

# 机器学习纳米学位

## 毕业项目—猫狗大战

---

fan yaohua

2018 年 4 月 24 日

## I. 问题的定义

---

### 项目概述

人类能轻易地识别照片中的物体，比如照片中是一只猫还是还是一只狗。

但计算机却很难识别，这涉及到计算机视觉领域。因为计算机底层只认得 0 和 1，对于图片来说，它只能区分一个个像素的 RGB 值。它难以整体的“感觉”到图像到底是什么。

猫狗大战是 [kaggle.com](https://www.kaggle.com/c/dogs-vs-cats) 在 2013 年举办的一场竞赛，即通过计算机识别一张图片是猫还是狗，它同时提供了 25000 张标注好的图片以供算法训练。随着近几年机器学习的发展，特别是计算硬件性能及大数据量级的提高，现在深度学习方法识别猫狗准确率非常高。

此项目基于深度学习和卷积神经网络尝试识别猫狗。

### 问题陈述

问题：使用深度学习方法识别一张图片是猫还是狗。

输入：一张彩色图片

输出：是猫还是狗

在这个项目中，该问题是一个二分类问题。作者将构建一个卷积神经网络，使它读入图片并输出图片是狗的概率[0,1]

## 评价指标

评估标准为 *LogLoss*，使用 kaggle 官方的二分类 *LogLoss* 公式：

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

其中

- $n$  是测试集中的图像数量
- $\hat{y}_i$  是图像是狗的预测概率
- $y_i$ ：如果图像是狗，则为 1；如果是猫，则为 0
- $\log()$  是自然对数  $e$

较小的 log loss 更好

## II. 分析

---

### 数据的探索

项目数据集可以从 [kaggle](#) 上下载。

此训练集共有 25000 张 jpg 图片，猫狗各 12500 张，通过文件名区分，图片尺寸不定大小不定。测试集共有 12500 张 jpg 图片，没有区分是猫还是狗。

