目录

猫狗大战 2
一、定义 2
1．项目概述 2
2．问题说明 2
3. 指标 2
二、分析 3
1. 数据研究 3
2. 算法与方法 4
3. 基准测试 5
三、方法 6
1. 数据预处理 6
2. 实施 6
3. 改进 7
四、结果 11
五、结论 11
参考文献 11

猫狗大战

李华
2018 年 5 月 16 日

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。目的识别猫狗，二分类问题。

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：

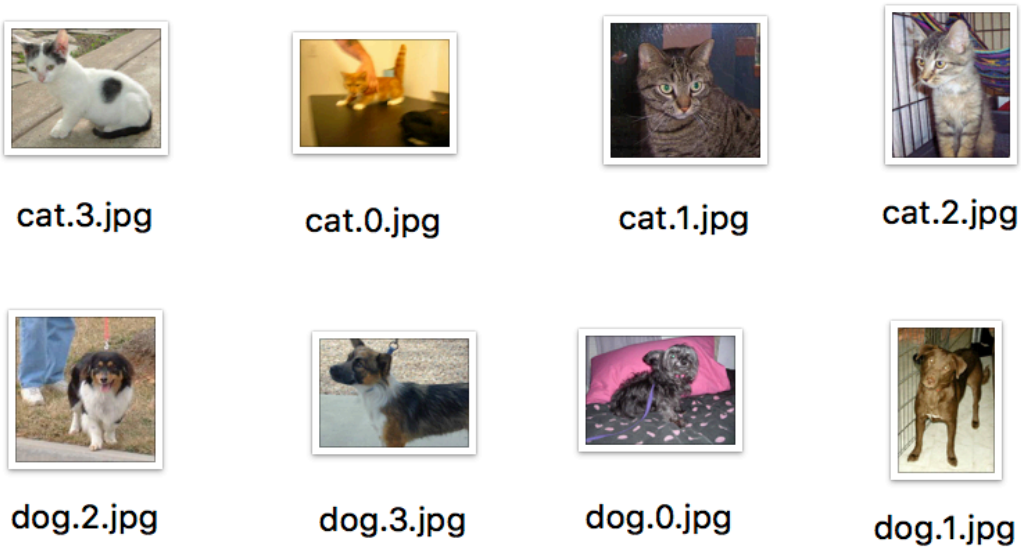


图 1. 训练数据

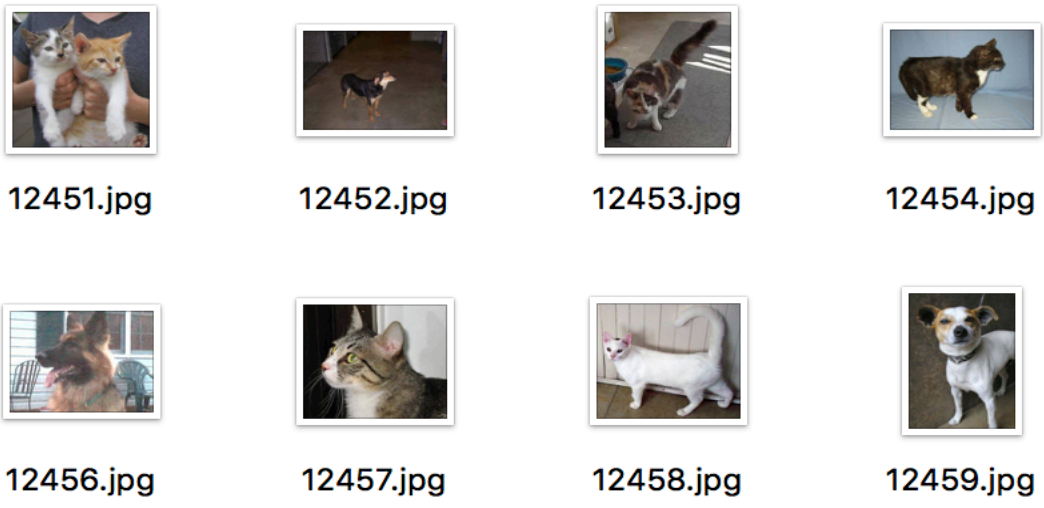


图 2. 测试数据

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。

2. 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。

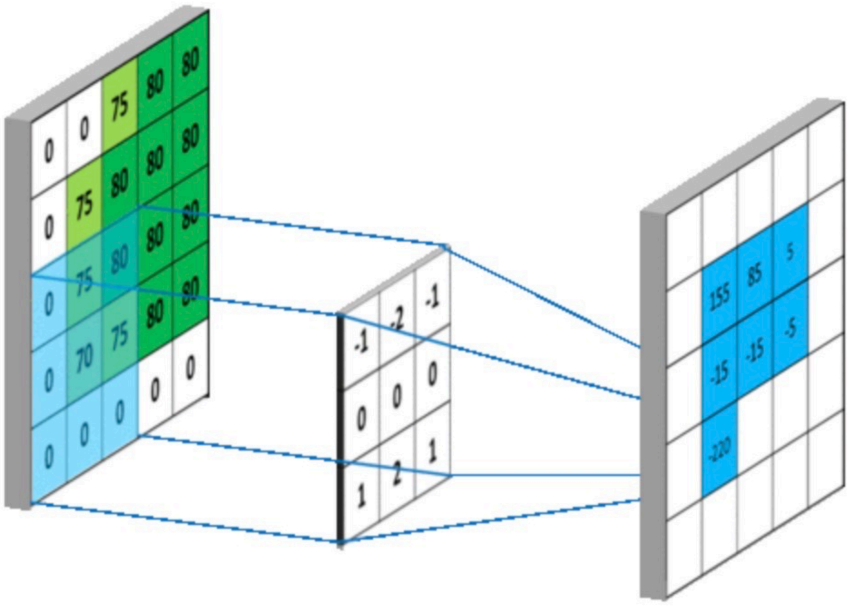


图 3. CNN 过程

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。

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。

目录结构建好之后，对训练数据进行 ImageDataGenerator 数据增强，进行倾斜，伸缩，拉伸等操作。

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 feature_b.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。

