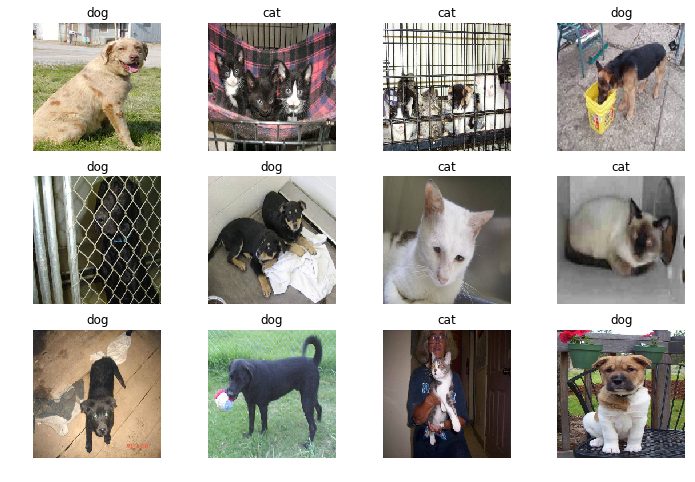
本文会通过 Keras 搭建一个深度卷积神经网络来识别一张图片是猫还是狗，在验证集上的准确率可以达到99.6%，建议使用显卡来运行该项目。本项目使用的 Keras 版本是1.2.2。如果你使用的是更高级的版本，可能会稍有参数变化。

# 猫狗大战

数据集来自 kaggle 上的一个竞赛：[Dogs vs. Cats](https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition)，训练集有25000张，猫狗各占一半。测试集12500张，没有标定是猫还是狗。

➜ 猫狗大战 ls train | head  
cat.0.jpg  
cat.1.jpg  
cat.10.jpg  
cat.100.jpg  
cat.1000.jpg  
cat.10000.jpg  
cat.10001.jpg  
cat.10002.jpg  
cat.10003.jpg  
cat.10004.jpg  
➜ 猫狗大战 ls test | head  
1.jpg  
10.jpg  
100.jpg  
1000.jpg  
10000.jpg  
10001.jpg  
10002.jpg  
10003.jpg  
10004.jpg  
10005.jpg

下面是训练集的一部分例子：



# 数据预处理

由于我们的数据集的文件名是以type.num.jpg这样的方式命名的，比如cat.0.jpg，但是使用 Keras 的 ImageDataGenerator 需要将不同种类的图片分在不同的文件夹中，因此我们需要对数据集进行预处理。这里我们采取的思路是创建符号链接(symbol link)，这样的好处是不用复制一遍图片，占用不必要的空间。

import os  
import shutil  
  
train\_filenames = os.listdir('train')  
train\_cat = filter(lambda x:x[:3] == 'cat', train\_filenames)  
train\_dog = filter(lambda x:x[:3] == 'dog', train\_filenames)  
  
def rmrf\_mkdir(dirname):  
 if os.path.exists(dirname):  
 shutil.rmtree(dirname)  
 os.mkdir(dirname)  
  
rmrf\_mkdir('train2')  
os.mkdir('train2/cat')  
os.mkdir('train2/dog')  
  
rmrf\_mkdir('test2')  
os.symlink('../test/', 'test2/test')  
  
for filename in train\_cat:  
 os.symlink('../../train/'+filename, 'train2/cat/'+filename)  
  
for filename in train\_dog:  
 os.symlink('../../train/'+filename, 'train2/dog/'+filename)

我们可以从下面看到文件夹的结构，train2里面有两个文件夹，分别是猫和狗，每个文件夹里是12500张图。

├── test [12500 images]  
├── test.zip  
├── test2  
│   └── test -> ../test/  
├── train [25000 images]  
├── train.zip  
└── train2  
 ├── cat [12500 images]  
 └── dog [12500 images]

# 导出特征向量

对于这个题目来说，使用预训练的网络是最好不过的了，经过前期的测试，我们测试了 ResNet50 等不同的网络，但是排名都不高，现在看来只有一两百名的样子，所以我们需要提高我们的模型表现。那么一种有效的方法是综合各个不同的模型，从而得到不错的效果，兼听则明。如果是直接在一个巨大的网络后面加我们的全连接，那么训练10代就需要跑十次巨大的网络，而且我们的卷积层都是不可训练的，那么这个计算就是浪费的。所以我们可以将多个不同的网络输出的特征向量先保存下来，以便后续的训练，这样做的好处是我们一旦保存了特征向量，即使是在普通笔记本上也能轻松训练。

from keras.models import \*  
from keras.layers import \*  
from keras.applications import \*  
from keras.preprocessing.image import \*  
  
