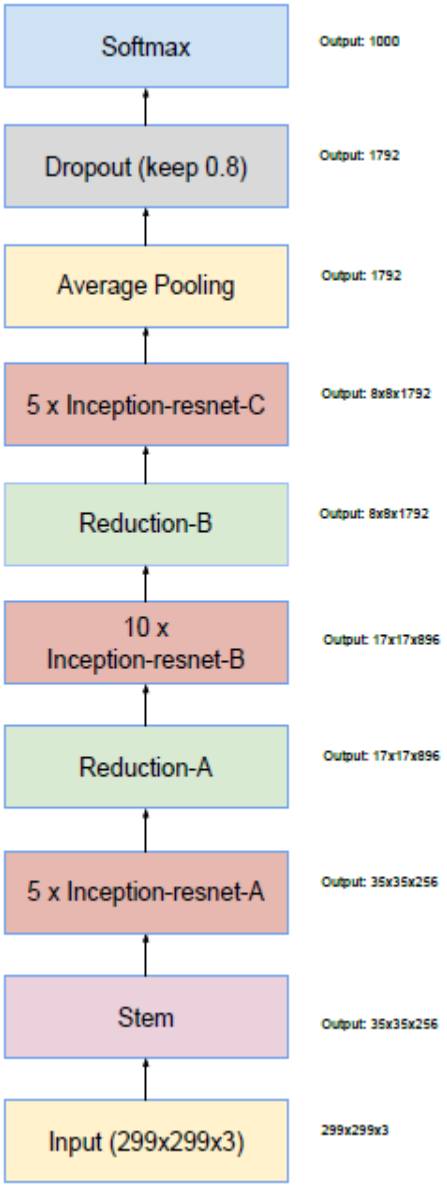

图 22　论文中 InceptionResNetV2 模型结构

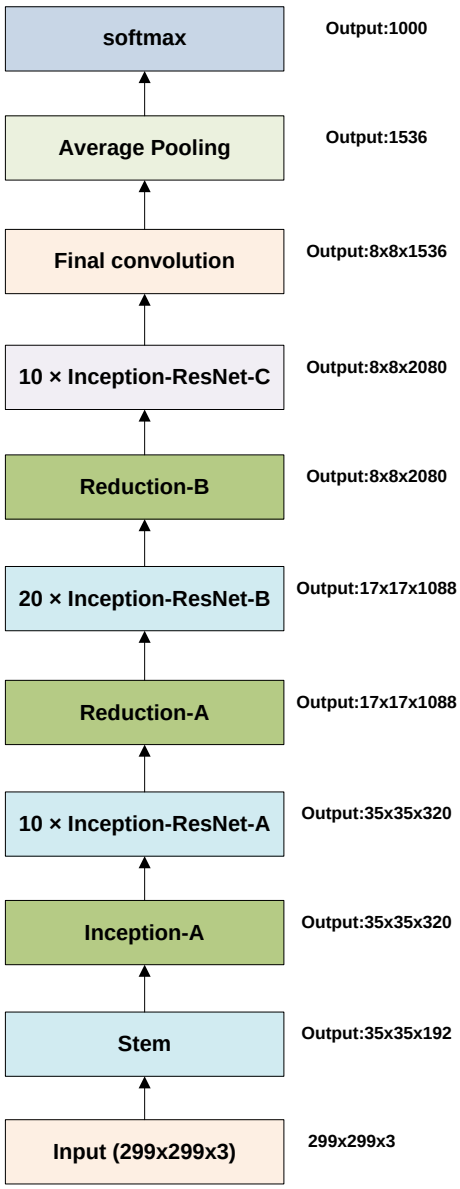

图 23　keras 库中 InceptionResNetV2 模型结构

3.2.2.2.2　迁移学习

整体说明：

使用 keras 的 InceptonResNetV2 预训练模型进行 fine-tuning，分别进行下面四种情况的 fine-tuning：

1. 只训练自定义的输出层
2. 冻结前 698 层，训练后面的层
3. 冻结前 618 层，训练后面的层
4. 冻结前 499 层，训练后面的层

执行过程和 Xception 的模型的迁移学习类似，但也有些步骤，做了一些优化和调整，所以主要讲一下有调整的地方。

3.2.2.2.2.1 模型搭建

模型的搭建过程和结构，与 Xception 迁移学习的基本一样[见 3.2.2.1.2.1 章节]，不同之处只是中间的预训练模型换成了 InceptionResNetV2，以及预处理函数换成 InceptionResNetV2 模型对应的函数。

3.2.2.2.2.2 数据读取

用与 Xception 迁移学习中同样的方法取数据时，在 AWS 上的 P3 实例运行，发现训练的速度比在 P2 上提升很少，而且 GPU 的使用率经常有空窗期，有可能是用 ImageDataGenerator 时，每一个 bacth 都需要从磁盘中读文件，IO 的速度比较慢引起。所以改用全量读内存的方式读取数据，而且我打算在 InceptionResNetV2 的迁移学习中，实验不同的冻结层数的情况，读内存的方式，多次实验时不需要再次读入。数据读入内存的过程如下：

