猫狗大战

李华
2018 年 5 月 16 日

目录

一、定义

1. 项目概述

Kaggle 上 13 年的竞赛项目猫与狗。数据集 train.zip 有 2.5 万张猫狗图片，test.zip 有 1.25 万张图片。

项目要求识别测试图片是猫还是狗，是二分类问题，属于计算机视觉领域图像分类问题。适合使用卷积神经网络来构建模型。

将训练数据分为猫和狗，然后使用 Xception 预训练模型进行异常值检测，剔除不是猫和狗的图片。剔除图片的数据进行特征提取，训练，调参等操作，最终找出最优模型，将测试集应用于模型进行预测，将预测结果按照提交文件格式提交到 kaggle，查看得分。

2. 问题说明

属于监督学习，通过训练已有的猫狗图片标签，识别未知标签的猫狗图片。

ImageNet 图像分类，需要从超过 100 万张图片的训练数据，训练的模型用于识别 1000 种图像分类。因为猫狗大战属于 ImageNet 的子问题，如果使用在 ImageNet 中预训练过的模型，进行迁移学习再次训练并学习对这些特征向量进行分类，识别率会更好。

3. 指标

猫狗大战二分类问题使用如下指标，二分类模型一般使用 Sigmoid 激活函数，损失函数为对数损失函数：

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

其中：

- n 测试集数量
- $\hat{y}_i$ 是被预测图像是狗的概率
- yi 图像为狗是 1，猫是 0
- log() 为自然对数

模型应用于测试数据的分类结果，将结果弄成提交样本的格式上传 Kaggle 进行评价，排名。Kaggle 应用对数损失函数对测试结果评价指标。log loss 越小越好。

二、分析

1. 数据研究

项目中的数据集，来自 kaggle 中的 data，包括三个文件 sample_submission.csv，test.zip，train.zip。大小分别为 111.23KB，271.3MB，543.52MB，可以点击 Download 或者在终端使用命令 kaggle competitions download -c dogs-vs-cats-redux-kernels-edition 下载。

其 train.zip 训练数据集包括 2.5 万张图片，并在文件名中标注了图片为猫还是狗，一半的猫，一半的狗，测试数据集包括 1.25 万张图片，没有分类别，文件以数字进行命名。一些图片示例如下：

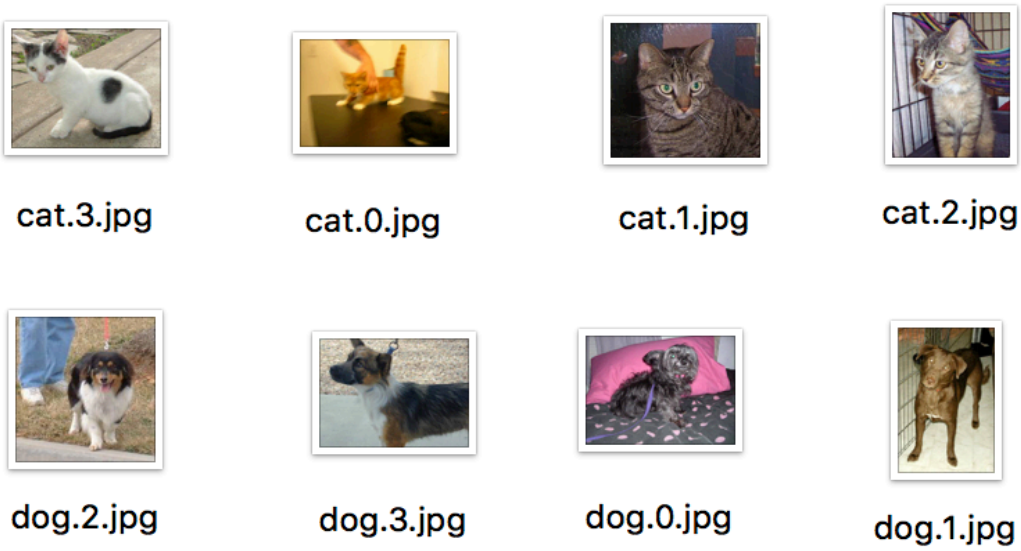


图 1. 训练数据

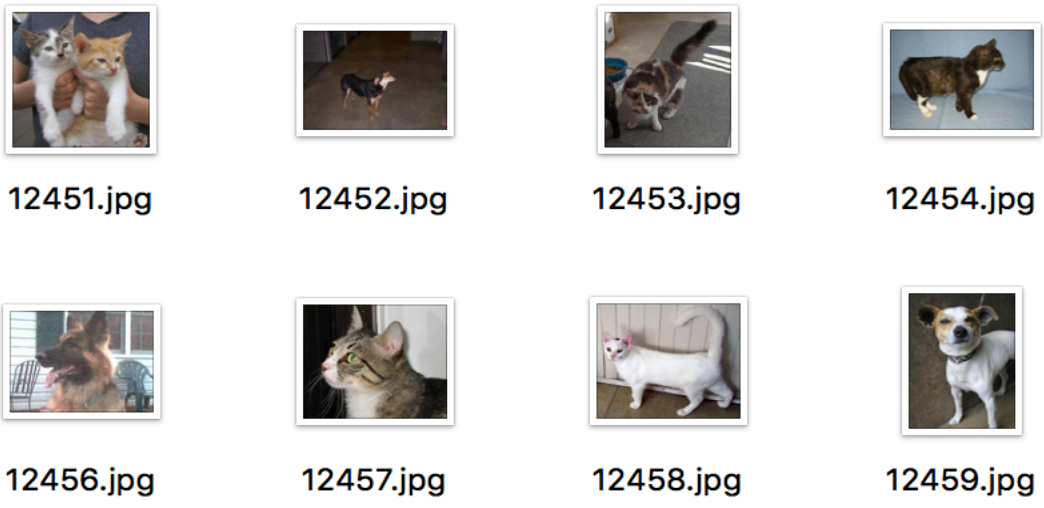
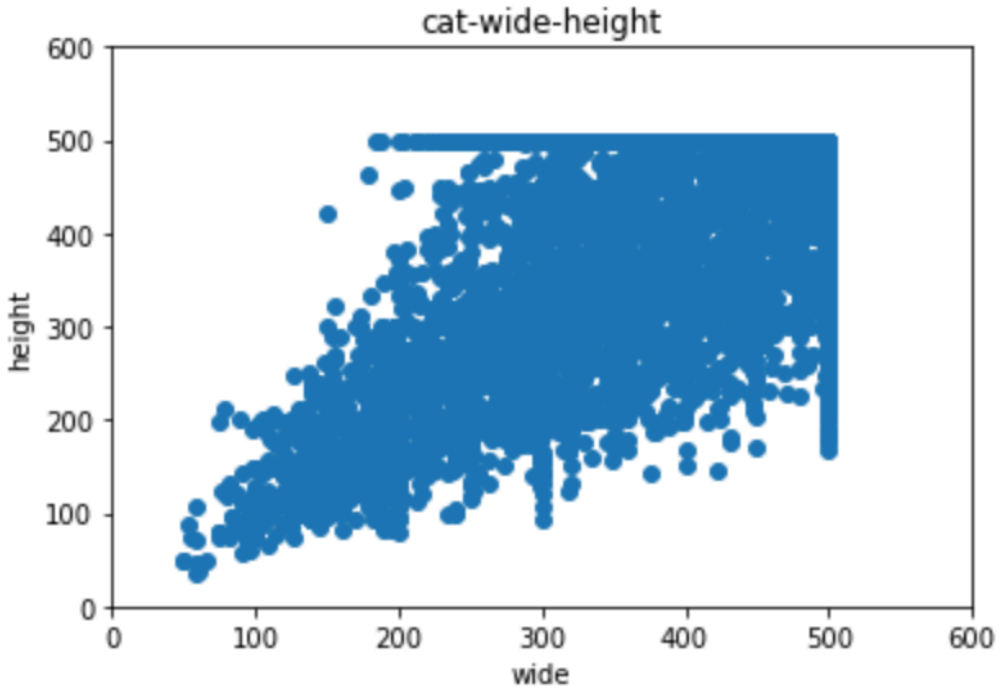
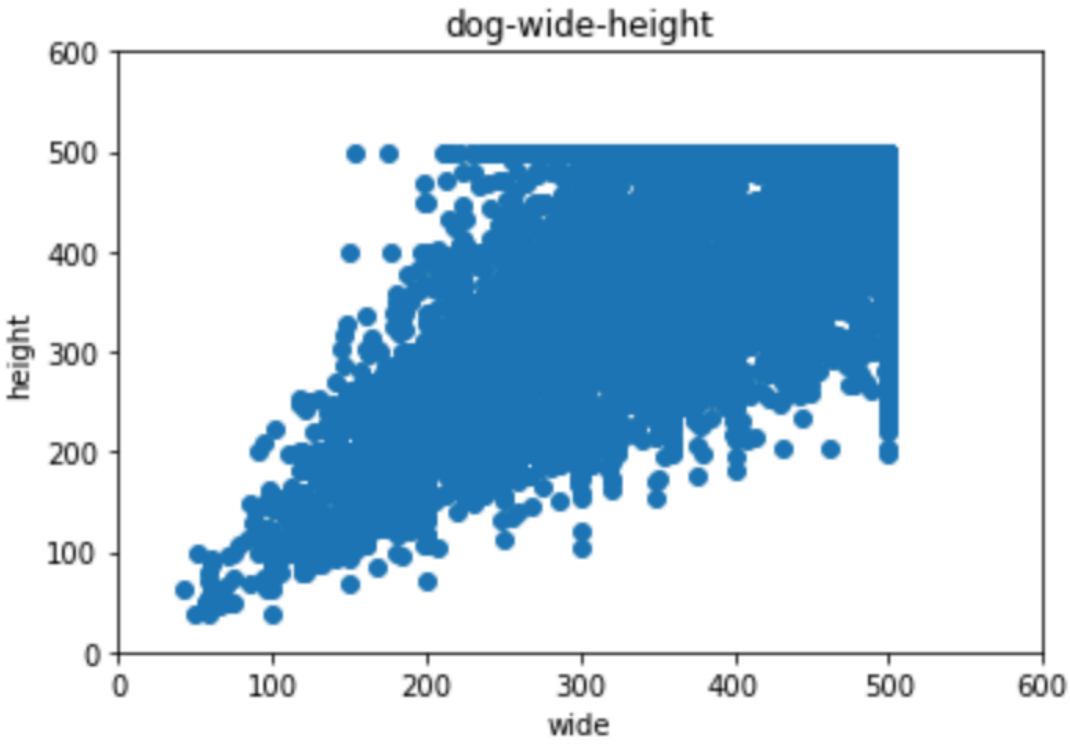


图 2. 测试数据

如上图所示，图 1 为 train.zip 训练集中的 4 张 cat 和 4 张 dog，图 2 为测试集

test.zip 中的 8 张未标记 cat/dog 的图片，从图中可以看出数据中图片尺寸大小不一，图片之间进行比较，有的图片是整只猫或整条狗，有些则只有猫和狗的一部分，有些图片上面一条猫或者狗，有些多条，数量不等，猫和狗的背景也大不同，有些背景比猫狗自身大，背景的复杂程度不同等。可以从下面散点图中看出图片尺寸情况（代码在 data_explore1.ipynb 中）：