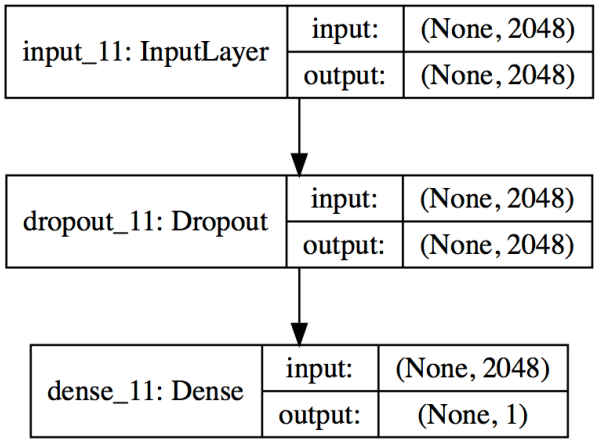
```
top - 12:14:15 up 184 days, 12 min,  2 users,  load average: 52.04, 41.75, 29.28
Tasks: 460 total,   2 running, 458 sleeping,   0 stopped,   0 zombie
%Cpu(s): 97.0 us,  2.2 sy,  0.0 ni,  0.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26347673+total, 14752612+free, 47025520 used, 68925096 buff/cache
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，对训练集进行数据增强，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 Xception1.h5 使用 sz 下载下来，在本机加载 Xception1.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：

```
# 进行可视化
SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
# plot_model(model, to_file='model.png',show_shapes=True)
```



然后模型训练，将训练集以 8:2 进行划分，80%训练，20%验证。将模型保存，并使用模型预测。

3. 改进

没有使用数据增强，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 6000，所以时间 32 分钟就可以跑完。

在 model.fit 时，训练 16 轮得到的效果产生如下图：