- **训练数据读入**

1) 定义 1 个 24970X299X299X3 大小的 4 维 np 数组 X，用于存训练集数据；定义 1 个 24970 大小的 1 维 np 数组 Y，用于存训练集的标签值；初始值均为 0；
2) 用 cv2 库的 imread 方法读图片文件，用 resize 方法将图片缩放到 InceptionResNetV2 模型要求的输入大小规格(299X299)；
3) 从 train2 目录读取，先加载 dog 目录的图片数据到 X，再加载 cat 目录的图片数据到 X。dog 目录图片对应的 Y[i]置 1，cat 目录图片对应的 Y[i]置 0；i 从 0 开始，每读 1 张图片加 1；
4) 由于训练集可以不用关心处理文件的顺序，只需要数据和标签的关系是正确的即可，所以读取文件时并不根据文件名的编号读取，而是以 os.listdir 返回的顺序读取的；
5) 数据读到内存后，再通过 sklearn.utils 库下面的 shuffle 函数，将顺序打乱，否则前面都是 dog 的数据，后面都是 cat 的数据，会让数据分布不均衡，影响训练的效果。
6) 具体可参见附件 helper.py 中的 read_images_to_memory 函数；

- **测试数据读入**

主要过程和训练数据的读入相似，不同之处主要在：

1) 读取图片文件的方法是 keras 库 preprocessing.image 模块下的 load_img 和 img_to_array 函数，用这两个函数跟用 cv2 库的 imread 和 resize 的方法效果是一样的；
2) 测试集的输出对文件的顺序是有要求的，必要按 1 到 12500 的顺序处理，所以读入时是根据文件名来的；
3) 不存在标签值需要处理；数据不需要 shuffle；
4) 具体可参见附件 helper.py 中的 load_test_data 函数；

3.2.2.2.2.3 训练

用 fit 函数完成训练过程，InceptionResNetV2 模型的训练代码如图 24 所示，训练参数说明：

1）训练 20 代，epochs=20；
2）checkpoint 控制只有验证集 loss 下降时才保存模型权重文件；
3）EarlyStopping 控制如果连续 3 个 epoch,loss 都没有没有下降,则停止训练；
4）validation_split=0.2，表示验证集比例为 20%
5）batch_size 控制每个 bacth 的大小；
6）shuffle=True，每轮迭代之前混洗数据；

```
#训练模型并保存在验证集上损失函数最小的权重
checkpoint = ModelCheckpoint(model_h5file_base, monitor='val_loss', verbose=1, save_best_only=True, mode='min',save_weights_only=True)
stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1, mode='min') #如连续3个epoch,loss都没有没有下降,则停止训练
callbacks_list = [stopping,checkpoint]

history=model.fit(X_train, Y_train, batch_size=128, epochs=epochs, validation_split=0.2, shuffle=True, callbacks=callbacks_list)
```

图 24　InceptionResNetV2 模型的训练代码(V5 版)

InceptionResNetV2 的实验总体上分了 5 个版本，如下面表 5 说明：

版本	代码	主要差异说明	采用
V1	keras_fine_tunig_InceptonResNetV2.ipynb	读入数据用 ImageDataGenerator lock_layers 有 BUG	否
V2	keras_fine_tunig_InceptonResNetV2-memory.ipynb	数据全量读入内存 lock_layers 有 BUG	否
V3	keras_fine_tunig_InceptonResNetV2-memory-v3.ipynb	数据全量读入内存 调整部分参数 lock_layers 有 BUG	否
V4	keras_fine_tunig_InceptonResNetV2-memory-v4.ipynb	数据全量读入内存 lock_layers 修复 BUG	是
V5	keras_fine_tunig_InceptonResNetV2-memory-v5.ipynb	数据全量读入内存 调整部分参数 lock_layer 修复 BUG	是

表 5　InceptionResNetV2 模型 fine-tuning 时的实验版本

表 5 补充说明：

1. V4 由 V2 复制而来，V5 由 V3 复制而来：
2. V5 版本相对 V4 版本做的调整内容（也即 V3 版本相对 V2 版本的调整）：
   a) Dense 增加正则项
   b) DropOut 参数改成 0.5
   c) callbacks_list 增加 EarlyStopping，增加 epochs，统一用 epochs=20，利用 EarlyStopping 控制结束时机
   d) 改 predict_on_model，test 数据加内存，一次性完成
3. lock_layers 的 BUG 说明：跑 V2、V3 版本时，发现放开不同层数的训练结果（val_loss）和只训练输出层相比，改进的不大。后来检查发现因为只训练输出层时，已经把前面所有的层锁住了，而 lock_layers 函数里没有重新把所有层放开，只做了按参数锁层的动作，所以导致调整此函数后，其实后面的训练也是只训练了输出层。