2. 探索性可视化

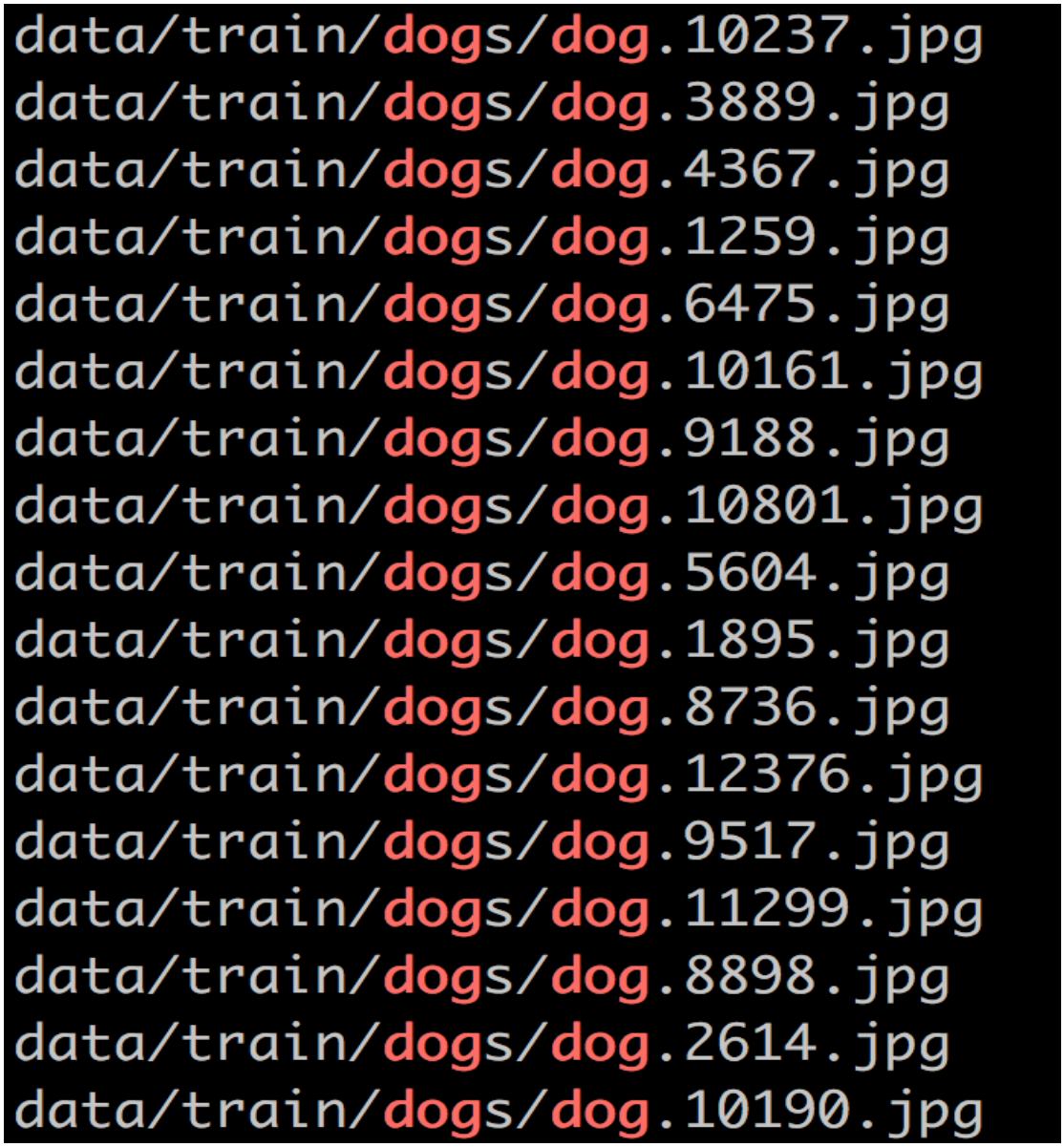
使用 ImageNet 上预训练的 Xception 进行数据探索，直接将预训练好的模型应用于训练集，ImageNet 有 2 中检测准确率的标准，top1 和 top5，top5 为预测一张图片，如果分类结果中概率前五中包含正确答案，则正确，top1 为预测一张图片，若分类结果中概率最大的分类是正确答案，则正确。探索数据是为了找出异常值，提高数据质量。

top 值越大，命中猫和狗的机率越大。我这里首先 top 值设置为 50，分别预测了训练集中猫和狗的图片。这里使用了如下链接中猫和狗的分类带号：https://blog.csdn.net/zhangjunbob/article/details/53258524，从此链接中找出所有是猫的代号和所有是狗的代号，如下：

```
dogs = [
 'n02085620','n02085782','n02085936','n02086079'
,'n02086240','n02086646','n02086910','n02087046'
,'n02087394','n02088094','n02088238','n02088364'
,'n02088466','n02088632','n02089078','n02089867'
,'n02089973','n02090379','n02090622','n02090721'
,'n02091032','n02091134','n02091244','n02091467'
,'n02091635','n02091831','n02092002','n02092339'
,'n02093256','n02093428','n02093647','n02093754'
,'n02093859','n02093991','n02094114','n02094258'
,'n02094433','n02095314','n02095570','n02095889'
,'n02096051','n02096177','n02096294','n02096437'
,'n02096585','n02097047','n02097130','n02097209'
,'n02097298','n02097474','n02097658','n02098105'
,'n02098286','n02098413','n02099267','n02099429'
,'n02099601','n02099712','n02099849','n02100236'
,'n02100583','n02100735','n02100877','n02101006'
,'n02101388','n02101556','n02102040','n02102177'
,'n02102318','n02102480','n02102973','n02104029'
,'n02104365','n02105056','n02105162','n02105251'
,'n02105412','n02105505','n02105641','n02105855'
,'n02106030','n02106166','n02106382','n02106550'
,'n02106662','n02107142','n02107312','n02107574'
,'n02107683','n02107908','n02108000','n02108089'
,'n02108422','n02108551','n02108915','n02109047'
,'n02109525','n02109961','n02110063','n02110185'
,'n02110341','n02110627','n02110806','n02110958'
,'n02111129','n02111277','n02111500','n02111889'
,'n02112018','n02112137','n02112350','n02112706'
,'n02113023','n02113186','n02113624','n02113712'
,'n02113799','n02113978']
```

```
cats=[
'n02123045','n02123159','n02123394','n02123597'
,'n02124075','n02125311','n02127052']
```

对于狗训练集运行代码 python Xception_pred.py > Xception_pred.log &，此代码将预测的前 50 概率类别中没有狗的图片路径打印出来。运行完 grep 过滤出来 dog 的路径，这些路径即为 Xception 预测出的前 50 概率类别中不包含狗的，使用的图像 size 为（299， 299）：



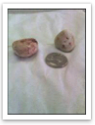
通过观察，有些不同，找到这些图片发现基本都不是狗，或者背景太复杂，导致狗在图片中微不足道。应该去掉这些异常者，否则影响模型学习结果。通过观察将这些图片剔除如下：

dog.1259.jpg


dog.1773.jpg

dog.1895.jpg


dog.2422.jpg


dog.2614.jpg


dog.3341.jpg


dog.3889.jpg

dog.4367.jpg


dog.4565.jpg

dog.5604.jpg


dog.6405.jpg


dog.6475.jpg


dog.6725.jpg


dog.7772.jpg

dog.8736.jpg

dog.8898.jpg


dog.9188.jpg


dog.9517.jpg


dog.10123.jpg

dog.10161.jpg


dog.10190.jpg

dog.10237.jpg


dog.10551.jpg


dog.10801.jpg

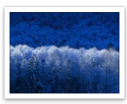
dog.11299.jpg


dog.12148.jpg


dog.12376.jpg

同样，对于猫我是首先做了 top50，发现出来的不是猫的图片总共有 99 个，这个数量有些大，所以继续扩大 top 到 top100，有 48 个。

手动将这 48 个图片移除。如部分个图片：


cat.5418.jpg


cat.6138.jpg


cat.6402.jpg


cat.6429.jpg


cat.6442.jpg


cat.6817.jpg


cat.7300.jpg

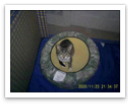

cat.7377.jpg

cat.7564.jpg


cat.7599.jpg


cat.7897.jpg

cat.7968.jpg


cat.8383.jpg

cat.8456.jpg


cat.9110.jpg


cat.9171.jpg

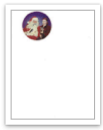

cat.9391.jpg


cat.9456.jpg


cat.9494.jpg


cat.9520.jpg


cat.9695.jpg


cat.10029.jpg


cat.10209.jpg


cat.10266.jpg


cat.10365.jpg

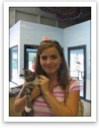

cat.10536.jpg


cat.10636.jpg


cat.10700.jpg

cat.10712.jpg


cat.11184.jpg


cat.12203.jpg


cat.12272.jpg


cat.12424.jpg

将训练数据按照 8:2 分成训练集和验证集。猫有 9960 张训练集，狗有 9980 张。猫验证集有 2492 张，狗验证集有 2493 张。

图像数据本身是 0-255 的 UNIT 型数据所以需要归一化。图像进行了归一化，后端使用的是 tensorflow，从源码可以看出，首先除以 127.5，然后减去 1，归

一化之后所有的像素值都在[-1,1]区间中。

3. 算法与方法

2.1 深度学习

深度学习是机器学习中一种。通过设置输入层，隐藏层，输出层对数据进行训练，传统 ml 复杂的特征工程，先深入探索数据，再进行降维处理，将最好功能的特征传递给 ml。深度学习只需将数据输入网络，经过隐藏层训练，然后从输入层输出就可以得到比较好的结果。深度学习消除了大量特征工程。

图像识别中将在 imagenet 图像库上训练过的模型运用于猫狗大战进行特征提取，减轻模型训练，能够更短时间获取更多性能。

2.2 卷积神经网络

卷积神经网络 cnn 是受到视觉皮层启发，由杨立昆最早应用于手写识别，由卷积层和池化层组成。现在已运用于计算机视觉、语音、NLP 等多方面都有了很大的进步。

卷积神经网络由一个特征提取层和特征映射层组成，特征提取层的神经元输入和其前一层局部相连提取特征，特征映射层使用 sigmoid 作为激活函数，每个卷积层连着局部平均值和二次提取的计算层。

cnn 使用卷积核减少网络参数数量，易训练。图像识别往往图像像素和附近的像素相关性高，cnn 很适用。利用卷积神经网络的局部像素训练特征，更能提高识别率。

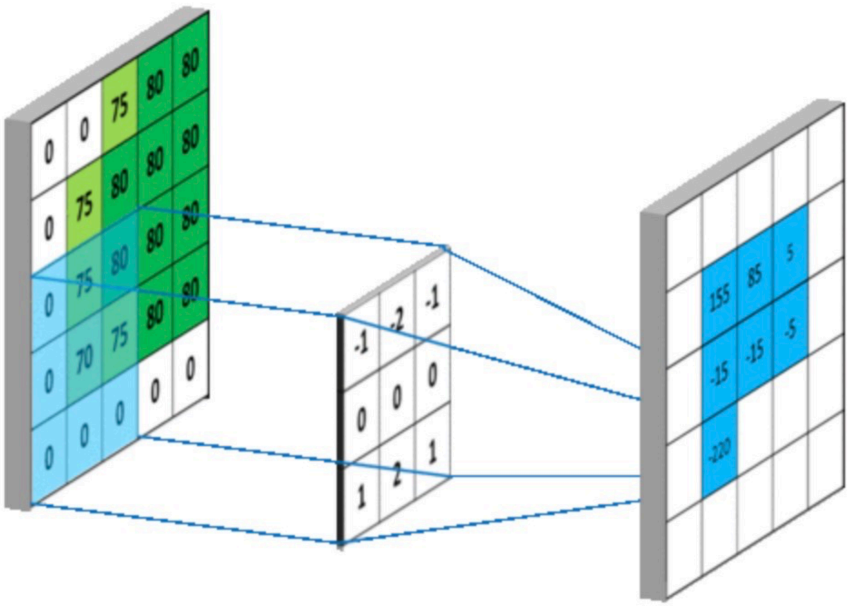


图 3.　CNN 过程

CNN 卷积核共享权值，减少过拟合，均值池化和最大值池化简化了模型的复杂度，减少模型参数。

CNN 由输入层、n 个卷积层和池化层、全连接层构成。

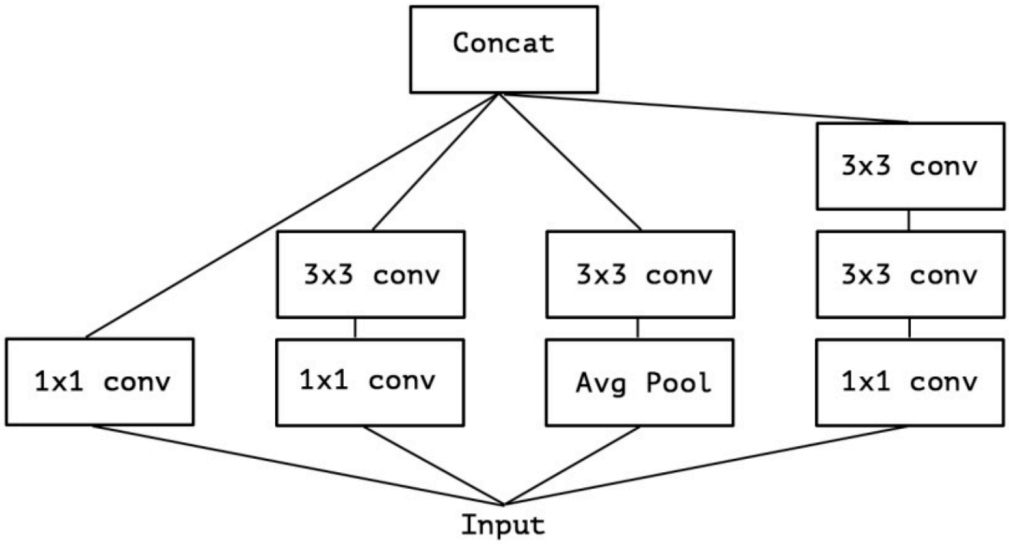
卷积层是提取特征，由于图片某处的特征和其附近的特征关联比较大，所以使用卷积进行特征提取。比如看一张猫的图片，可能看到猫的眼睛或嘴巴就可以确定。比如有一个 5x5 的图像，用一个 3x3 的卷积核对图像进行卷积操作，有一个滑动窗口，把卷积核与对应图像素做乘积然后求和，得到 3x3 的卷积结果。这个过程可以理解为我们使用一个过滤器（卷积核）来过滤图像的每个小部分，从而得到小区域的特征值。实际训练过程中，卷积核的值是在学习过程中学到的。多卷积核中的每个卷积核代表了一种图像模式，如果某个图像块和此卷积核卷积出的值大，则此卷积核和图像块相似度高。

池化有最大值池化和平均值池化。实际应用中即使对图像做完了卷积，图像仍然大（因为卷积核比较小），为了降低数据维度，就进行采样，有效地避免了过拟合。

Dropout 是最近 2 年提出的，为了防止模型过拟合。在每个训练批次中，通过忽略一半的特征检测器（让一半的隐层节点值为 0），能够明显地减少过拟合现象。此方式可以减少特征检测器间的相互依赖。Dropout 以一定的比例随机的将隐层激活值变成 0，这样得到更好的结果。随机选择忽略隐层节点，每个批次的训练过程中，由于每次随机忽略的隐层节点都不同，这样就使每次训练的网络都是不一样的，每次训练都可以单做产生新模型，隐含节点概率随机出现性不能保证每 2 个隐含节点每次都同时出现，这样权值的更新不再依赖于有固定关系隐含节点的共同作用，防止某些特征仅仅在其它特定特征下才有效果的情况。因此 dropout 过程就是一个非常有效的神经网络模型平均方法，通过训练大量的不同的网络来平均预测概率。不同数据集上训练不同模型（每个批次的训练数据都是随机选择），最后在每个模型用相同的权重来融合。

Inception 结构最初在 GoogLeNet 出现，为 Inception V1，接着出现了 Inception V2，Inception V3 和 Inception-ResNet。下图为 Inception V3 的结构：

Figure 1. A canonical Inception module (Inception V3).