```
Epoch 8/16
20000/20000 [==============================] - 0s 17us/step - loss: 0.0196 - acc: 0.9935 - val_loss: 0.0207 - val_acc
: 0.9942
Epoch 9/16
20000/20000 [==============================] - 0s 18us/step - loss: 0.0183 - acc: 0.9942 - val_loss: 0.0205 - val_acc
: 0.9942
Epoch 10/16
20000/20000 [==============================] - 0s 20us/step - loss: 0.0183 - acc: 0.9940 - val_loss: 0.0208 - val_acc
: 0.9940
Epoch 11/16
20000/20000 [==============================] - 0s 22us/step - loss: 0.0181 - acc: 0.9942 - val_loss: 0.0205 - val_acc
: 0.9942
Epoch 12/16
20000/20000 [==============================] - 0s 19us/step - loss: 0.0179 - acc: 0.9942 - val_loss: 0.0200 - val_acc
: 0.9942
Epoch 13/16
20000/20000 [==============================] - 0s 20us/step - loss: 0.0173 - acc: 0.9944 - val_loss: 0.0199 - val_acc
: 0.9942
Epoch 14/16
20000/20000 [==============================] - 0s 16us/step - loss: 0.0159 - acc: 0.9947 - val_loss: 0.0197 - val_acc
: 0.9944
Epoch 15/16
20000/20000 [==============================] - 0s 16us/step - loss: 0.0164 - acc: 0.9951 - val_loss: 0.0197 - val_acc
: 0.9940
Epoch 16/16
20000/20000 [==============================] - 0s 24us/step - loss: 0.0161 - acc: 0.9953 - val_loss: 0.0197 - val_acc
: 0.9942
```

可以看出验证集上准确率达到 0.9942。

将此模型应用于测试集如下图：

In [255]:

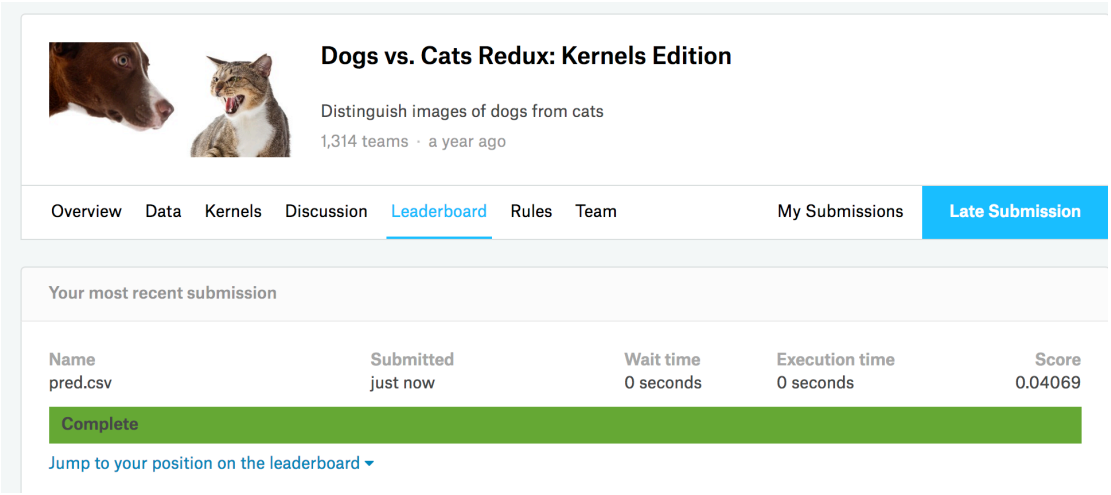
```
df = pd.read_csv("data/sample_submission.csv")
# df.head(8)

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 0.04069，kaggle 公共排行榜上第 17 名为 0.04077，低于第 17 名 Score，所以可以排到第 17 名，远远高于 1314 的 10%。

使用数据增强，提取特征过程图如下：

```
Using TensorFlow backend.
2018-05-20 20:08:11.644499: 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 20:10:47.816361: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:10:55.660912: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:10:55.660914: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:10:58.735013: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:11:02.198223: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:11:07.287322: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:11:07.287322: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:11:13.465401: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:11:18.448822: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:11:25.066767: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
```

```
2018-05-20 20:37:05.838369: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:37:05.838400: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:37:08.896939: W tensorflow/core/framework/allocator.cc:101] Allocation of 33191424000 exceeds 10% of system memory.
2018-05-20 20:37:12.306165: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:17.356471: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:17.356471: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:23.470894: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:28.391083: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:35.001766: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:35.001767: W tensorflow/core/framework/allocator.cc:101] Allocation of 66382848000 exceeds 10% of system memory.
2018-05-20 20:37:45.056184: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 20:37:48.649475: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 20:37:48.649475: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 20:37:54.165342: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 20:37:56.669220: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 20:38:02.327373: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.
2018-05-20 20:38:02.327373: W tensorflow/core/framework/allocator.cc:101] Allocation of 33644544000 exceeds 10% of system memory.

real    34m9.671s
user    1386m40.751s
sys     68m52.331s
```