3.2.2.2.2.4 预测结果

预测方法同 Xception 迁移学习中的方法，训练和预测结果参考 3.2.1 执行结果章节的表 4 的内容。

从结果可以看出，虽然 V4 的版本比 V5 的版本，对应的实验中 val_loss 更低，但是最终的测试集结果却差别不大，而且分数最好的模型在 V5 中，可见 V5 中增加正则项，增大 dropout 的参数，确实能起到抑制过拟合的作用。

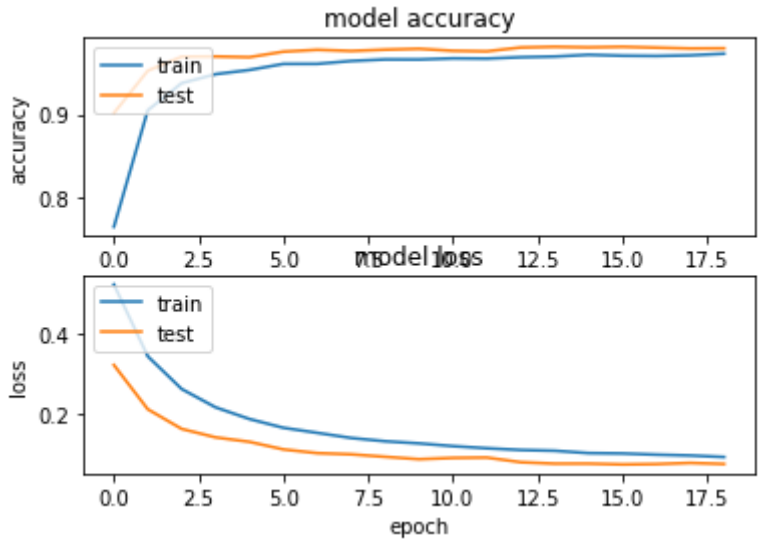

图 25　InceptionResNetV2 模型只放开输出层时的学习曲线(V3)

从图 25 的学习曲线看，整个曲线前面收敛快，后面慢慢收敛，训练集和验证集的准确率 与 loss 值，越来越靠近，是比较合理的一个学习过程。从训练过程（图 26）看，也已经触发了 EarlyStopping。

```
Epoch 00015: val_loss did not improve from 0.07770
Epoch 16/20
19944/19944 [==============================] - 124s 6ms/step - loss: 0.1030 - acc: 0.9705 - val_loss: 0.0758 - val_acc: 0.9811

Epoch 00016: val_loss improved from 0.07770 to 0.07577, saving model to InceptonResNetV2-base-tuning-v3.h5
Epoch 17/20
19944/19944 [==============================] - 124s 6ms/step - loss: 0.1001 - acc: 0.9702 - val_loss: 0.0767 - val_acc: 0.9801

Epoch 00017: val_loss did not improve from 0.07577
Epoch 18/20
19944/19944 [==============================] - 124s 6ms/step - loss: 0.0980 - acc: 0.9709 - val_loss: 0.0795 - val_acc: 0.9791

Epoch 00018: val_loss did not improve from 0.07577
Epoch 19/20
19944/19944 [==============================] - 124s 6ms/step - loss: 0.0942 - acc: 0.9729 - val_loss: 0.0769 - val_acc: 0.9793

Epoch 00019: val_loss did not improve from 0.07577
Epoch 00019: early stopping
```

图 26　InceptionResNetV2 模型只放开输出层时的训练过程最后几代输出

实验的具体结果看表 4 中的统计。

3.2.2.3ResNet50

3.2.2.3.1　模型简介

ResNet50 是深度残差网络（ResNet）中的一种，如图 27 所示，ResNet 还有 18 层、34 层、101 层等结构的模型。ResNet 模型的核心是残差学习，它主要解决的问题是深度神经网络中常见的“梯度消失”[1]的问题。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 [3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 27　ResNet 的模型结构，来自论文[4]

那么 ResNet 是怎么解决这个问题的呢？主要思路是引入残差学习模块，如图 28 所示。假设 H(x)是需要拟合的函数，使 F(x)=H(x)-x，这里 F(x)就是残差，那么 H(x)=F(x)+x，拟合 H(x)过程就可以转换成拟合 F(x)+x 的过程。通常优化残差的函数 F(x)，使其跟 0 拟合，比优化原来函数 H(x)更容易。通过快捷连接（shortcut connections），可以实现 F(x)+x 映射。网络中引入这个 x 后，再求偏导时，总有一个层数是等于 1 的，这样就不会让梯度消失。求偏导过程可参考：https://www.zhihu.com/question/38499534

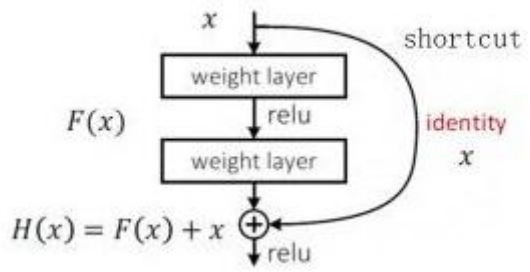

图 28　残差学习基础模块

3.2.2.3.2　迁移学习

整体说明：

使用 keras 的 ResNet50 预训练模型进行 fine-tuning，分别进行下面四种情况的 fine-tuning:

1. 只训练自定义的输出层；
2. 冻结前 164 层，训练后面的层；
3. 冻结前 142 层，训练后面的层；
4. 冻结前 112 层，训练后面的层；

执行过程和 InceptionResNetV2 的模型的迁移学习基本一致，代码由 InceptonResNetV2 模型的 V5 版本（keras_fine_tunig_InceptonResNetV2-memory-v5.ipynb）复制而来，只调整了模型、预处理函数、冻结层数等相关配置，只跑了一个版本。

3.2.2.3.2.1 模型搭建

模型的搭建过程和结构，与 Xception 迁移学习的基本一样[见 3.2.2.1.2.1 章节]，不同之处只是中间的预训练模型换成了 ResNet50，以及预处理函数换成 ResNet50 模型对应的函数。

3.2.2.3.2.2 数据读取

用与 InceptionResNetV2 迁移学习中同样的方法，用 read_images_to_memory 加载训练集，用 load_test_data 加载测试集，全量读入内存，只是图片大小参数改为 224, 224。

3.2.2.3.2.3 训练

用 fit 函数完成训练过程，参数和代码与 InceptionResNetV2 模型的训练类似。

3.2.2.3.2.4 预测结果

预测方法同 Xception 迁移学习中的方法，训练和预测结果参考 3.2.1 执行结果章节的表 4 的内容。

只放开输出层训练时的学习曲线如图 29：

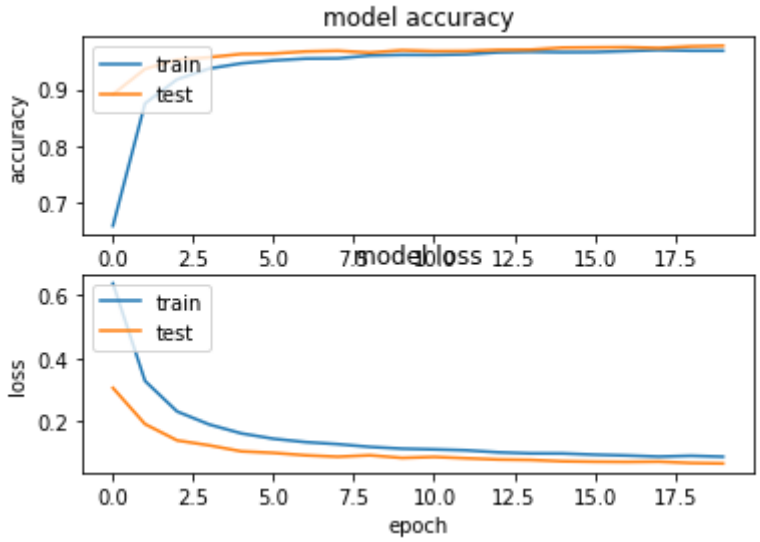

图 29　ResNet50 模型只放开输出层时的学习曲线

从图 29 的学习曲线看，整个曲线前面收敛快，后面慢慢收敛，训练集和验证集的准确率 与 loss 值，越来越靠近，是比较合理的一个学习过程。但从训练过程输出看（图 30），最后两代 val_loss 还在下降，所以可以偿试把 epochs 设置的更大一些。

```
Epoch 00016: val_loss improved from 0.07222 to 0.07083, saving model to ResNet50-base-tuning-v1.h5
Epoch 17/20
19944/19944 [==============================] - 47s 2ms/step - loss: 0.0905 - acc: 0.9671 - val_loss: 0.0701 - val_acc: 0.9739

Epoch 00017: val_loss improved from 0.07083 to 0.07012, saving model to ResNet50-base-tuning-v1.h5
Epoch 18/20
19944/19944 [==============================] - 47s 2ms/step - loss: 0.0870 - acc: 0.9688 - val_loss: 0.0710 - val_acc: 0.9725

Epoch 00018: val_loss did not improve from 0.07012
Epoch 19/20
19944/19944 [==============================] - 47s 2ms/step - loss: 0.0896 - acc: 0.9677 - val_loss: 0.0667 - val_acc: 0.9753

Epoch 00019: val_loss improved from 0.07012 to 0.06670, saving model to ResNet50-base-tuning-v1.h5
Epoch 20/20
19944/19944 [==============================] - 47s 2ms/step - loss: 0.0867 - acc: 0.9677 - val_loss: 0.0653 - val_acc: 0.9763

Epoch 00020: val_loss improved from 0.06670 to 0.06526, saving model to ResNet50-base-tuning-v1.h5
```

图 30　ResNet50 模型只放开输出层时的训练过程最后几代输出

实验的具体结果看表 4 中的统计。

3.2.3 融合模型

融合模型的执行过程参考了培神的知乎文章《手把手教你如何在 Kaggle 猫狗大战冲到 Top2%》

3.2.3.1模型介绍

使用 keras 的 ResNet50、Xception、InceptionResNetV2 三个预训练模型，分别预测输出各自模型的特征向量，然后再将三个特征向量合成一条特征向量，输入到 1 个简单的二分类模型，模型结构如图 31 所示，总体过程说明如下：