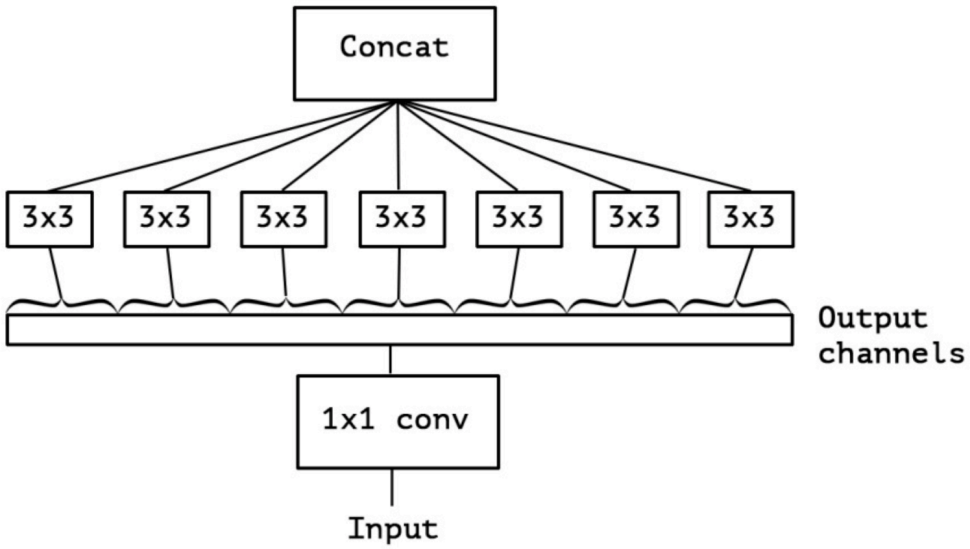


一个卷积层的卷积核有 width、height、channel3 个维度，2 个空间维度（ 宽和高）以及 1 个通道维度，因此一个卷积核需要同时考虑跨通道相关性和空间相关性。Inception 设计思想是要通过分开处理 cross-channel 相关性和 spatial 相关性进行解耦合。Iception 首先通过 1x1 卷积处理跨通道相关性，将输入数据映射到 3 或 4 个小于原始输入的不同空间，然后通过 3x3 或 5x5 卷积再映射到更小的 3 维空间。

如果假设跨通道相关性和空间相关性完全分开处理，会不会比 Inception 假设更合理？

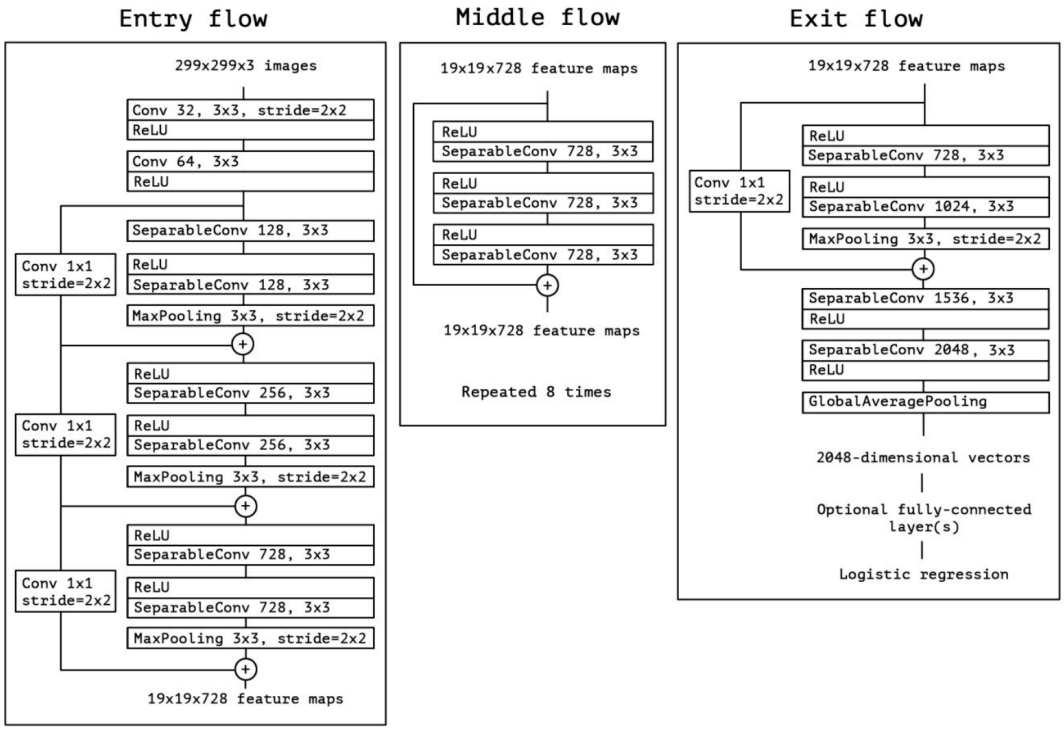
基于上述进行极端的特例，首先使用 1x1 卷积映射跨通道相关性，然后独立映射每个输出通道的空间相关性。

Figure 4. An “extreme” version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



这个极端版本与深度可分卷积几乎相同。depthwise separable 卷积又叫可分卷积，在输入的每个通道独立执行空间卷积，然后进行 1x1 卷积，在新的通道空间得到输出。Inception 极端版本和深度可分卷积的不同：1. 操作顺序（深度可分卷积先进行通道的空间卷积，再进行 1x1 卷积，而 Inception 首先 1x1 卷积）。2. 上步之后是否进行非线性操作（Inception 两个操作后都进行 ReLU 非线性激活，而深度可分卷积不使用）。

假设跨通道相关性和空间相关性可以完全脱离，提出了 Xception，Xception 结构如下：



Xception 基于跨通道相关性和空间相关性的完全解耦合，有 36 个卷积层，被分层 14 个模块，模块之间有线性残差连接。图像分类中，最后一层为逻辑回归，也可以在逻辑回归层之前加上全连接层。

2.3 解决方案

猫狗大战我使用 Python3.6 开发，Keras API 后端使用 tensorflow 进行模型实现与训练。

对 Keras 提供的 ImageNet 数据集预训练过的模型进行迁移学习，通过 Dropout、图像增强等方法解决过拟合。

使用 InageNet 预训练模型 Xception，Dropout 则使每次训练时随机丢掉一部分特征，随机丢弃一些特征，减少过拟合，预测时不会丢弃特征。通过对图像增强提高泛化。

loss 对数函数的特性，预测错误时， log 输入为 0，导致 log 值接近负无穷，分类正确时影响不大，则提交 Kaggle 评估得分时，对输出概率进行截断范围，采用范围[0.005, 0.995]。

3. 基准测试

Xception 将图像空间信息与 channel 信息分开处理，使用 depthwise separable convolution 方法对每个 channel 分开卷积，使用 1×1 卷积核对其进行合并，进一步降低参数数量，提升模型能力。模型目前只能以 TensorFlow 为后端使用, 因为 Xception 依赖于"SeparableConvolution"层,目前该模型只支持 channels_last 的维度顺序(width, height, channels)。

Kaggle 排行榜前 10%名，即在公共排行榜上的 logloss 低于 0.06127。

三、方法

1. 数据预处理

首先对数据图像建成 keras 所需要的目录结构，将原始数据 train.zip 和 test.zip 下载到 data 目录，进行解压，在解压完的 train 目录中创建 cats 和 dogs 目录并将猫和狗数据分别移动到 cats 目录和 dogs 目录，test 目录中创建 test 目录并将测试图像移至其中，可以使用 shell 命令，我这边使用了 python 代码为 layered.py。

数据使用 ImageDataGenerator 生成（299，299）大小的图片，并对图片进行提取特征。

2. 实施

使用 keras 预训练模型 Xception，再将图片数据输入之前，需要将图片尺寸调整为 299*299，然后将训练图片输入模型前进行数据增强，最后使用 Xception 提取特征，进行再训练，我这边训练使用的是 48 核 cpu，256G 内存的服务器进行 cpu 训练的，训练时间部分日志如下：

开头的日志：

```
2018-05-20 12:06:25.518986: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFl
ow binary was not compiled to use: AVX2 FMA
2018-05-20 12:08:34.563256: W tensorflow/core/framework/allocator.cc:101] Allocation of 14208640000 exceeds 10% of system memory.
2018-05-20 12:08:36.281386: W tensorflow/core/framework/allocator.cc:101] Allocation of 14208640000 exceeds 10% of system memory.
2018-05-20 12:08:36.281915: W tensorflow/core/framework/allocator.cc:101] Allocation of 14208640000 exceeds 10% of system memory.
2018-05-20 12:08:37.578910: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:09:59.042927: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:09:59.042958: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:10:03.521464: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:10:03.521465: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
```

结尾的日志：

```
2018-05-20 12:48:03.241909: W tensorflow/core/framework/allocator.cc:101] Allocation of 13829760000 exceeds 10% of system memory.
2018-05-20 12:48:06.758304: W tensorflow/core/framework/allocator.cc:101] Allocation of 13829760000 exceeds 10% of system memory.
2018-05-20 12:48:06.758312: W tensorflow/core/framework/allocator.cc:101] Allocation of 13829760000 exceeds 10% of system memory.
2018-05-20 12:48:08.111654: W tensorflow/core/framework/allocator.cc:101] Allocation of 13829760000 exceeds 10% of system memory.
2018-05-20 12:48:11.444143: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:14.643187: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:14.643187: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:17.879949: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:21.507110: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:24.939405: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:24.939405: W tensorflow/core/framework/allocator.cc:101] Allocation of 27659520000 exceeds 10% of system memory.
2018-05-20 12:48:29.187735: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:48:30.750419: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:48:30.750419: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:48:33.130978: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:48:34.181046: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:48:36.528183: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.
2018-05-20 12:48:36.528183: W tensorflow/core/framework/allocator.cc:101] Allocation of 14018560000 exceeds 10% of system memory.

real    43m45.349s
user    1369m56.533s
sys     421m39.383s
```

执行命令：time python X_feature.py

过程中 cpu 使用和内存使用情况：

```
top - 12:17:14 up 184 days, 15 min,  2 users,  load average: 52.55, 45.33, 32.83
Tasks: 463 total,   4 running, 458 sleeping,   0 stopped,   1 zombie
%Cpu(s):  3.5 us, 96.5 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26347673+total,  3097252 free, 19143838+used, 68941112 buff/cache
KiB Swap: 67108860 total, 67108860 free,        0 used. 68007648 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
10057 user_00   20   0  0.790t 0.173t  46532 S  4207 70.6 368:50.51 python
24767 yarn      20   0 3647836 1.506g  12436 S 299.0  0.6 316:44.22 java
```

这是高峰时期的情况，可以看出 cpu 使用 42 核，内存使用 70%以上。

```
top - 12:14:15 up 184 days, 12 min,  2 users,  load average: 52.04, 41.75, 29.28
Tasks: 460 total,   2 running, 458 sleeping,   0 stopped,   0 zombie
%Cpu(s): 97.0 us,  2.2 sy,  0.0 ni,  0.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26347673+total, 14752612+free, 47025520 used, 68925096 buff/cache
KiB Swap: 67108860 total, 67108860 free,        0 used. 21241830+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
10057 user_00   20   0  0.392t 0.039t  46404 S  4742 15.9 239:36.87 python
 4563 user_00   20   0  157992   2616   1560 R   0.7  0.0   0:05.77 top
```