提取特征花费了 34 分钟。

model.fit 训练 16 轮得到的效果如下图：

```
: 0.9906
Epoch 8/16
20000/20000 [==============================] - 0s 18us/step - loss: 0.0322 - acc: 0.9882 - val_loss: 0.0286 - val_acc
: 0.9912
Epoch 9/16
20000/20000 [==============================] - 0s 21us/step - loss: 0.0304 - acc: 0.9898 - val_loss: 0.0277 - val_acc
: 0.9910
Epoch 10/16
20000/20000 [==============================] - 0s 16us/step - loss: 0.0305 - acc: 0.9886 - val_loss: 0.0274 - val_acc
: 0.9910
Epoch 11/16
20000/20000 [==============================] - 0s 16us/step - loss: 0.0304 - acc: 0.9892 - val_loss: 0.0268 - val_acc
: 0.9918
Epoch 12/16
20000/20000 [==============================] - 0s 18us/step - loss: 0.0285 - acc: 0.9900 - val_loss: 0.0269 - val_acc
: 0.9916
Epoch 13/16
20000/20000 [==============================] - 0s 16us/step - loss: 0.0288 - acc: 0.9890 - val_loss: 0.0264 - val_acc
: 0.9918
Epoch 14/16
20000/20000 [==============================] - 0s 17us/step - loss: 0.0276 - acc: 0.9901 - val_loss: 0.0263 - val_acc
: 0.9920
Epoch 15/16
20000/20000 [==============================] - 0s 19us/step - loss: 0.0273 - acc: 0.9901 - val_loss: 0.0262 - val_acc
: 0.9916
Epoch 16/16
20000/20000 [==============================] - 0s 19us/step - loss: 0.0273 - acc: 0.9904 - val_loss: 0.0260 - val_acc
: 0.9920
```

Out[303]: <keras.callbacks.History at 0x11b9c0860>

可以看出验证集准确率达到 0.9920，从验证集准确率上看数据增强的准确率低于未增强时的验证准确率。

将此模型应用于测试集如下图：

In [269]:

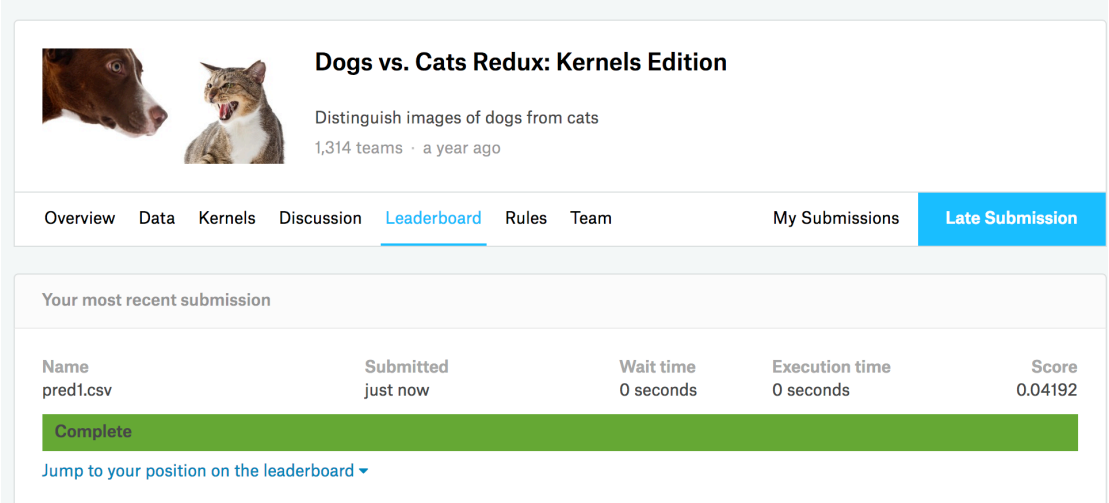
```
df = pd.read_csv("data/sample_submission.csv")
# df.head(8)

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)
```

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

Out[269]:

	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

将产生的 pred1.csv 文件上传 kaggle 看得分：



得分为 0.04192，kaggle 上第 21 名为 0.4214，所以排名第 21 名。高于 131 名。

四、结果

将生成的两个 csv 文件提交到 kaggle 看得分。

经过对比可以看出来数据增强排名反而低于不增强，可能是由于对图片翻转拉伸有些过度，或者是一些异常值导致的，不过再 Imagenet 上训练过的应用于此排行都高于 1314 的 10%。

数据不增强得分 0.04069，排名 17 名，数据增强得分 0.04192，排名 21 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。

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.