1) 分别导出本项目数据（train2 和 test 目录下的图片）在以上三个预训练模型中（不含输出层）预测输出的特征向量；
2) 将三个模型输出的特征向量合成一条特征向量；
3) 搭建简单模型：1 个 dropout+1 个全连接层，其中全连接层激活函数为 sigmod
4) 根据合并好的 tran2 特征向量训练搭建的简单模型；
5) 根据合并好的 test 特征向量预测测试集；

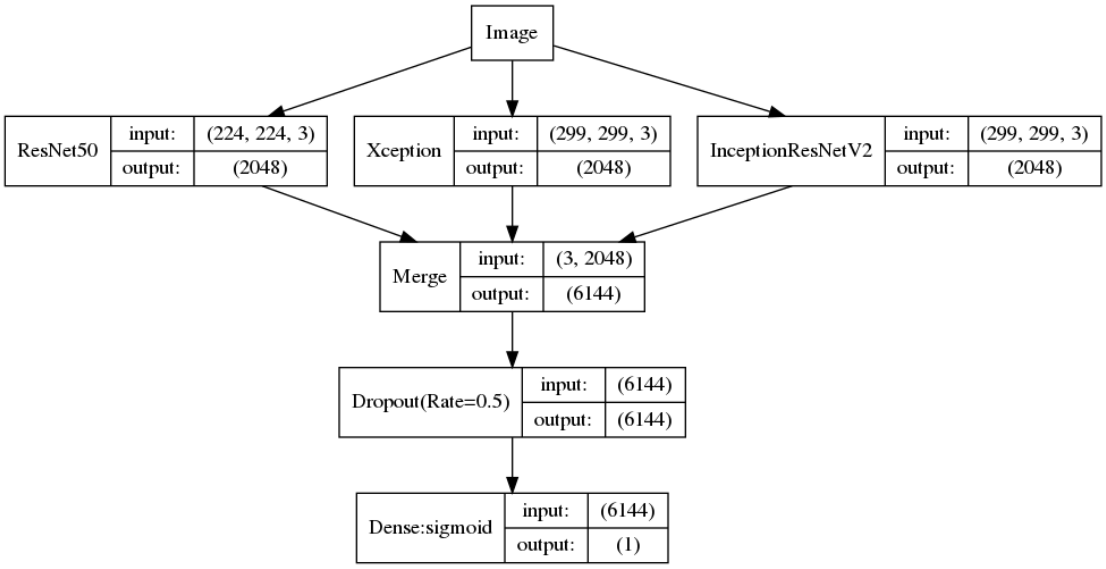

图 31　融合模型结构图

3.2.3.2特征提取

定义提取特征的函数 write_feature_data(MODEL, image_shape, weights_file, preprocess_input = None)，参数说明：

1) MODEL：用于提取特征的模型；
2) image_shape，提取特征的模型对应的输入图片的标准大小；
3) weights_file，取值有：None, imagenet 或权重文件路径；
4) preprocess_input，提取特征的模型对应的预处理函数；

过程说明：

1) 加载模型权重文件,根据 weights_file 参数加载；
2) 读取数据，过程类似 Xception 实验时的处理，使用 keras 库提供的 ImageDataGenerator 生成器读取训练集数据和测试集数据，但没有使用数据增加的功能；
3) 预测生成特征，使用 predict_generator 得到分别得到训练集和测试集的特征向量；
4) 保存特征向量，使用 h5py 库将，特征向量保存在 h5 文件中。
5) 具体参见附件 keras_merge_3_app.ipynb 中的 write_feature_data 函数

此项目中将做两类实验的特征提取，一是使用根据 imagenet 预训练的模型提取特征，一是根据此前迁移学习中保存的训练模型提取特征。

3.2.3.3特征融合

使用 numpy 数组，将三个模型中提取的特征向量，合成一条特征向量。

3.2.3.4训练

用 fit 函数完成训练过程，参数和代码均与 InceptionResNetV2 模型的训练一样，分别尝试了‘adadelta’和‘Adam’两种优化算法。

3.2.3.5预测结果

预测方法同 Xception 迁移学习中的方法略有差别，因为生成特征向量时是根据 ImageDataGenerator 生成器读取数据的，文件读入的顺序并不是按文件编号从小到大排序的，所以需要对预测结果需要根据文件名编号重新排序。

训练和预测结果参考 3.2.1 执行结果 章节的表 4 融合模型部分的内容。从表中可以看出，使用 Adam 做优化算法时的，根据迁移学习的权重文件提取的特征向量训练出来的模型预测的结果得到的 kaggle 得分是 0.03623，是在本次所做的所有实验中得分最好的。

从图 32，图 33 的学习曲线看，使用 Adam 优化算法，比使用 adadelta 优化算法收敛的过程更合理（这可能跟用 Adam 时，学习率设的比较小有关）。