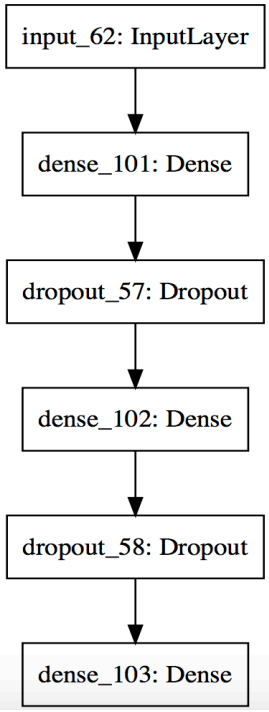
这是正常使用情况，可以看出 cpu 使用达到 46 核，内存使用 15.9%。

利用 GlobalAveragePooling2D 将卷积层输出的每个激活图直接求平均值，降低过拟合风险。然后我们定义了一个 train_datagen 和一个 test_datagen 及 validation_generator，利用 model.predict_generator 函数来导出特征向量，最后选择了 Xception。从上面截图上可以看出时间大概花了 42 分钟。

将生成的 Xception_feature_v1.h5 使用 sz 下载下来，在本机加载 Xception_feature_v1.h5，将图像输入层，然后进行 Dropout，这也是一种正则化手段，它通过随机将部分神经元的输出置零来实现。使用 sigmoid 激活函数进行二分类，模型编译的时候使用 RMSprop 优化器，模型可视化如下：

```
In [343]: from IPython.display import SVG
          from keras.utils.vis_utils import model_to_dot

          SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[343]:



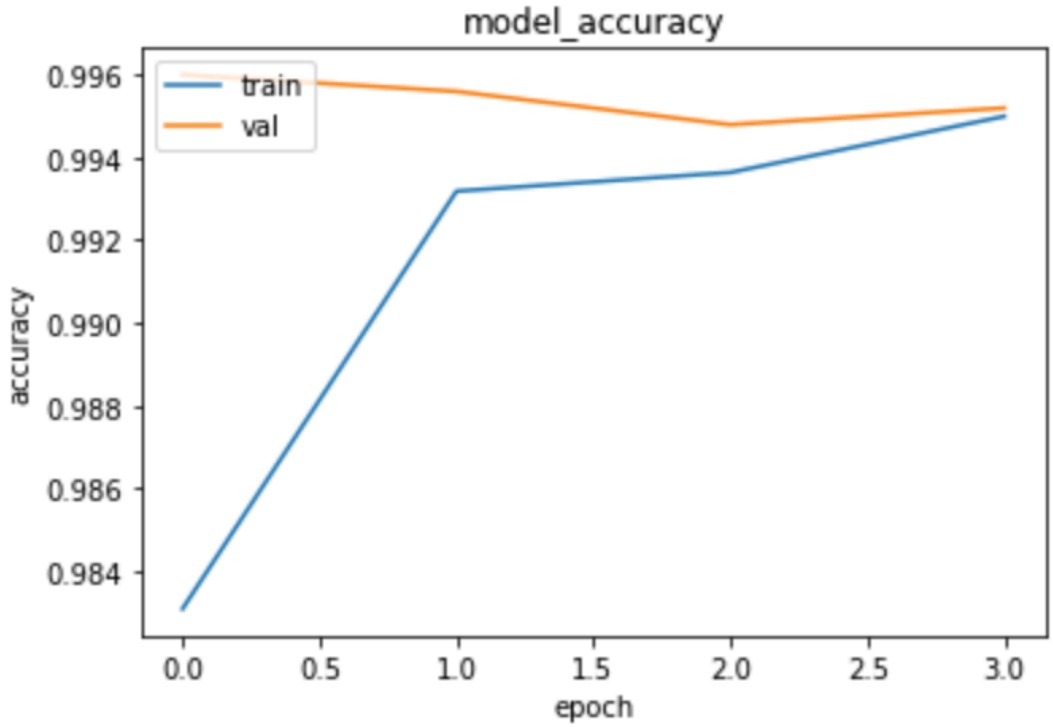
训练过程：

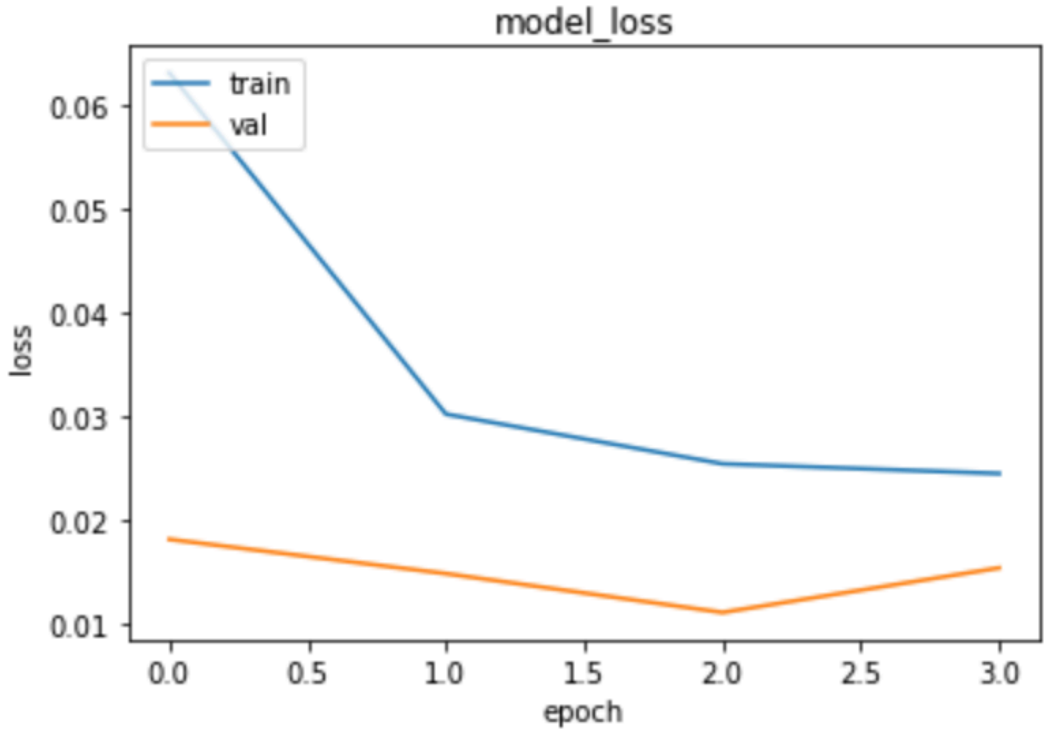
```
In [424]: hist = model.fit(train, train_label,
                    epochs=30,
                    batch_size=128,
                    validation_data=(validation, validation_label),
                    callbacks=[checkpointer, earlystopping])
```

```
19940/19940 [==============================] - 12s 610us/step - loss: 0.0632 - acc: 0.9831 - val_loss: 0.0182 - val_a
cc: 0.9960

Epoch 00001: val_loss improved from inf to 0.01820, saving model to weights.hdf5
Epoch 2/30
19940/19940 [==============================] - 8s 380us/step - loss: 0.0303 - acc: 0.9932 - val_loss: 0.0149 - val_ac
c: 0.9956

Epoch 00002: val_loss improved from 0.01820 to 0.01490, saving model to weights.hdf5
Epoch 3/30
19940/19940 [==============================] - 7s 363us/step - loss: 0.0255 - acc: 0.9936 - val_loss: 0.0112 - val_ac
c: 0.9948

Epoch 00003: val_loss improved from 0.01490 to 0.01115, saving model to weights.hdf5
Epoch 4/30
19940/19940 [==============================] - 7s 330us/step - loss: 0.0246 - acc: 0.9950 - val_loss: 0.0154 - val_ac
c: 0.9952

Epoch 00004: val_loss did not improve
```

可以看出设置训练 30 层，但是使用了 EarlyStopping，训练了 4 层后，val_loss 达到最优时，就停止了。代码详见 train_pred.html 和 train_pred.ipynb。

模型训练过程中的 loss 曲线和 accuracy 曲线：





从上面 2 张图中可以看出，随着训练进行，训练集的 loss 下降，accuracy 上升，val_loss 达到最优时，停止训练。

3. 改进

模型提取特征跑的时间：

```
Using TensorFlow backend.
2018-05-20 17:14:45.051942: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFl
ow binary was not compiled to use: AVX2 FMA
Found 25000 images belonging to 2 classes.
Found 12500 images belonging to 1 classes.
2018-05-20 17:15:36.068848: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 17:15:43.966796: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 17:15:43.966800: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 17:15:47.057299: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 17:15:50.934599: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:15:55.519072: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
```

```
2018-05-20 17:41:51.000297: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 17:41:54.302158: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:41:59.152366: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:41:59.153169: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:42:05.602756: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:42:10.532798: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:42:17.283113: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:42:17.283113: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 17:42:27.556790: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 17:42:30.989267: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 17:42:30.989311: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 17:42:36.494211: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 17:42:38.997664: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 17:42:44.652514: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 17:42:44.652515: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.

real    32m17.180s
user    1372m13.389s
sys     73m10.835s
```

这步由于增加了 batch_size 为 4000，所以时间 32 分钟就可以跑完。

In [255]:

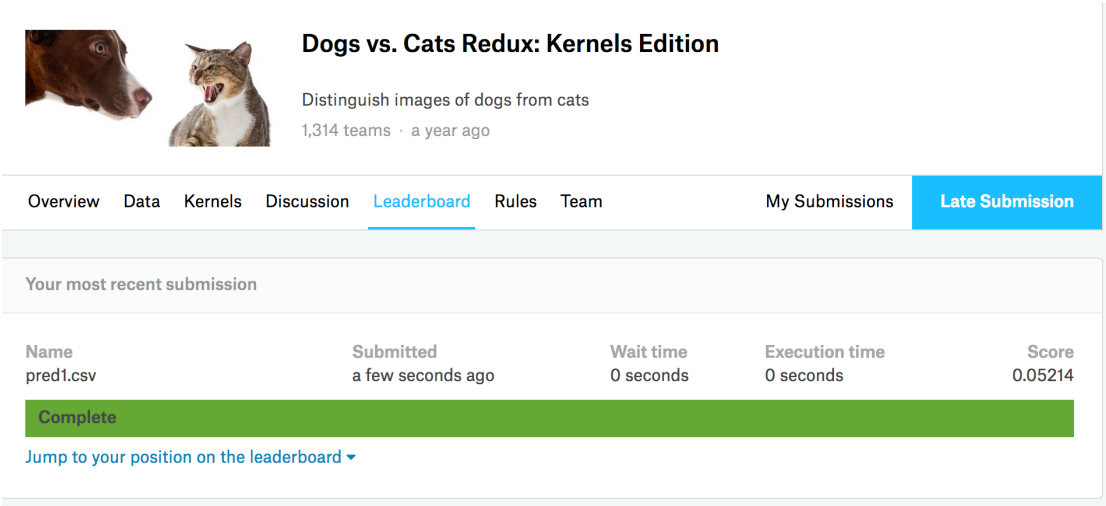
```
df = pd.read_csv("data/sample_submission.csv")
# df.head(8)

test_gen = ImageDataGenerator()
test_generator = test_gen.flow_from_directory("data/test", (299, 299), shuffle=False,
                                              batch_size=64, class_mode=None)

for i, fname in enumerate(test_generator.filenames):
    # print(i, fname, y_pred[i])
    index = int(fname[fname.rfind('/')+1:fname.rfind('.')])
#     print(index-1, y_pred[i])
    df.set_value(index-1, 'label', y_pred[i])
# index=None, 将index不写入csv文件
# df.to_csv('pred.csv')
df.to_csv('pred.csv', index=None)
df.head(8)
```

```
Found 12500 images belonging to 1 classes.
```

Out[255]:

	id	label
0	1	0.995
1	2	0.995
2	3	0.995
3	4	0.995
4	5	0.005
5	6	0.005
6	7	0.005
7	8	0.005

将 pred.csv 上传到 kaggle 看得分如下：



可以看出得分 0.05214，kaggle 公共排行榜上第 80 名为 0.05215，低于第 80 名 Score，所以可以排到第 80 名，高于 1314 的 10%。

model.fit 训练 30 轮得到的效果如下图：

```
In [382]: hist = model.fit(train, train_label,
                           epochs=30,
                           batch_size=128,
                           validation_data=(validation, validation_label))
```

```
c: 0.9942
Epoch 25/30
19940/19940 [==============================] - 7s 358us/step - loss: 0.0107 - acc: 0.9982 - val_loss: 0.0460 - val_ac
c: 0.9952
Epoch 26/30
19940/19940 [==============================] - 7s 348us/step - loss: 0.0155 - acc: 0.9980 - val_loss: 0.0424 - val_ac
c: 0.9958
Epoch 27/30
19940/19940 [==============================] - 7s 354us/step - loss: 0.0098 - acc: 0.9984 - val_loss: 0.0470 - val_ac
c: 0.9954
Epoch 28/30
19940/19940 [==============================] - 7s 359us/step - loss: 0.0131 - acc: 0.9982 - val_loss: 0.0473 - val_ac
c: 0.9944
Epoch 29/30
19940/19940 [==============================] - 7s 361us/step - loss: 0.0095 - acc: 0.9985 - val_loss: 0.0488 - val_ac
c: 0.9956
Epoch 30/30
19940/19940 [==============================] - 7s 364us/step - loss: 0.0098 - acc: 0.9987 - val_loss: 0.0514 - val_ac
c: 0.9954
```

可以看出验证集准确率达到 0.9954。这个是未加 EarlyStopping 时的训练过程，加了 EarlyStopping 之后的训练过程如下：