import h5py  
  
def write\_gap(MODEL, image\_size, lambda\_func=None):  
 width = image\_size[0]  
 height = image\_size[1]  
 input\_tensor = Input((height, width, 3))  
 x = input\_tensor  
 if lambda\_func:  
 x = Lambda(lambda\_func)(x)  
   
 base\_model = MODEL(input\_tensor=x, weights='imagenet', include\_top=False)  
 model = Model(base\_model.input, GlobalAveragePooling2D()(base\_model.output))  
  
 gen = ImageDataGenerator()  
 train\_generator = gen.flow\_from\_directory("train2", image\_size, shuffle=False,   
 batch\_size=16)  
 test\_generator = gen.flow\_from\_directory("test2", image\_size, shuffle=False,   
 batch\_size=16, class\_mode=None)  
  
 train = model.predict\_generator(train\_generator, train\_generator.nb\_sample)  
 test = model.predict\_generator(test\_generator, test\_generator.nb\_sample)  
 with h5py.File("gap\_%s.h5"%MODEL.func\_name) as h:  
 h.create\_dataset("train", data=train)  
 h.create\_dataset("test", data=test)  
 h.create\_dataset("label", data=train\_generator.classes)  
  
write\_gap(ResNet50, (224, 224))  
write\_gap(InceptionV3, (299, 299), inception\_v3.preprocess\_input)  
write\_gap(Xception, (299, 299), xception.preprocess\_input)

为了复用代码，我觉得写一个函数是非常有必要的，那么我们的函数就需要输入模型，输入图片的大小，以及[预处理函数](https://github.com/fchollet/keras/blob/master/keras/applications/inception_v3.py#L389-L393)，因为 Xception 和 Inception V3 都需要将数据限定在 (-1, 1) 的范围内，然后我们利用 GlobalAveragePooling2D 将卷积层输出的每个激活图直接求平均值，不然输出的文件会非常大，且容易过拟合。然后我们定义了两个 generator，利用 model.predict\_generator 函数来导出特征向量，最后我们选择了 ResNet50, Xception, Inception V3 这三个模型（如果有兴趣也可以导出 VGG 的特征向量）。每个模型导出的时间都挺长的，在 aws p2.xlarge 上大概需要用**十分钟到二十分钟**。 这三个模型都是在 [ImageNet](http://www.image-net.org/) 上面预训练过的，所以每一个模型都可以说是身经百战，通过这三个老司机导出的特征向量，可以高度概括一张图片有哪些内容。

最后导出的 h5 文件包括三个 numpy 数组：

* train (25000, 2048)
* test (12500, 2048)
* label (25000,)

如果你不想自己计算特征向量，可以直接在这里下载导出的文件：[GitHub releases](https://github.com/ypwhs/dogs_vs_cats/releases/tag/gap) [百度云](https://pan.baidu.com/s/1pK7psxX#list/path=%2Fdataset%2FDogs%20vs%20Cats)

参考资料：

* [ResNet](https://arxiv.org/abs/1512.03385) 15.12
* [Inception v3](https://arxiv.org/abs/1512.00567) 15.12
* [Xception](https://arxiv.org/abs/1610.02357) 16.10

# 载入特征向量

经过上面的代码以后，我们获得了三个特征向量文件，分别是：

* gap\_ResNet50.h5
* gap\_InceptionV3.h5
* gap\_Xception.h5

我们需要载入这些特征向量，并且将它们合成一条特征向量，然后记得把 X 和 y 打乱，不然之后我们设置validation\_split的时候会出问题。这里设置了 numpy 的随机数种子为2017，这样可以确保每个人跑这个代码，输出都能是一样的结果。

import h5py  
import numpy as np  
from sklearn.utils import shuffle  
np.random.seed(2017)  
  
X\_train = []  
X\_test = []  
  
for filename in ["gap\_ResNet50.h5", "gap\_Xception.h5", "gap\_InceptionV3.h5"]:  
 with h5py.File(filename, 'r') as h:  
 X\_train.append(np.array(h['train']))  
 X\_test.append(np.array(h['test']))  
 y\_train = np.array(h['label'])  
  
X\_train = np.concatenate(X\_train, axis=1)  
X\_test = np.concatenate(X\_test, axis=1)  
  
X\_train, y\_train = shuffle(X\_train, y\_train)