图片场景有：单独出现，多个出现，有人类入镜等。例如：



cat.247.jpg



cat.81.jpg



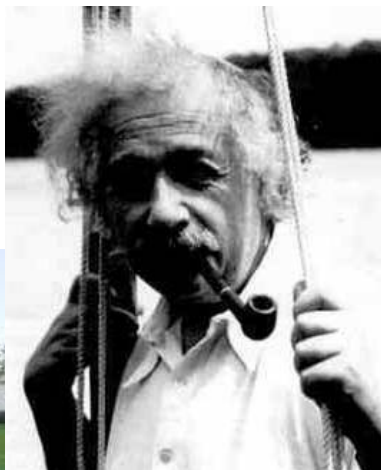
cat.1102.jpg

图 1 不同图片场景

还有极少的异常值。例如：



dog.1043.jpg



dog.1773.jpg

图 2 异常图片

由于神经网络容错能力较强，对于这极少部分异常值，例如图 2。后续可单独处理。

由于图片大小不一致，这里将会采用 Keras 的 `ImageDataGenerator` 函数进行统一的预处理，生成批次的带实时数据增益的张量图像数据。

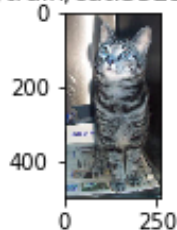
另外需要将数据集划分出训练集和验证集，比例定为 8:2。

## 探索性可视化

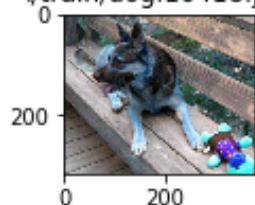
number of dogs: 12500

number of cats: 12500

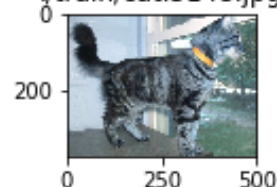
./train/cat.3525.jpg



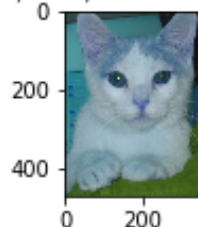
./train/dog.10418.jpg