```
In [424]: hist = model.fit(train, train_label,
                           epochs=30,
                           batch_size=128,
                           validation_data=(validation, validation_label),
                           callbacks=[checkpointer, earlystopping])
```

```
Train on 19940 samples, validate on 4985 samples
Epoch 1/30
19940/19940 [==============================] - 12s 610us/step - loss: 0.0632 - acc: 0.9831 - val_loss: 0.0182 - val_a
cc: 0.9960

Epoch 00001: val_loss improved from inf to 0.01820, saving model to weights.hdf5
Epoch 2/30
19940/19940 [==============================] - 8s 380us/step - loss: 0.0303 - acc: 0.9932 - val_loss: 0.0149 - val_ac
c: 0.9956

Epoch 00002: val_loss improved from 0.01820 to 0.01490, saving model to weights.hdf5
Epoch 3/30
19940/19940 [==============================] - 7s 363us/step - loss: 0.0255 - acc: 0.9936 - val_loss: 0.0112 - val_ac
c: 0.9948

Epoch 00003: val_loss improved from 0.01490 to 0.01115, saving model to weights.hdf5
Epoch 4/30
19940/19940 [==============================] - 7s 330us/step - loss: 0.0246 - acc: 0.9950 - val_loss: 0.0154 - val_ac
c: 0.9952
```

训练了 4 轮，到达 val_loss 最优后，就停止了。

将此模型应用于测试集如下图：

In [388]:
```
df = pd.read_csv("data/sample_submission.csv")
# df.head(8)

test_gen = ImageDataGenerator()
test_generator = test_gen.flow_from_directory("data/test", (299, 299), shuffle=False,
                                              batch_size=64, class_mode=None)