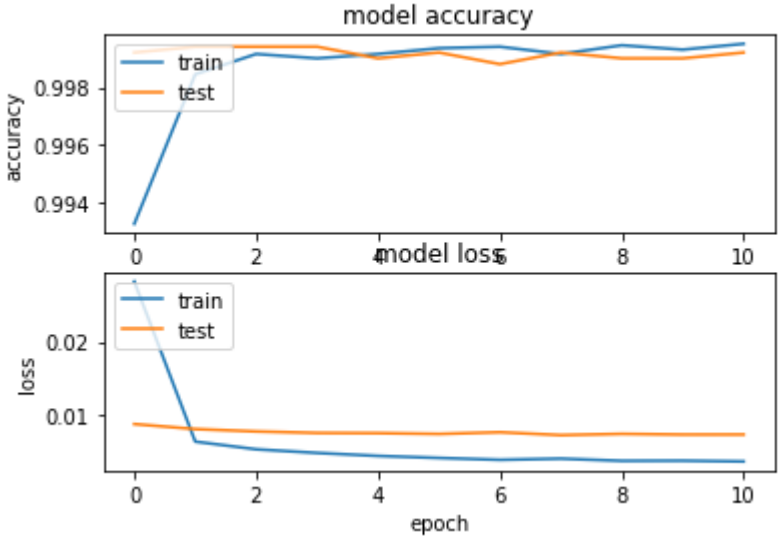

图 32　使用 adadelta 优化算法的学习曲线

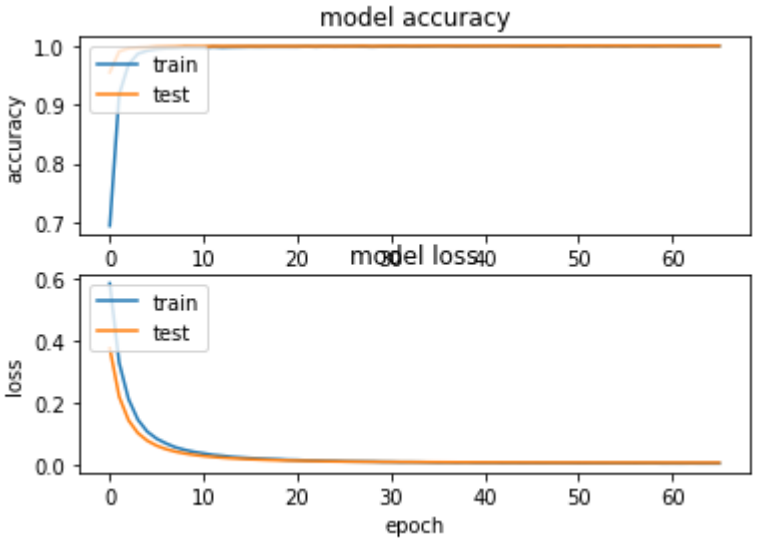

图 33　使用 Adam 优化算法的学习曲线

4 结果

4.1模型的评价与验证

经过不同实验的尝试，所有单模型在放开部分层 fine-tuning 的情况下，都超过了基准模型（提交 kaggle 后，LogLoss 都低于 0.06127，具体结果见表 4），而融合模型则取得了更好的分数，特别是使用 fine-tuing 后的权重文件做融合时，可以说它们很好的提取了图片中的特征。

我将在异常图片检测时选出的部分图片做再次验证，以确认模型的鲁棒性。选择鲁棒性验证图片的规则如下：

1）异常图片检测时 top50-top100 选出的差异图片；之所有没直接用 top50 或 top100 选出的图片，是因为两者之差可能更具代表性。因为直接用 top100，大部分图片都是非猫非狗，或很难辨认的图片，太过极端了；

2）没放入训练集的图片，即 diff/dog 中所有图片和 diff/cat 下的 6 张图片；拿训练集时，模型没见过的图片预测，更有说服力；

3）模型使用：实验中的最优模型 Merge-tuning-2-v2.h5。

图 34　鲁棒性验证结果

预测结果如图 34 所示，我将预测值在 0.4 到 0.6 之间的图片结果置为不确定 unsure，大于 0.6 为狗，小于 0.4 为猫（展示值为：1-预测值）。从图 35 可以看出，总共 15 张图片，结果为：

1) 有 5 张是不确定的；
2) dog.1773.jpg 按理应该也是不确定，但结果是 91%是狗；
3) dog.6475.jpg 肉眼看不出哪里有狗，但结果是 85.67%是狗；
4) 其他 8 张图片都预测正确了；

虽然有部分图片仍然是预测不准确的，但相比于原来 InceptionResNetV2、InceptionV3、Xception 三个模型（之一或以上）的 top50 都没准确预测出的图片，能多预测准确一半以上，可见模型有较好的鲁棒性。

4.2合理性分析

从前面的执行结果、鲁棒性验证看，实验得到的最优模型（或者最优特征提取过程），达到了项目基准模型的要求。它在三个成熟的模型 ResNet50、Xception、InceptionResNetV2 的基础上演变而来，在提取特征时进一步使用项目的数据集优化了预训练的模型权重，而最终的结果得分，也体现了这种优化确实起到了作用。