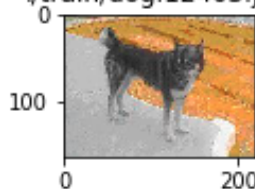
./train/cat.5148.jpg



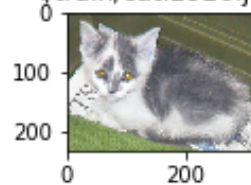
./train/cat.2395.jpg



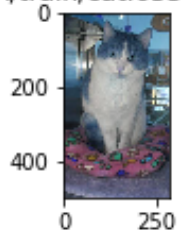
./train/dog.12403.jpg



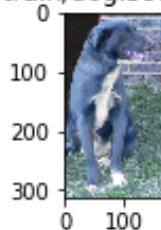
./train/cat.2926.jpg



./train/cat.633.jpg



./train/dog.5933.jpg



./train/dog.8218.jpg

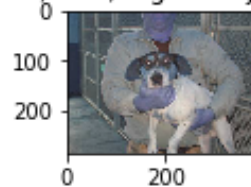


图 3 训练集随机图像

number of dogs and cats: 12500

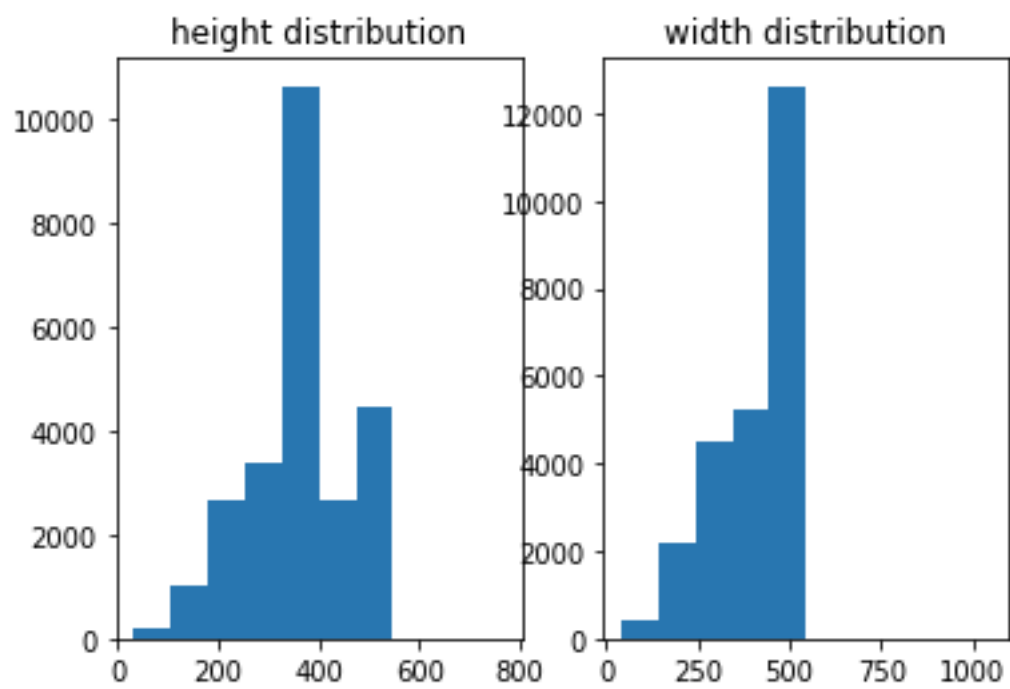


图 4 测试集随机图像

从图 3 图 4，我们能看出一些特点：

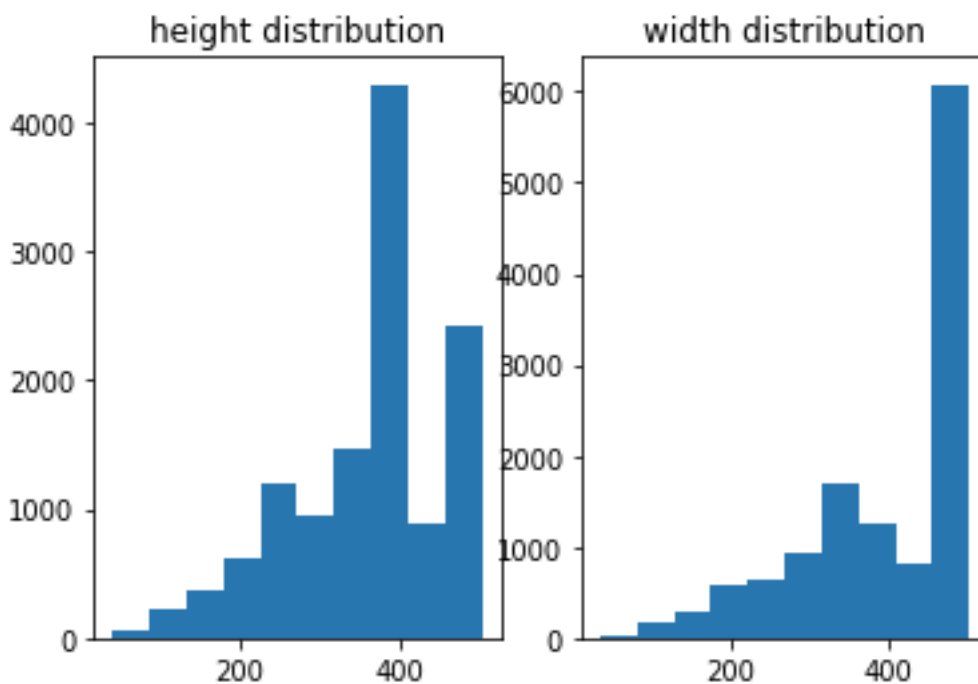
1. 图片一般拍摄自生活场景，内容为单只猫/狗为主，主要拍摄全身或头部特写，一般占据图像 1/2 左右
2. 图片偶有跟人的合影
3. 图片的光照条件和拍摄效果不同，有过曝，也有欠曝，也有拍摄模糊对焦不清的