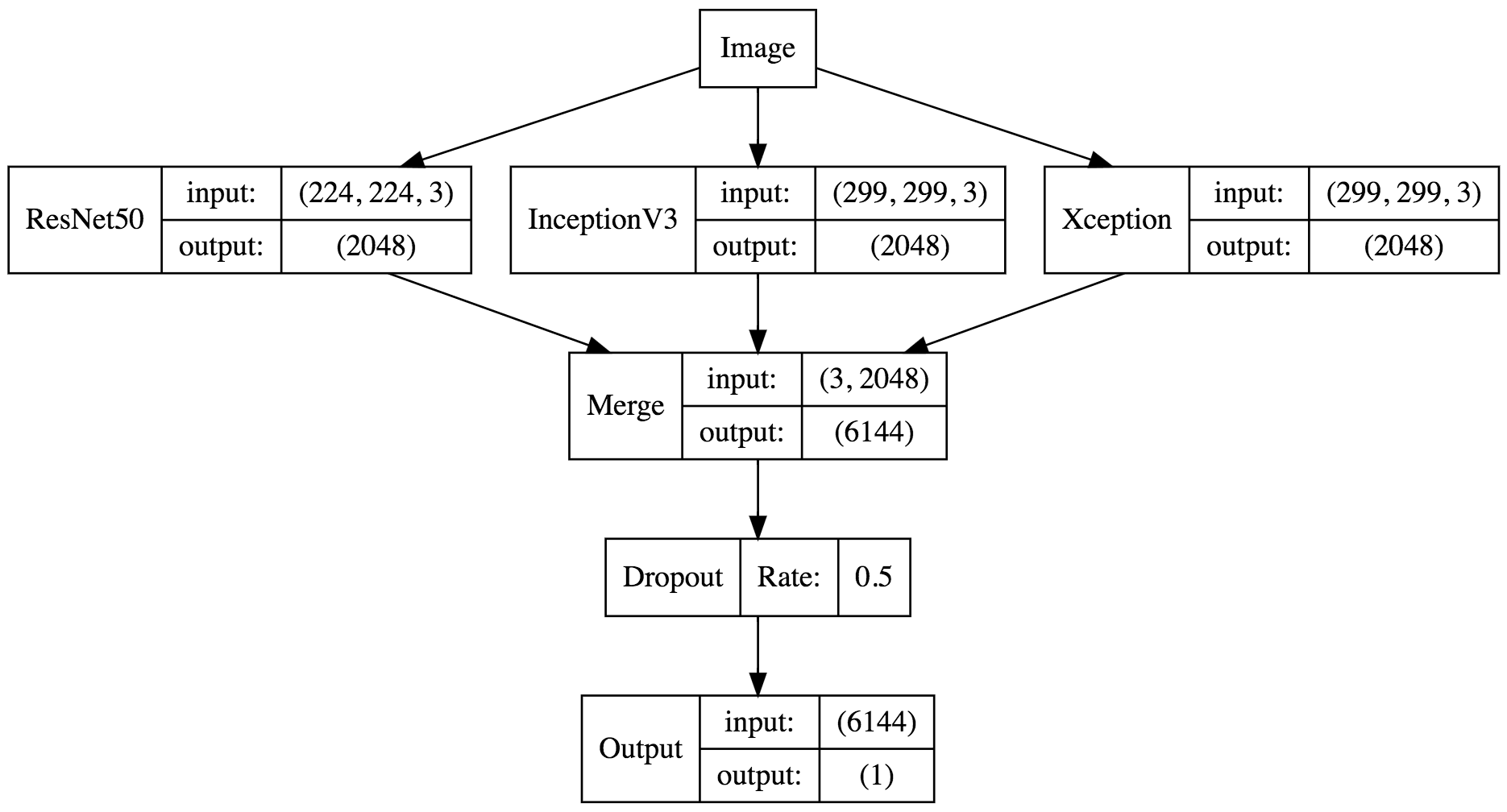
# 构建模型

模型的构建很简单，直接 dropout 然后分类就好了。

from keras.models import \*  
from keras.layers import \*  
  
np.random.seed(2017)  
  
input\_tensor = Input(X\_train.shape[1:])  
x = Dropout(0.5)(input\_tensor)  
x = Dense(1, activation='sigmoid')(x)  
model = Model(input\_tensor, x)  
  
model.compile(optimizer='adadelta',  
 loss='binary\_crossentropy',  
 metrics=['accuracy'])

我们还可以对模型进行可视化：

digraph G{  
 node [shape=record]  
 a[label="ResNet50|{input:|output:}|{(224, 224, 3)|(2048)}"]  
 b[label="InceptionV3|{input:|output:}|{(299, 299, 3)|(2048)}"]  
 c[label="Xception|{input:|output:}|{(299, 299, 3)|(2048)}"]  
 Merge[label="Merge|{input:|output:}|{(3, 2048)|(6144)}"]  
 Dropout[label="Dropout|Rate:|0.5"]  
 Output[label="Output|{input:|output:}|{(6144)|(1)}"]  
 Image -> a -> Merge  
 Image -> b -> Merge  
 Image -> c -> Merge  
 Merge -> Dropout -> Output  
}