5 项目结论

5.1结果可视化

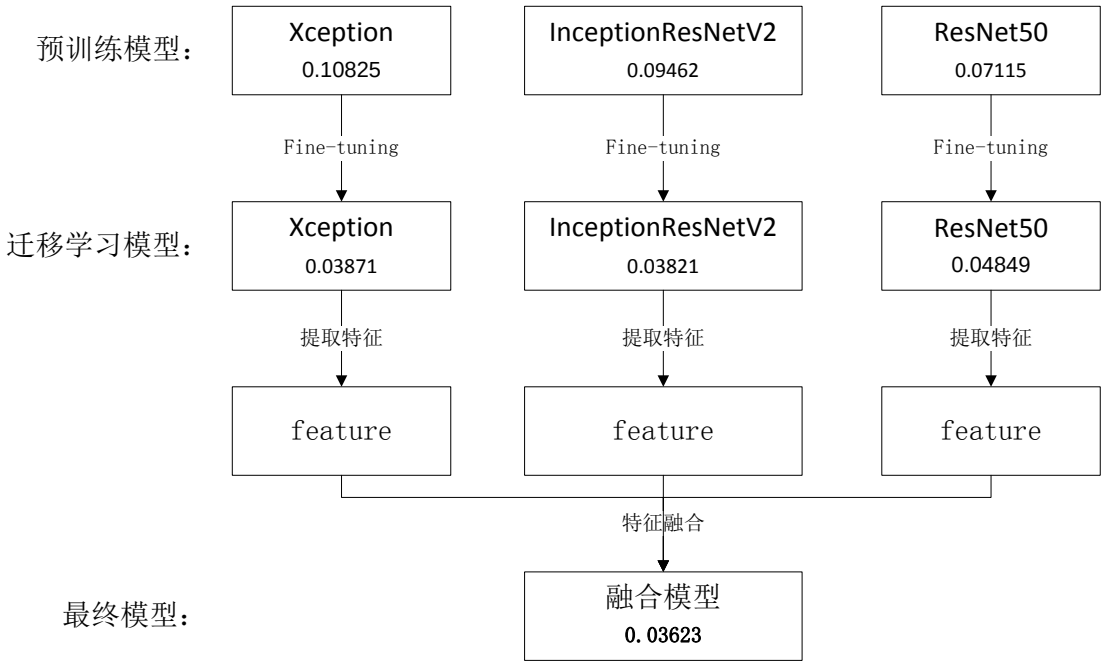

图 35　项目主要阶段及各阶段相应得分

图 35 展示了整个项目过程的各个主要阶段，及各个阶段中模型所取得得分，得分值可以跟表 4 中对应起来看，说明：

1) 预训练模型是指只重构输出层时，训练的模型 kaggle 得分；
2) 迁移学习模型的得分是各个模型经过不同实验的迁移学习后的最优得分；
3) 从这个图的层次上可以看出，这是个递进优化的过程；

5.2对项目的思考

本项目主体思路是选取 keras 库中部分已有知名的卷积神经网络模型，进行迁移学习，以应用到项目中，达到项目基准模型的要求。为了取得最优的结果，通过放开不同的层数，选择不同的优化算法等方式，在三个不同的卷积模型，分别进行了多次 fine-tuning。最终，在这些模型上的 fine-tuning 结果，都取得了不错的成绩，各自的最佳成绩的都达到了项目的要求。

为了进一步提升成绩，尝试了融合模型的方法，通过将三个模型提取的特征向量做融合，再通过简单的二分类模型训练。三个模型都是在 ImageNet 上训练出来的，在输出层前提取的特征向量，可以说包含了一张图片中的主要抽象特征。将三个模型导出的向量融合在一起，又起到了互相补充的作用。而通过 fine-tuning 后的模型来提取特征，则进一步强化了猫狗的特征。

项目执行过程中对单模型的迁移学习花了不少时间，在 aws，p2 机器上，每一代差不多都要 5-10 分钟左右，在 p3 机器上则要 1-2 分钟左右。在尝试放开不同层数实验的时候，我又偏偏不明智的先调了 InceptionResNetV2 模型（所选模型中深度最深的一个模型，每代

训练时间比 ResNet50 多 1 倍以上），程序发现有 BUG 时，来回多跑了好几次。这里总结几个走了弯路的经验：

1) lock_layers 的 BUG，前文已有提及，不易查觉，因为不会报错；
2) 跑单个模型 fine-tuing 时没有保存不含输出层的权重文件，后面融合前，需要再次生成不含输出层的权重文件；
3) ImageDataGenerator 读取文件的顺序不是文件编码顺序，所以对预测结果需要重新调整输出顺序，才能提交 kaggle；
4) 使用 EarlyStopping，就不用因为 epochs 的问题重复训练了（前提是 epochs 够大，本项目中就有 epochs 设的不大，没触发 EarlyStopping 就结束的情况）；