for i, fname in enumerate(test_generator.filenames):
    # print(i, fname, y_pred[i])
    index = int(fname[fname.rfind('/')+1:fname.rfind('.')])
#     print(index-1, y_pred[i])
    df.set_value(index-1, 'label', y_pred[i])
# index=None, 将index不写入csv文件
# df.to_csv('pred.csv')
# df.to_csv('pred.csv', index=None)
df.to_csv('pred1.csv', index=None)
df.head(8)
```

Out[388]:

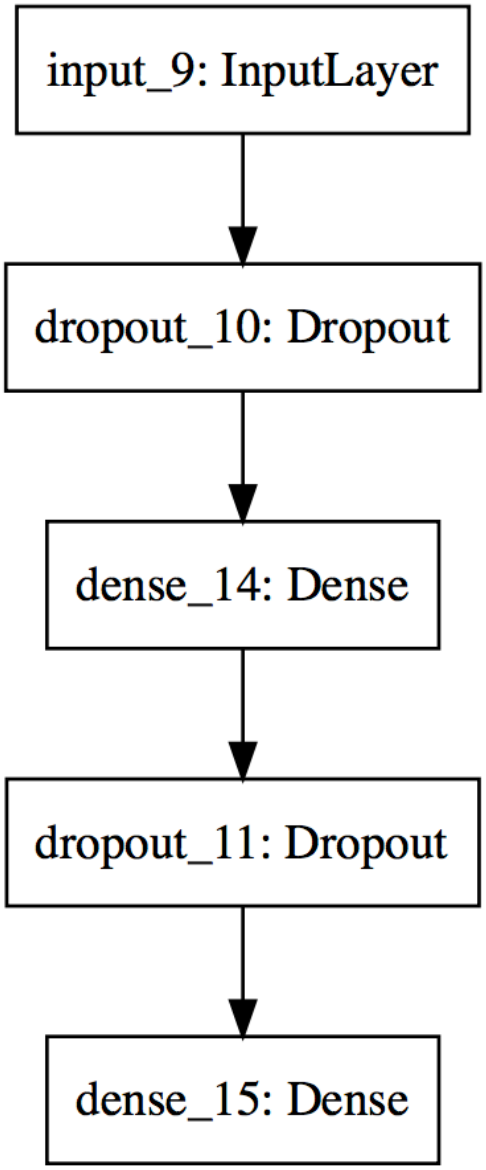
```
Found 12500 images belonging to 1 classes.
```

	id	label
0	1	0.995
1	2	0.995
2	3	0.995
3	4	0.995
4	5	0.005
5	6	0.005
6	7	0.005
7	8	0.005

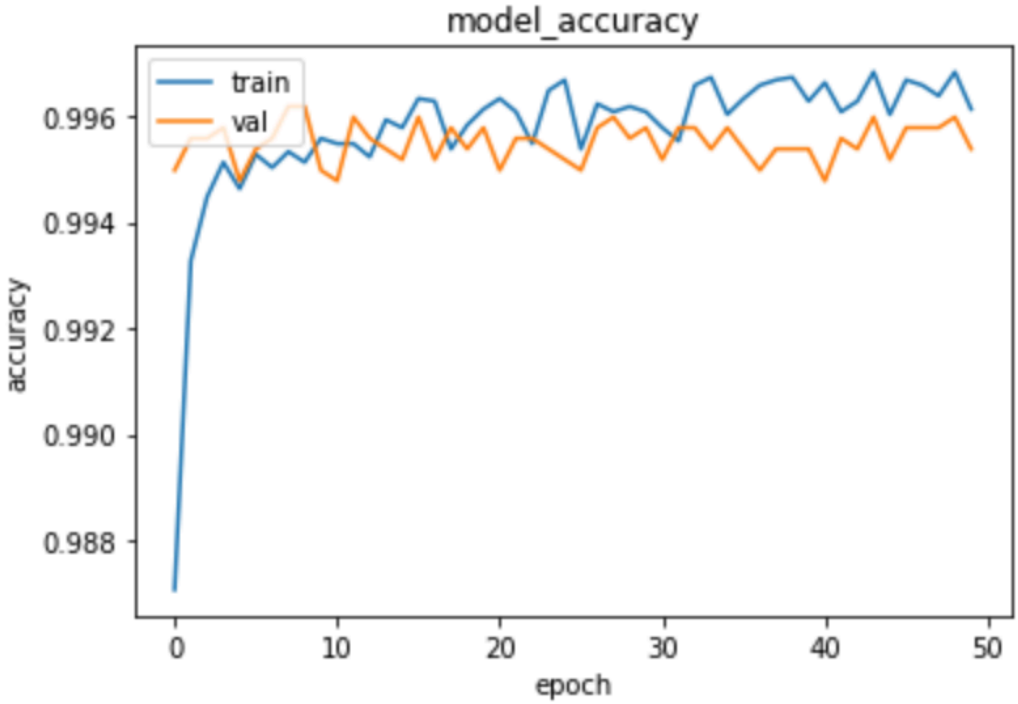
通过上面 loss 和 accuracy 曲线可以看出，出现了过拟合，感觉由于在全连接层做了两次 Dropout，每次丢掉 50%，丢弃太多，保留太少导致，可能全连接层过多导致的。改进：去掉一些全连接层和 Dropout 层。

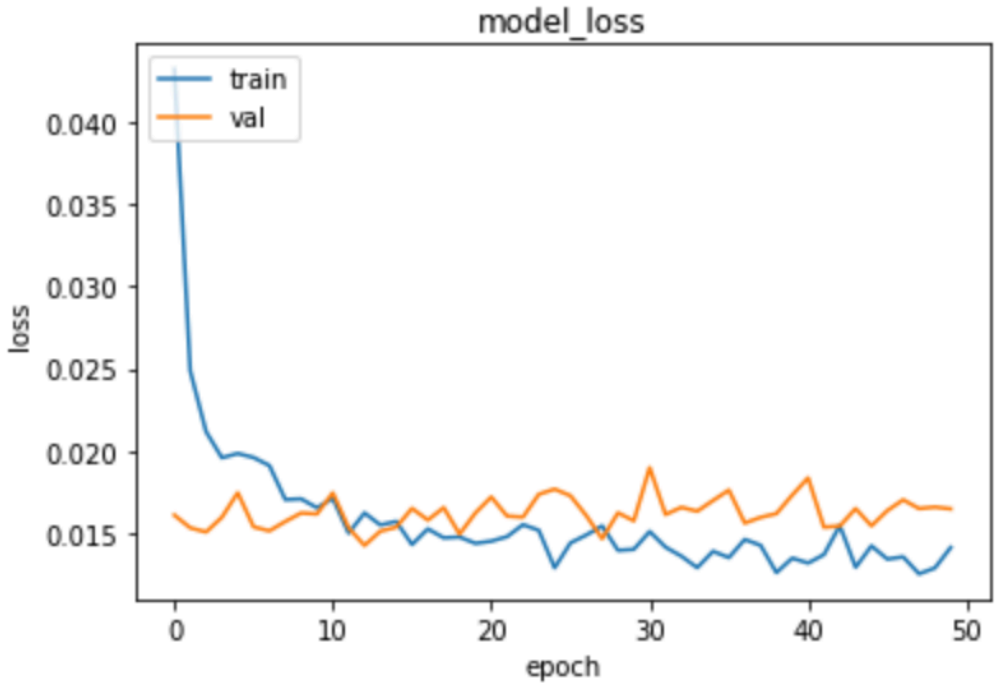
1） 先去掉 x = Dense(1024, activation='relu')(x)这个全连接层如下：

Out[82]:

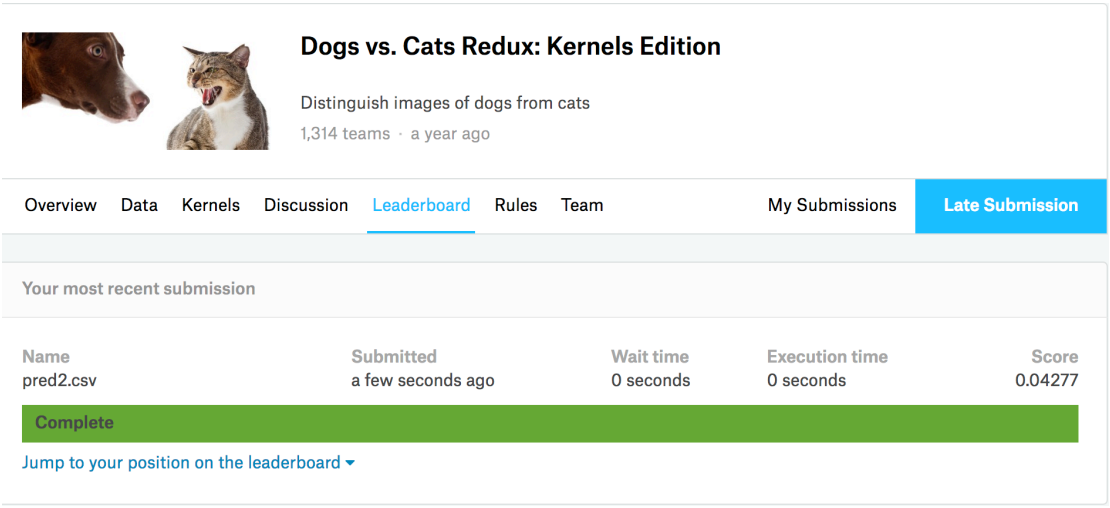


训练曲线如下：



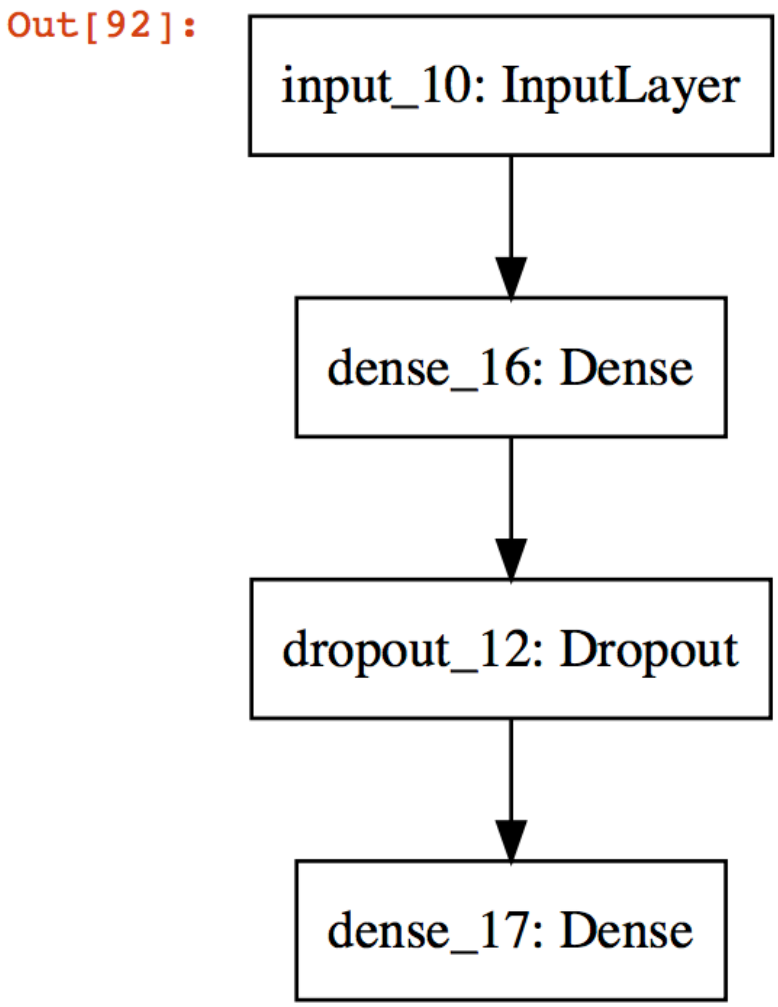


从中可以看出比第一次效果验证集 loss 低了很多。
提交 kaggle 得分：

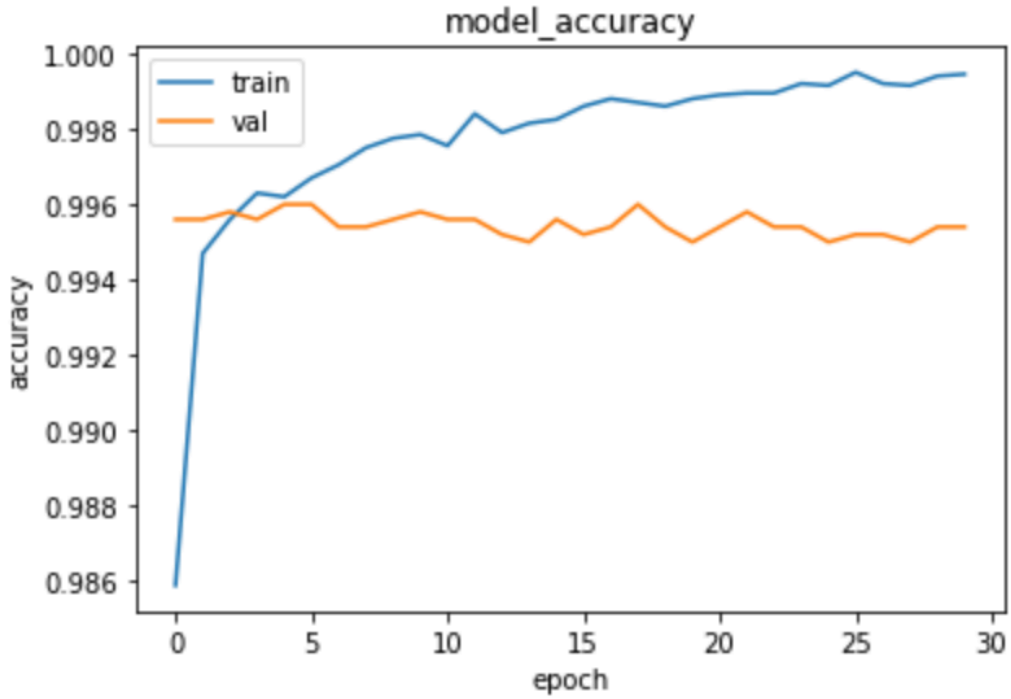


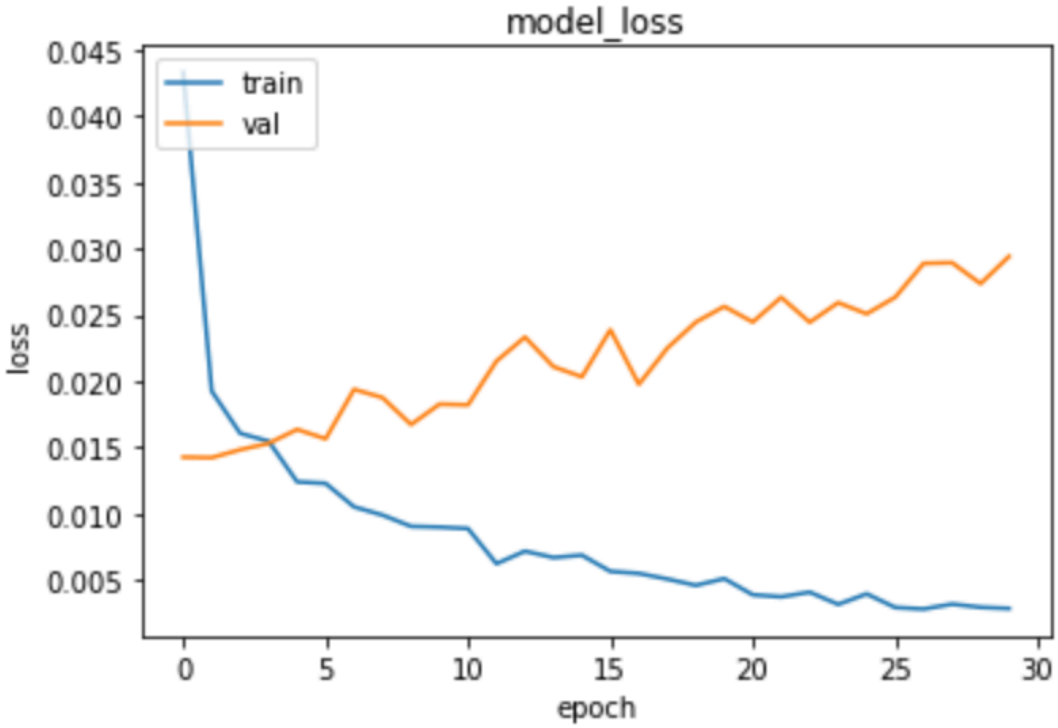
排名：22 名

2） 去掉一个 x = Dropout(0.5)(x)层如下：



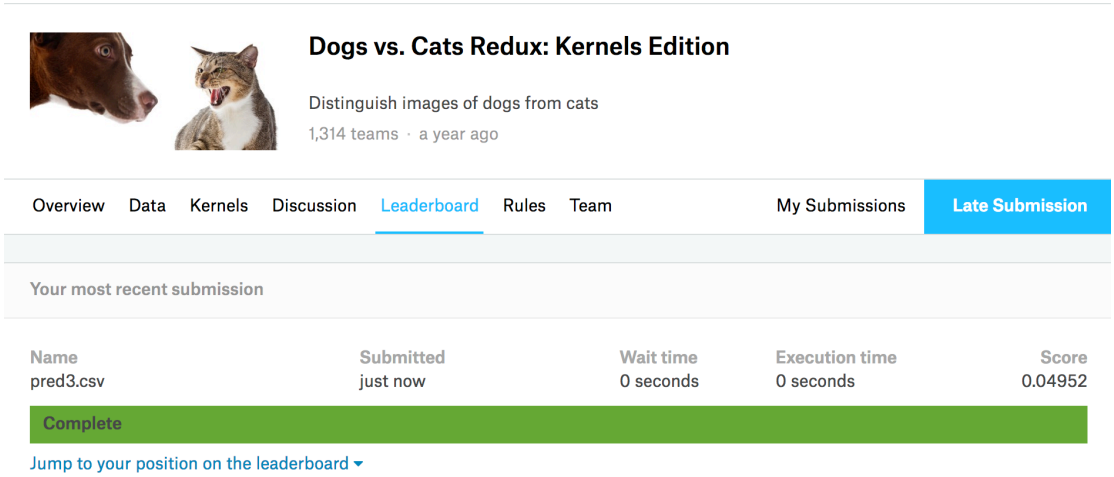
训练曲线如下：





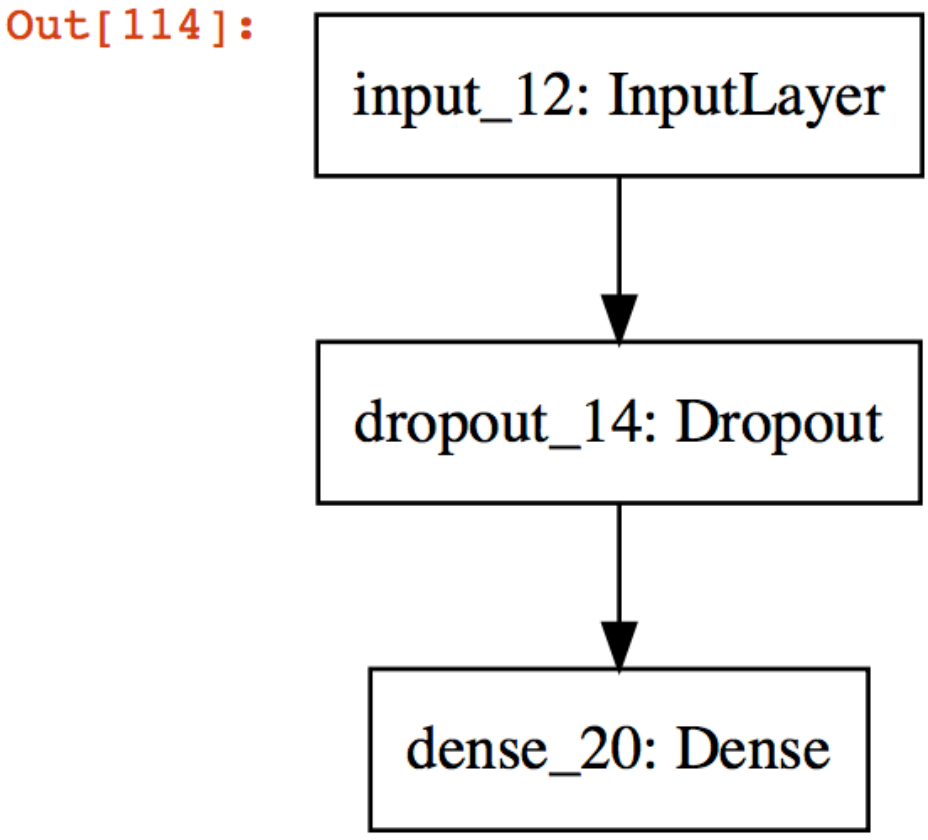
从图中看出比 1）效果差，但比刚开始好点，也有严重的过拟合现象。可以得出特征提取结束直接跟上 Dropout 层比直接跟 Dense 层效果好。

提交 kaggle：

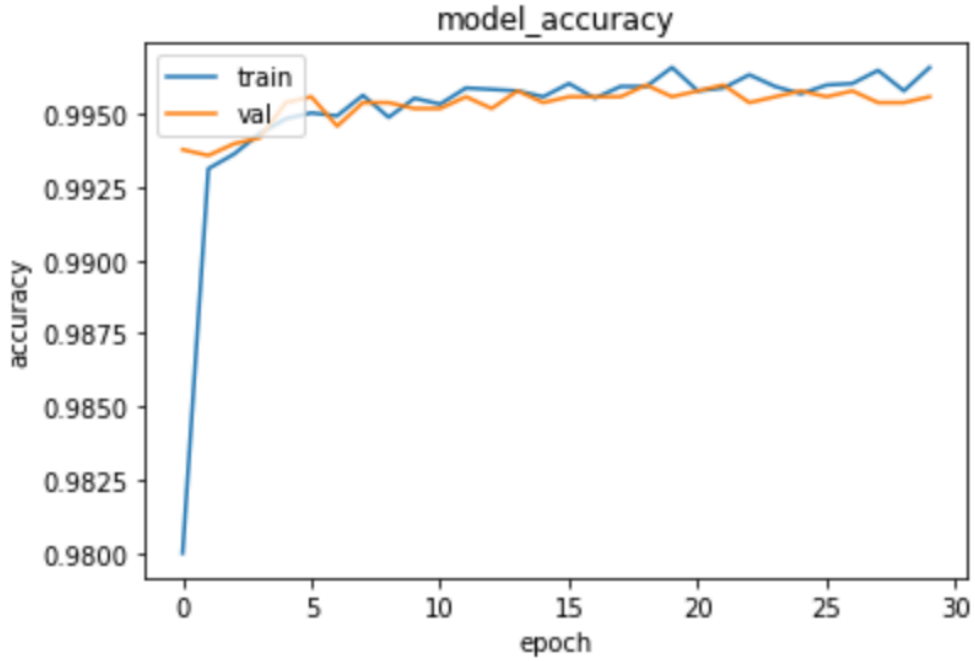


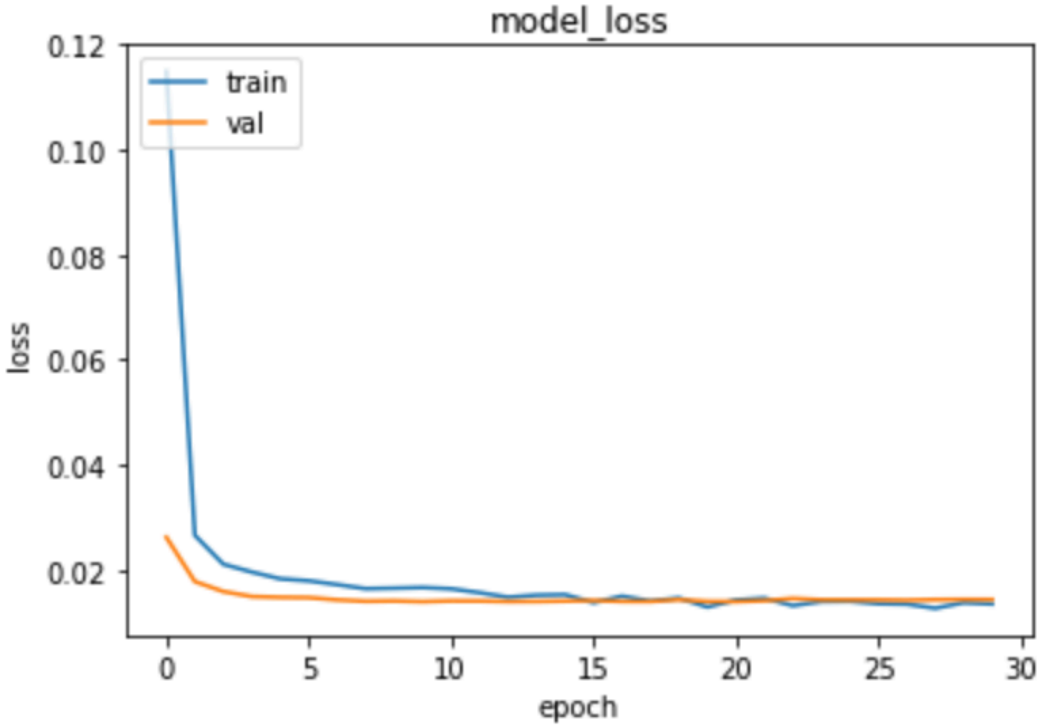
排名：58 名

3） 去掉 x = Dense(64, activation='relu')(x)层如下：



训练曲线如下：





从中可以看出此时以没有过拟合了，随着训练轮数的增加，训练集 loss 下降，验证集 loss 也下降，训练集和验证集准确率都提高了。将此时的模型应用于测试集。

提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred4.csv	just now	0 seconds	0 seconds	0.04152

Complete

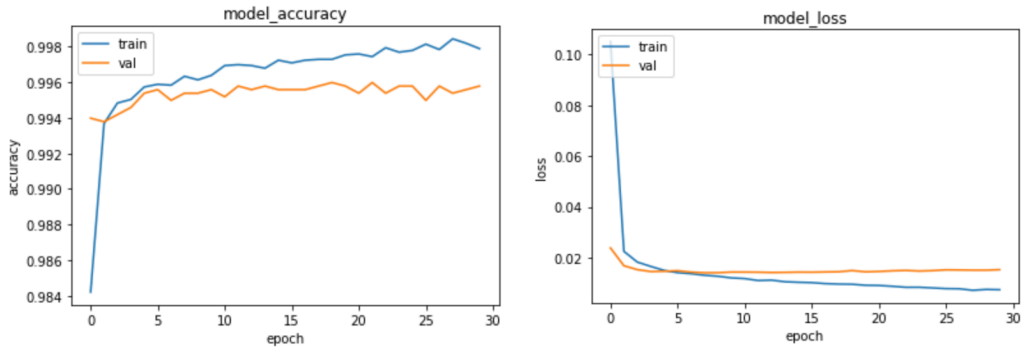
Jump to your position on the leaderboard

排名：20 名

4） 在 3）的基础上调节 Dropout 的参数。

a. x = Dropout(0.1)(x)

得到的训练曲线：



提交 kaggle：

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred5.csv	just now	1 seconds	0 seconds	0.04202

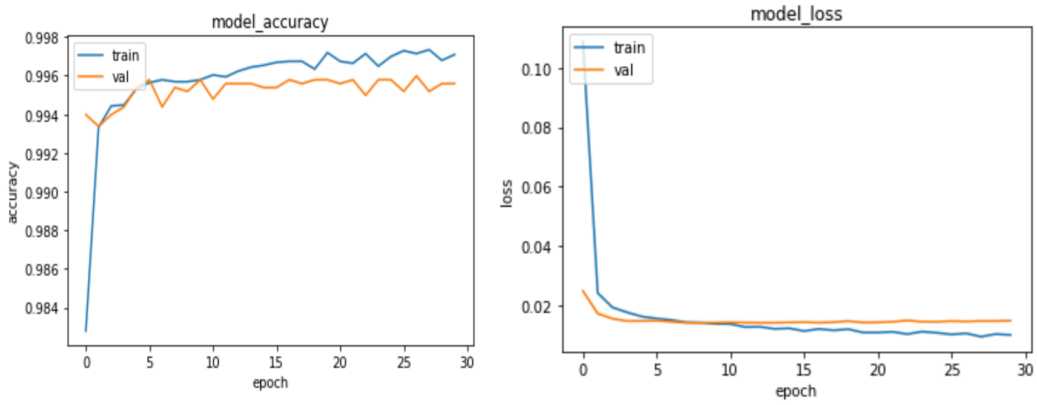
Complete

Jump to your position on the leaderboard

排名：21 名

b. x = Dropout(0.3)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred6.csv	just now	0 seconds	0 seconds	0.04163

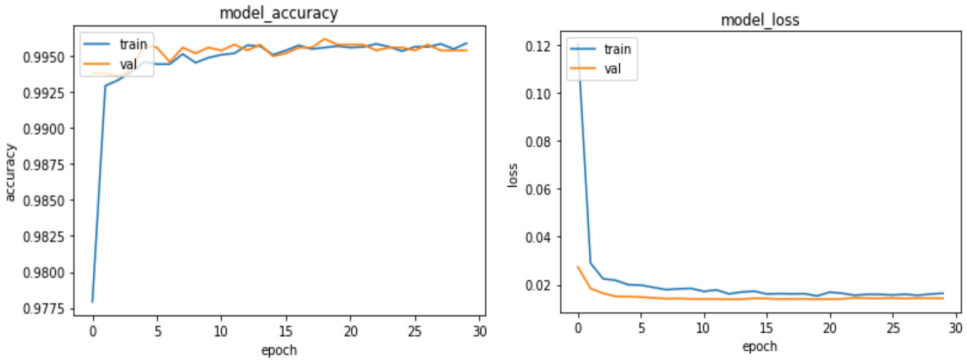
Complete