下面为数据维度分布



训练集：  
median of height: 374.0  
median of width: 447.0

图 5 训练集数据维度

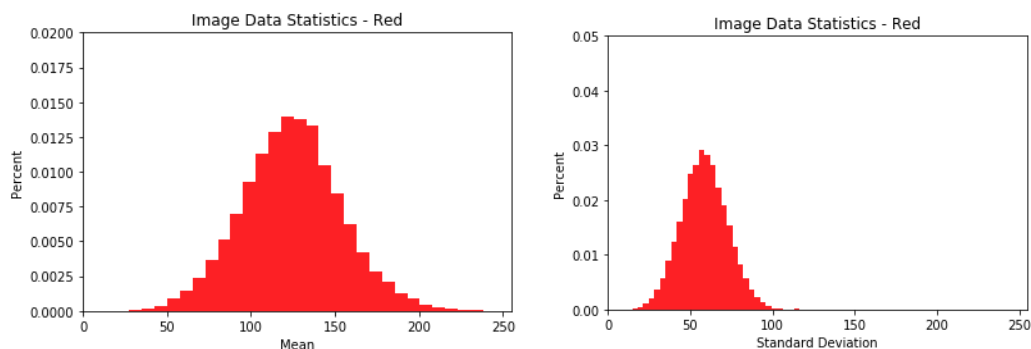


测试集：  
median of height: 374.0  
median of width: 447.0

图 6 测试集数据维度

根据图 5 图 6 统计，可得图片高宽比相对比较集中，在训练模型时可直接把图像拉伸至 1:1 作为输入

图片数据的要素除了尺寸还有像素中各个颜色通道的分布。如果训练数据的颜色分布不是均匀的或者是正态分布的，那么说明这个训练数据本身是有偏向的。



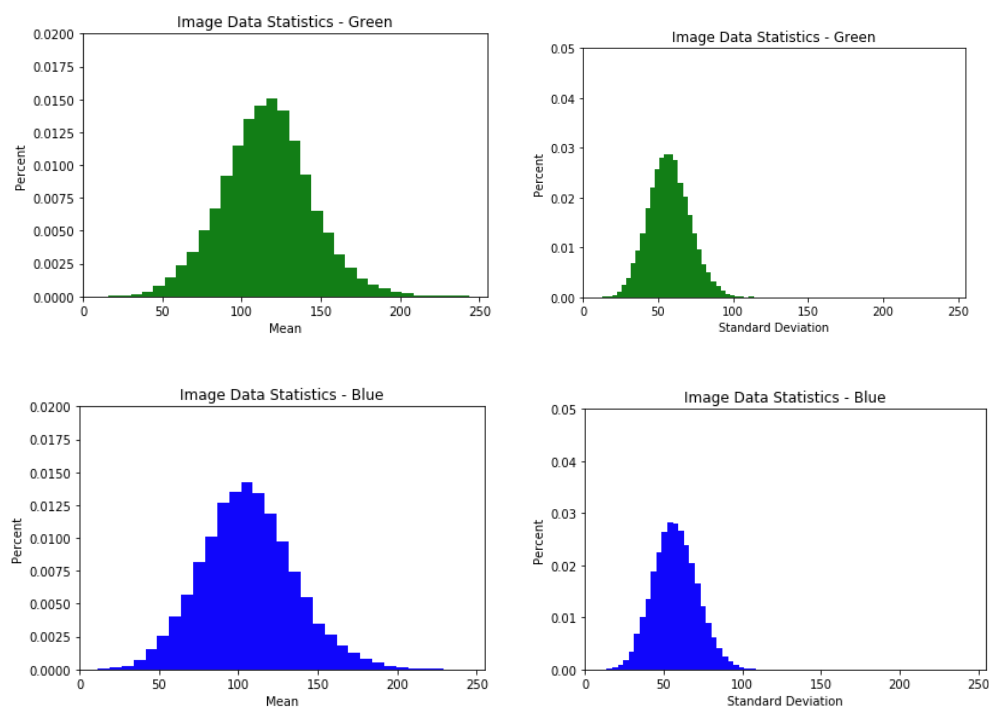


图 7 训练数据不同颜色通道的统计信息

为了了解训练数据中各个颜色通道中平均值的分布，在图 7 中给出了训练数据中每个图片各个颜色通道的统计分布柱状图(Histogram)。可以看出各个颜色通道的分布都类似于正态分布。红色和绿色分布的峰值大约位于 120 附近，而蓝色最大值在 110 附近。虽然不是完全的正态分布，但整体上看各个训练样本的颜色平均值偏差也不是很大。因此就颜色来说，该训练数据集没有在颜色上有特别的偏向。

## 算法和技术

猫狗识别是一个二元分类的问题。伴随着数据集的增加和计算能力的提升，特别是 GPU 计算能力的爆发性提升，使用卷积神经网络的深度学习便很好的解决了图片识别的问题。本次的项目中将采用 tensorflow 和 keras 来搭建学习模型，快速简单易上手。

CNN(Convolutional Neural Network 卷积神经网络) 是当前最强大的深度学习神经网络之一，特别是在图像识别应用上已经取得了惊人的表现。CNN 的设计原型最早起源于 LeNet5[1]，当时主要用于文本识别，后经人们不断对网络在结构



和性能上的优化形成了今天我们广泛使用的 CNN。

CNN 在图像识别上的优势主要体现为：

- 可以缩减计算过程中的权重 (weights) 数量；
- 在物体识别上能够抵抗轻微的扭曲，形变等干扰；
- 具有自动学习，特征归纳的特性；
- 对物体的识别不受该物体在图片中位置变动的影响；

一个经典的 CNN 结构主要有以下几种层组成

### 卷积层 (Convolutional Layer)

卷积层主要对输入 input volume 进行卷积操作，同时根据参数 stride 的设置，确定卷积核每两次卷积操作直接滑动的距离。卷积核 (kernel) 类似一个图像滤波器 (filter)，每一个卷积核用一个矩阵表示，将其在图片上按照间隔大小滑动处理一遍得到的结果，即为卷积层的 activation map (或 feature map)。