5.3 需要作出的改进

本项目最终的得分虽然达到了项目的要求，但执行过程仍然有很多可以改进的地方，鉴于项目时间的要求等原因，没有做更细致的训练。以下几点，我觉得是可以做出改造的：

1) Xception 模型的迁移学习，可以再试几个放开不同层数的训练；
2) Fine-tunig 可以尝试更多参数组合，但这需要花更多的训练时间；
3) 有些训练过程，loss 还在下降，就到达 epochs 的设置值，可以将 epochs 设置的更大一点；
4) 肯定还有其他需要改进的地方；

6 附件列表

附件类别	文件名	文件说明
模型图	model_Xception.png	迁移学习中，在 Xception 模型基础上搭建的模型图
	model_InceptionResNetV2.png	迁移学习中，在 InceptionResNetV2 模型基础上搭建的模型图
	model_ResNet50.png	迁移学习中，在 ResNet50 模型基础上搭建的模型图
代码	helper.py	辅助函数
	visuals.ipynb	探索性可视化的代码
	pre_dir_process.ipynb	建文件目录的代码
	pick_up_anormal_images_batch_dog. ipynb	挑选训练集 dog 目录中异常图片的代码
	pick_up_anormal_images_batch_cat.ipynb	挑选训练集 cat 目录中异常图片的代码
	pick_up_anormal_images.ipynb	整合上面两份代码（pick dog 和 pick cat）的功能到一个文件中处理
	prep_data_process.ipynb	数据预处理的代码
	keras_fine_tuning_Xception.ipynb	Xception 迁移学习的代码
	keras_fine_tunig_InceptonResNetV2.ipynb	InceptonResNetV2 迁移学习 v1
	keras_fine_tunig_InceptonResNetV2-memory-v2.ipynb	InceptonResNetV2 迁移学习 v2
	keras_fine_tunig_InceptonResNetV2-memory-v3.ipynb	InceptonResNetV2 迁移学习 v3
	keras_fine_tunig_InceptonResNetV2-memory-v4.ipynb	InceptonResNetV2 迁移学习 v4
	keras_fine_tunig_InceptonResNetV2-memory-v5.ipynb	InceptonResNetV2 迁移学习 v5
	keras_fine_tunig_ResNet50-v1.ipynb	ResNet50 迁移学习的代码
	keras_merge_3_app.ipynb	融合模型，加载 keras 预训练权重
	keras_merge_3_app_by_tuning.ipynb	融合模型，加载迁移学习保存的预训练权重
	keras_merge_3_app_predict.ipynb	鲁棒性验证代码
预测结果	pred-xception-freeze-2.csv	Xception 放开输出层的预测结果
	pred-xception-fine-tuning-1.csv	Xception 冻结前 97 层的预测结果
	pred-InceptonResNetV2-base-tuning-v4.csv	InceptonResNetV2 v4 只训练输出层
	pred-InceptonResNetV2-fine-tuning-1-v4.csv	InceptonResNetV2 v4 冻结前 698 层
	pred-InceptonResNetV2-fine-tuning-2-v4.csv	InceptonResNetV2 v4 冻结前 618 层
	pred-InceptonResNetV2-fine-tuning-3-v4.csv	InceptonResNetV2 v4 冻结前 499 层
	pred-InceptonResNetV2-base-tuning-v3.csv	InceptonResNetV2 v5 只训练输出层
	pred-InceptonResNetV2-fine-tuning-1-v5.csv	InceptonResNetV2 v5 冻结前 698 层
	pred-InceptonResNetV2-fine-tuning-2-v5.csv	InceptonResNetV2 v5 冻结前 618 层
	pred-InceptonResNetV2-fine-tuning-3-v5.csv	InceptonResNetV2 v5 冻结前 746 层

	pred-ResNet50-base-tuning-v1.csv	ResNet50 只训练输出层
	pred-ResNet50-fine-tuning-1-v1.csv	ResNet50 冻结前 164 层
	pred-ResNet50-fine-tuning-2-v1.csv	ResNet50 冻结前 142 层
	pred-ResNet50-fine-tuning-3-v1.csv	ResNet50 冻结前 112 层
	pred-Merge-tuning-v1.csv	ImageNet 模型融合，算法：adadelta
	pred-Merge-tuning-2-v1.csv	ImageNet 模型融合，算法：Adam
	pred-Merge-tuning-v2.csv	fine-tuning 模型融合，算法：adadelta
	pred-Merge-tuning-2-v2.csv	fine-tuning 模型融合，算法：Adam

7 参考文献

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS , 2012

[2] K.Simonyan, and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556v6, 2015

[3] C.Szegedy, V.Vanhoucke, S.Ioffe, J.hlens, and Z.Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567v3, 2015

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385v1, 2015

[5] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357v3, 2017

[6] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv:1602.07261v2 ,2016

[7] 周志华. 《机器学习》. 清华大华出版社, 2016

[8] cs231n 卷积神经网络：http://cs231n.github.io/convolutional-networks/#overview

[9] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. arXiv:1312.4400v3, 2014.

[10] Global average pooling. http://blog.leanote.com/post/sunalbert/Global-average-pooling