Jump to your position on the leaderboard

排名：20 名

c. x = Dropout(0.6)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred7.csv	just now	2 seconds	0 seconds	0.04157

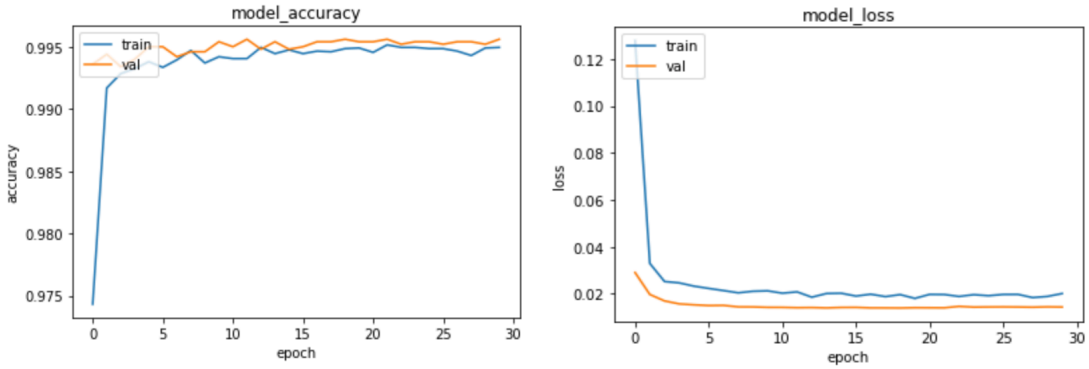
Complete

Jump to your position on the leaderboard

排名：20 名

d. x = Dropout(0.7)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred8.csv	just now	0 seconds	0 seconds	0.04135

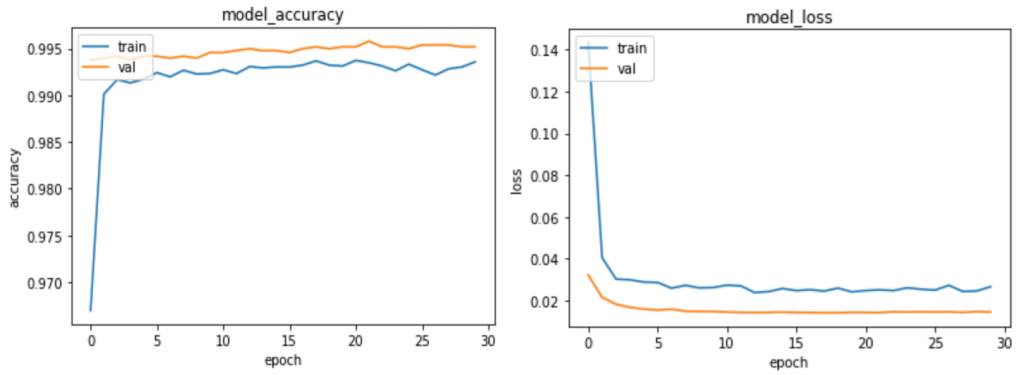
Complete

Jump to your position on the leaderboard

排名：20 名

e. x = Dropout(0.8)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred9.csv	just now	0 seconds	0 seconds	0.04187

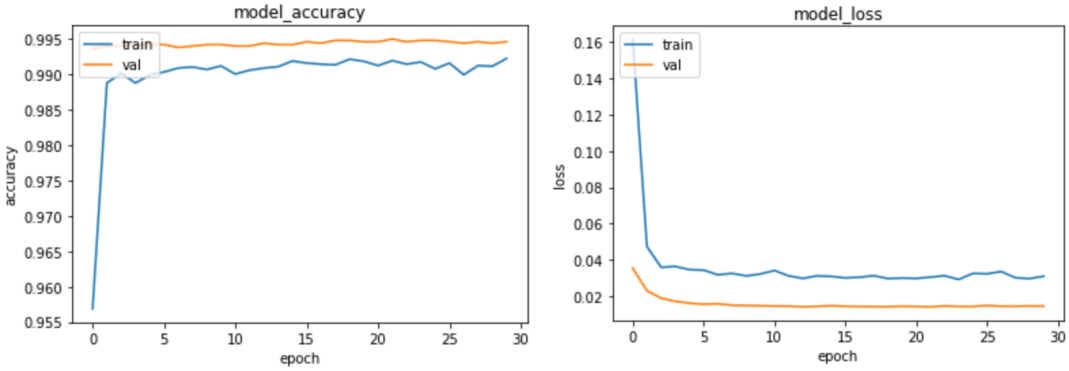
Complete

Jump to your position on the leaderboard

排名：21 名

f. x = Dropout(0.85)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred10.csv	just now	0 seconds	0 seconds	0.04206

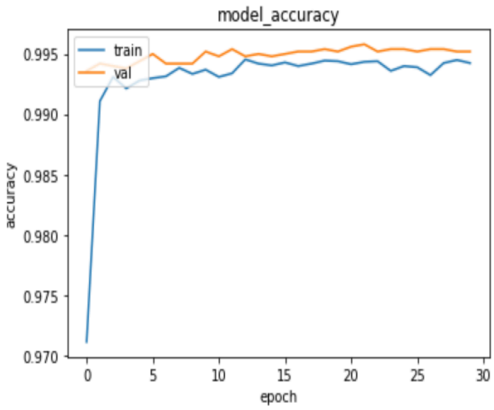
Complete

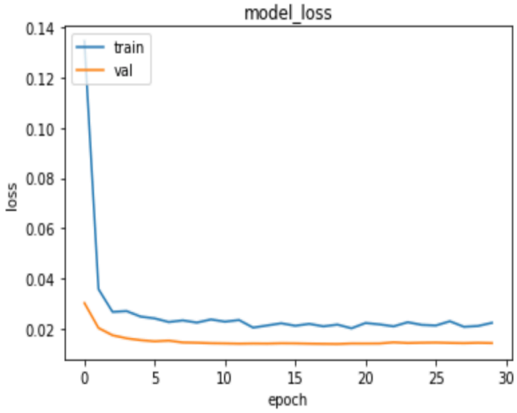
Jump to your position on the leaderboard

排名：21 名

g. x = Dropout(0.75)(x)

由于参数为参数从 0.8 往上明显 Score 变高了，所以参数设为 0.75 看一下，训练曲线：





提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred11.csv	just now	0 seconds	0 seconds	0.04170

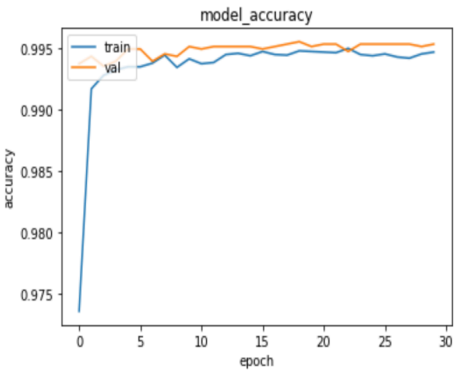
Complete

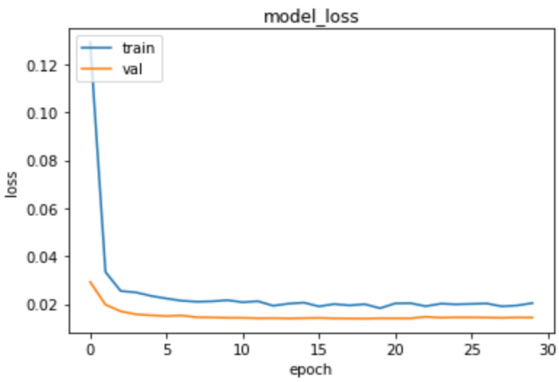
Jump to your position on the leaderboard

排名：20 名。可以看出比参数为 0.7（Score 为 0.04135）的 Score 高。下来看一下参数为 0.71。

h. x = Dropout(0.71)(x)

训练曲线：





提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred12.csv	just now	0 seconds	0 seconds	0.04134

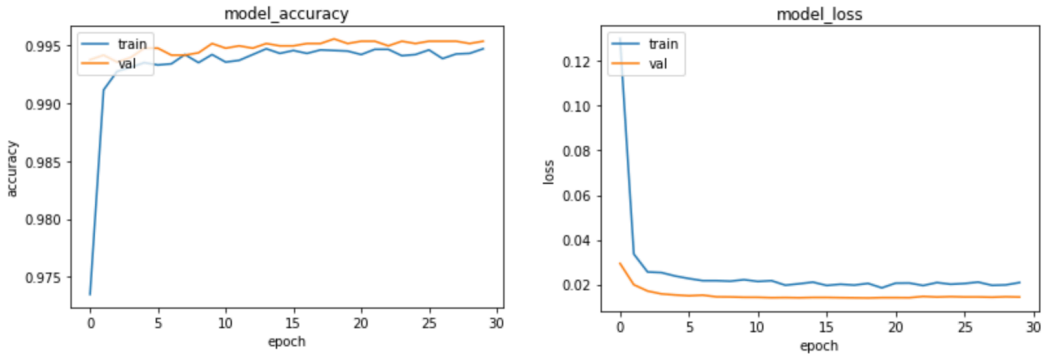
Complete

Jump to your position on the leaderboard

排名：和 19 名并列。

i. x = Dropout(0.72)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred13.csv	just now	0 seconds	0 seconds	0.04140

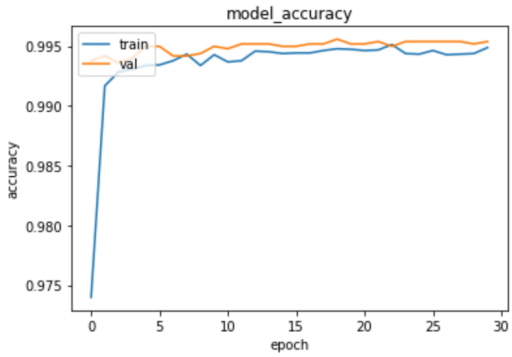
Complete

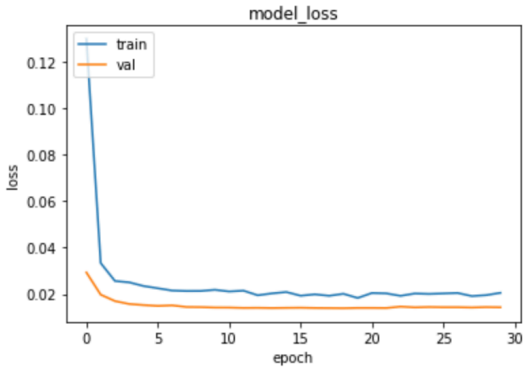
Jump to your position on the leaderboard

排名：20 名。效果比上面参数为 0.71 差了。再试一下参数为 0.715。

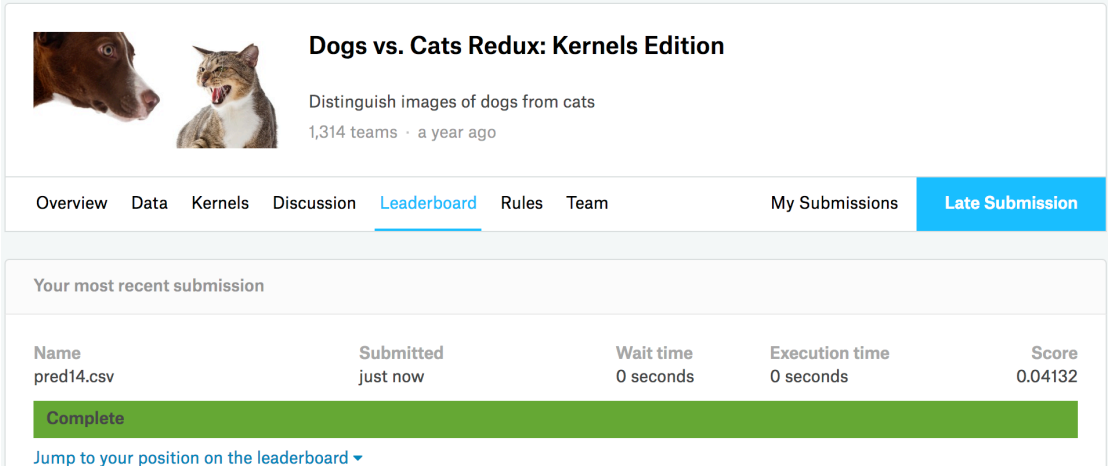
j. x = Dropout(0.715)(x)

训练曲线：





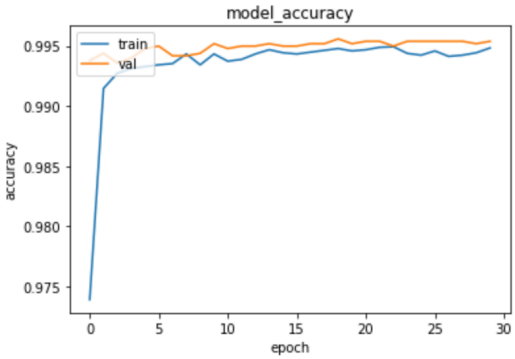
提交 kaggle：

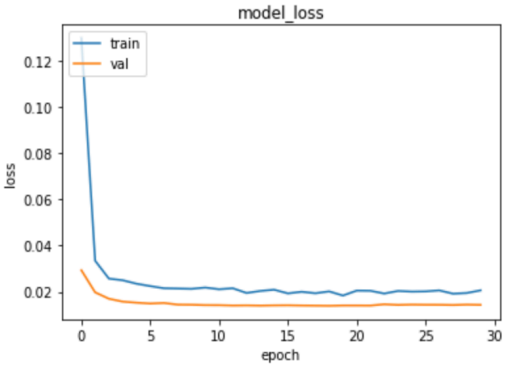


排名：19 名

k. x = Dropout(0.716)(x)

训练曲线：





提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred15.csv	just now	0 seconds	0 seconds	0.04136