# 训练模型

模型构件好了以后，我们就可以进行训练了，这里我们设置验证集大小为 20% ，也就是说训练集是20000张图，验证集是5000张图。

model.fit(X\_train, y\_train, batch\_size=128, nb\_epoch=8, validation\_split=0.2)

Train on 20000 samples, validate on 5000 samples  
Epoch 1/8  
20000/20000 [==============================] - 1s - loss: 0.1193 - acc: 0.9591 - val\_loss: 0.0283 - val\_acc: 0.9936  
Epoch 2/8  
20000/20000 [==============================] - 0s - loss: 0.0319 - acc: 0.9898 - val\_loss: 0.0181 - val\_acc: 0.9952  
Epoch 3/8  
20000/20000 [==============================] - 0s - loss: 0.0252 - acc: 0.9916 - val\_loss: 0.0172 - val\_acc: 0.9934  
Epoch 4/8  
20000/20000 [==============================] - 0s - loss: 0.0214 - acc: 0.9936 - val\_loss: 0.0140 - val\_acc: 0.9956  
Epoch 5/8  
20000/20000 [==============================] - 0s - loss: 0.0200 - acc: 0.9926 - val\_loss: 0.0139 - val\_acc: 0.9954  
Epoch 6/8  
20000/20000 [==============================] - 0s - loss: 0.0189 - acc: 0.9933 - val\_loss: 0.0129 - val\_acc: 0.9956  
Epoch 7/8  
20000/20000 [==============================] - 0s - loss: 0.0170 - acc: 0.9946 - val\_loss: 0.0123 - val\_acc: 0.9960  
Epoch 8/8  
20000/20000 [==============================] - 0s - loss: 0.0163 - acc: 0.9945 - val\_loss: 0.0119 - val\_acc: 0.9958  
Out[4]:

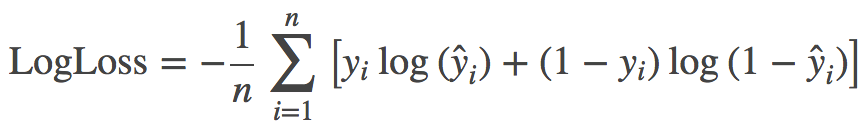
我们可以看到，训练的过程很快，十秒以内就能训练完，准确率也很高，在验证集上最高达到了99.6%的准确率，这相当于一千张图只错了4张，可以说比我还厉害。

# 预测测试集

模型训练好以后，我们就可以对测试集进行预测，然后提交到 kaggle 上看看最终成绩了。

y\_pred = model.predict(X\_test, verbose=1)  
y\_pred = y\_pred.clip(min=0.005, max=0.995)  
  
import pandas as pd  
from keras.preprocessing.image import \*  
  
df = pd.read\_csv("sample\_submission.csv")  
  
gen = ImageDataGenerator()  
test\_generator = gen.flow\_from\_directory("test2", (224, 224), shuffle=False,   
 batch\_size=16, class\_mode=None)  
  
for i, fname in enumerate(test\_generator.filenames):  
 index = int(fname[fname.rfind('/')+1:fname.rfind('.')])  
 df.set\_value(index-1, 'label', y\_pred[i])  
  
df.to\_csv('pred.csv', index=None)  
df.head(10)

预测这里我们用到了一个小技巧，我们将每个预测值限制到了 [0.005, 0.995] 个区间内，这个原因很简单，kaggle 官方的评估标准是 [LogLoss](https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/details/evaluation)，对于预测正确的样本，0.995 和 1 相差无几，但是对于预测错误的样本，0 和 0.005 的差距非常大，是 15 和 2 的差别。参考 [LogLoss 如何处理无穷大问题](https://www.kaggle.com/wiki/LogLoss)，下面的表达式就是二分类问题的 LogLoss 定义。



还有一个值得一提的地方就是测试集的文件名不是按 1, 2, 3 这样排的，而是按下面的顺序排列的：

['test/1.jpg',  
 'test/10.jpg',  
 'test/100.jpg',  
 'test/1000.jpg',  
 'test/10000.jpg',  
 'test/10001.jpg',  
 'test/10002.jpg',  
 'test/10003.jpg',  
 ......

因此我们需要对每个文件名进行处理，然后赋值到 df 里，最后导出为 csv 文件。

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  
8 9 0.005  
9 10 0.005

# 总结

我们可以从上图中看到，模型对于前十个样本都给出了很肯定的预测，提交到 kaggle 以后，得分也是很棒，0.04141，在全球排名中可以排到20/1314。我们如果要继续优化模型表现，可以使用更棒的预训练模型来导出特征向量，或者对预训练模型进行微调（fine-tune），或者进行数据增强（data augmentation）等。

参考链接：[面向小数据集构建图像分类模型](http://keras-cn.readthedocs.io/en/latest/blog/image_classification_using_very_little_data/)