对于卷积操作，从高维视角来看，每一个 filter 代表一个 feature identifier，比如直线边缘特征代表，曲线特征代表等。当使用不同的 filter 进行卷积操作后，得到的 activation map 相当于对应的特征检测结果。因此 filter 越多，输出层得到的输入层的信息越多。

### ReLU(Rectified Linear Units) Layers

在每一个卷积层之后，一般会紧接着使用一个非线性层。该非线性层的主要目的是在系统中引入非线性特征。最常见的 CNN 网络中使用 ReLU 层，其具有比 tanh 和 sigmoid 函数更好的速度与效率，ReLU 层主要是对于 input 的所有值应用函数  $f(x) = \max(0, x)$ ，也就是说该层可以使所有的 negative activations 为 0，大大地减少储存空间。

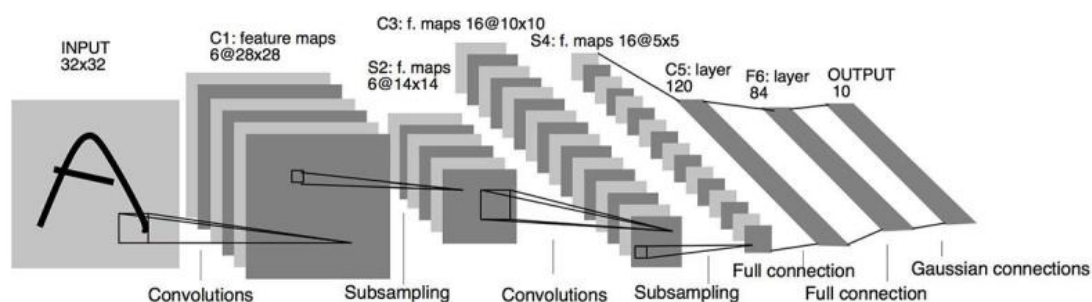
### 池化层 (Pooling Layers)

在 ReLU 层后一般为 pooling 层，即降采样层，最常见的是 maxpooling 方法，例如选择滤波器大小为  $2 \times 2$ ，stride (滑动间隔) 与滤波器长度一致，是比较常见的结果。Pooling 层对输入矩阵的一块区域整体进行操作，只在该区域保留一个

数值，而不同的 pooling 方法决定该数值的计算方法，maxpooling 层对于一个区域输出最大值，同时还有 average pooling 和 L2-norm pooling 等方式。

## 全连接层 (Fully Connected Layer)

全连接层一般位于 CNN 网络的最后部分，该层的输入可以是卷积层，ReLU 层，Pooling 层的结果，输出一个 N 维向量，N 即为要识别的类别数。例如在 softmax 方法下，结果向量是 [0 0.1 0.1 0.75 0 0 0 0 0.05]，则说明他有 10% 的概率是第一类，10% 的概率是第二类，75% 的概率是第三类，5% 的概率是第九类。下图为 LeNet-5 结构，其最后即为几个全连接层，使最终输出为 N 维向量。



## Dropout Layers

由大牛 Hinton 提出的 Dropout Layers，是为了防止过度拟合问题出现，在训练时专门加入的层。该层在 forward 过程中随机 drop out 一系列的激活值，将其值改成 0，减少过度拟合。该层使用下，使得网络在随机去掉一些激活值后依然能够有正确的分类等结果，即对不同有更好的适应性。

本项目中分别使用 Xception[2]、InceptionResNetV2[3]、inceptionV3[3]等模型。inceptionV3 的输入尺寸是 299\*299，不同于 VGG 的 244\*244，同时 inceptionV3 提出了卷积分解，使用 N\*1 的卷积级联既加速了计算又增加了网络深度，可以降低参数量，减轻过拟合，增加网络非线性的表达能力。因为小尺寸的滤波器可以提高网络的深度，进而让网络学习更加复杂的特征。将 Inception 模块和残差链接结合提出的 InceptionResNet 可以使得训练收敛更快，精度更高。Xception 是由 InceptionV3 的演化而来，和 InceptionV3 相比，Xception 的参数量有所下降，准确率也更高，在 Xception 中加入的类似 ResNet 的残差连接机制也显著加快