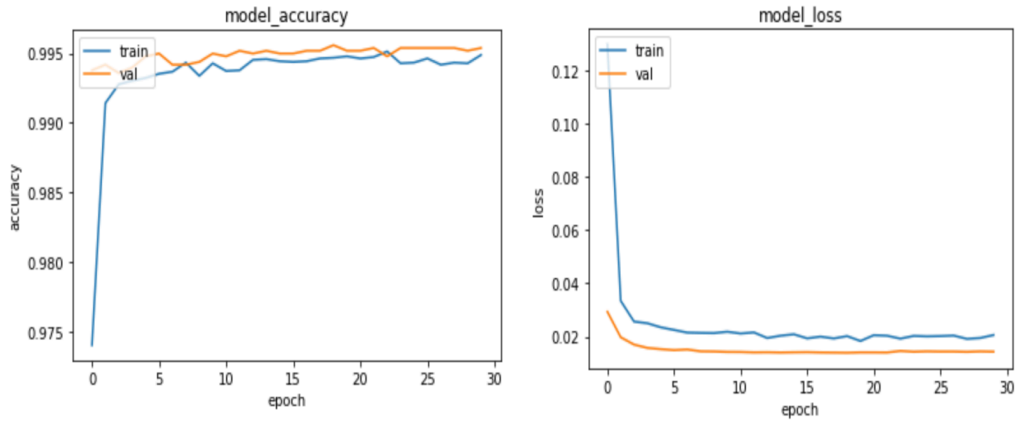
Complete

Jump to your position on the leaderboard

排名：20 名。

l. x = Dropout(0.7155)(x)

训练曲线：



提交 kaggle：

Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · a year ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
pred16.csv	just now	0 seconds	0 seconds	0.04132

Complete

Jump to your position on the leaderboard

排名：19 名。Score 得分和参数为 0.715 一样。

四、结果

Id	全连接层	Dropout 参数	Score	排名	预测文件
1	0 input_65 1 dense_110 2 dropout_6 3 dense_111 4 dropout_6 5 dense_112	2 个 Dropout 层的参数都为 0.5	0.5214	80	pred1.csv
2	去掉 Dense(1024, activation='relu')(x) 层	2 个 Dropout 层的参数都为 0.5	0.04277	22	pred2.csv
3	去掉 Dropout(0.5)(x)层	Dropout 层的参数为 0.5	0.04952	58	pred3.csv
4	去掉 Dense(64, activation='relu')(x) 层	Dropout 层的参数为 0.5	0.04152	20	pred4.csv

剩下来的层为：

0 input_31

1 dropout_34

2 dense_41

基于这些层进行 Dropout 层参数的调节：

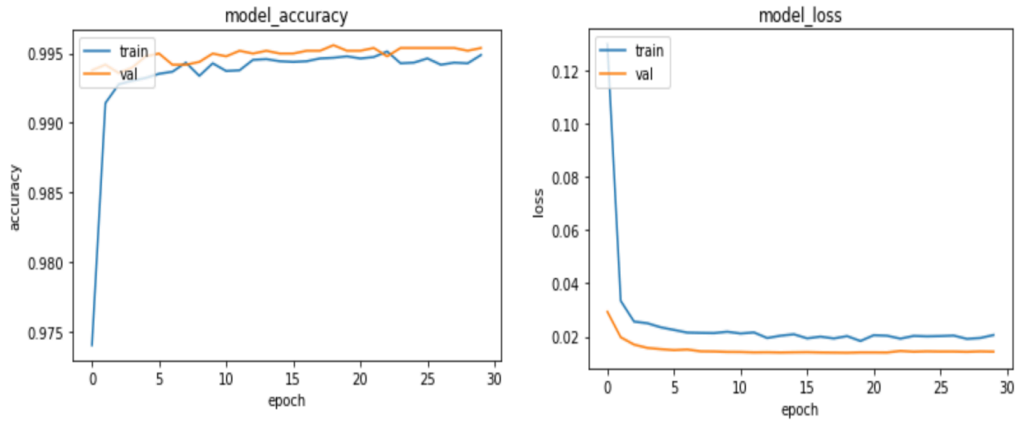
Id	Dropout 参数	Score	排名	预测文件
5	0.1	0.04202	21	pred5.csv
6	0.3	0.04163	20	pred6.csv
7	0.6	0.04157	20	pred7.csv
8	0.7	0.04135	20	pred8.csv
9	0.8	0.04187	21	pred9.csv
10	0.85	0.04206	21	pred10.csv
11	0.75	0.04170	20	pred11.csv
12	0.71	0.04134	19 并列	pred12.csv
13	0.72	0.04140	20	pred13.csv
14	0.715	0.04132	19	pred14.csv
15	0.716	0.04136	20	pred15.csv
16	0.7155	0.04132	19	pred16.csv

从中可以看出：随着全连接层的减少，Score 越低，排名越高。有一个特例，就是 Id 为 3 时虽然比 2 时多去掉了 Dropout 层，但排名从 22 下降到 58，Id 为 4 中可见排名又上升到 20，可见直接 Input 层接上 Dropout 层效果更好。下面的这个表格，在 4 的基础上进行 Dropout 参数调节，随着 Dorpout 参数从 0.1 增加一直到参数为 0.7 时 Score 分数下降，可见模型是越来越好，参数到 0.8 时，Score 分数增加了，然后调节了参数为 0.75，Score 为 0.04170，高于参数 0.7 时 Score，

低于参数 0.8 时 Score，依照这种方法试了参数为 0.71、0.72、0.715、0.716、0.7155，比较发现参数为 0.7155 和 0.715 时，模型 Score 得分最低，为 0.04132，排名为 19 名。

五、结论

猫狗数据集属于 ImageNet 数据集的子类，实验使用单模型 Xception 进行迁移学习就达到了基准指标，最终实验结果远超基准指标。模型表现最好的训练曲线如下：



随着训练轮数的增加，训练集 loss 下降，accuracy 上升，验证集 loss 下降，accuracy 略微上升。用此模型可以在 kaggle 上排名 19。

数据探索找出异常值，即数据的质量对模型很重要，利于模型提取更好的特征，防止学习一些其他特征扰乱其识别测试集。

参考文献

[1]Karen Simonyan, Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.
[2]Ian Goodfellow, Yoshua Bengio, Aaron Courville, 深度学习。
[3]周志华，机器学习。
[4]François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions, 2017.