了 Xception 的收敛过程并获得了显著更高的准确率。

权重初始化的目的是为了尽量避免随着网络层数的加深, 而出现的梯度消失或者梯度爆炸的情况。目前常用的初始化方法有随机初始化(random initialization)、Xavier initialization、He initialization[4]。由于随机初始化的弊端是一旦随机分布选择不当, 就会导致网络优化陷入困境;而 Xavier initialization 可以解决随机初始化的问题, Xavier 初始化是保持输入和输出的方差一致, 这样就避免了所有输出值都趋向于 0。但是它对于非线性函数不具备普适性。He initialization 可以解决非线性初始化的问题, 要保持 variance 不变, 只需要在 Xavier 的基础上再除以 2。

常用的优化器有随机梯度下降 SGD 及其优化算法 Adagrad、Adadelata、Adam 等。SGD 每次只随机选择一个样本来更新模型参数, 因此每次的学习是非常快速的, 并且可以进行在线更新。但是 SGD 最大的缺点在于每次更新可能并不会按照正确的方向进行, 因此可以带来优化波动(扰动)。不过从另一个方面来看, SGD 所带来的波动有个好处就是, 对于类似盆地区域 (即很多局部极小值点) 那么这个波动的特点可能会使得优化的方向从当前的局部极小值点跳到另一个更好的局部极小值点, 这样便可能对于非凸函数, 最终收敛于一个较好的局部极值点, 甚至全局极值点。由于波动, 因此会使得迭代次数 (学习次数) 增多, 即收敛速度变慢。不过最终其会和全量梯度下降算法一样, 具有相同的收敛性, 即凸函数收敛于全局极值点, 非凸损失函数收敛于局部极值点。

Adagrad 是对学习率进行了一个约束, 自适应地为各个参数分配不同学习率的算法, 适合处理稀疏梯度。缺点是其学习率是单调递减的, 训练后期学习率非常小, 需要手工设置一个全局的初始学习率等。Adadelata 是对 Adagrad 的扩展, 能对学习率进行自适应约束, 又进行了计算上的简化, 训练时的加速效果不错, 时间也很快, 有效地克服 Adagrad 学习率收敛至零的缺点。通过比较, Adadelata 算法的表现效果通常不错, 学习率自适应优化, 不用手动调节。最后使用 Dropout 来防止过拟合, 因为随机的丢弃一部分隐藏节点既加快了计算又减弱了神经元节点间的联合适应性, 增强了泛化能力。

# 基准模型

使用 ResNet50[5]的迁移学习模型作为基准模型。  
下图为使用基准模型在 kaggle 上的跑出的分数截图

1 submissions for fanyaohua		Sort by	Most recent
All   Successful   Selected			
Submission and Description		Public Score	Use for Final Score
<a href="#">pred.csv</a> a few seconds ago by fanyaohua resnet50		0.07635	<input type="checkbox"/>

图 8 基准模型在 kaggle 的得分为 0.07635

基准阈值为 kaggle 排行榜前 10%，也就是第 131/1314 名，也就是在 Public Leaderboard 上的 logloss 要低于 0.06127。

## III. 方法

### 数据预处理

```
fyh@fyh-ubuntu: ~/ai
fyh@fyh-ubuntu:~/ai$ ls train | head
cat.0.jpg
cat.10000.jpg
cat.10001.jpg
cat.10002.jpg
cat.10003.jpg
cat.10004.jpg
cat.10005.jpg
cat.10006.jpg
cat.10007.jpg
cat.10008.jpg
fyh@fyh-ubuntu:~/ai$ ls test | head
10000.jpg
10001.jpg
10002.jpg
10003.jpg
10004.jpg
10005.jpg
10006.jpg
10007.jpg
10008.jpg
10009.jpg
fyh@fyh-ubuntu:~/ai$
```

图 9 数据集目录及文件结构

图 9 是 kaggle 提供的数据集中 train 和 test 文件夹的文件概览。

我们将使用 Keras 提供的 ImageDataGenerator 函数进行图片预处理。

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 ImageDataGenerator 的 flow\_from\_directory 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。

```

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')

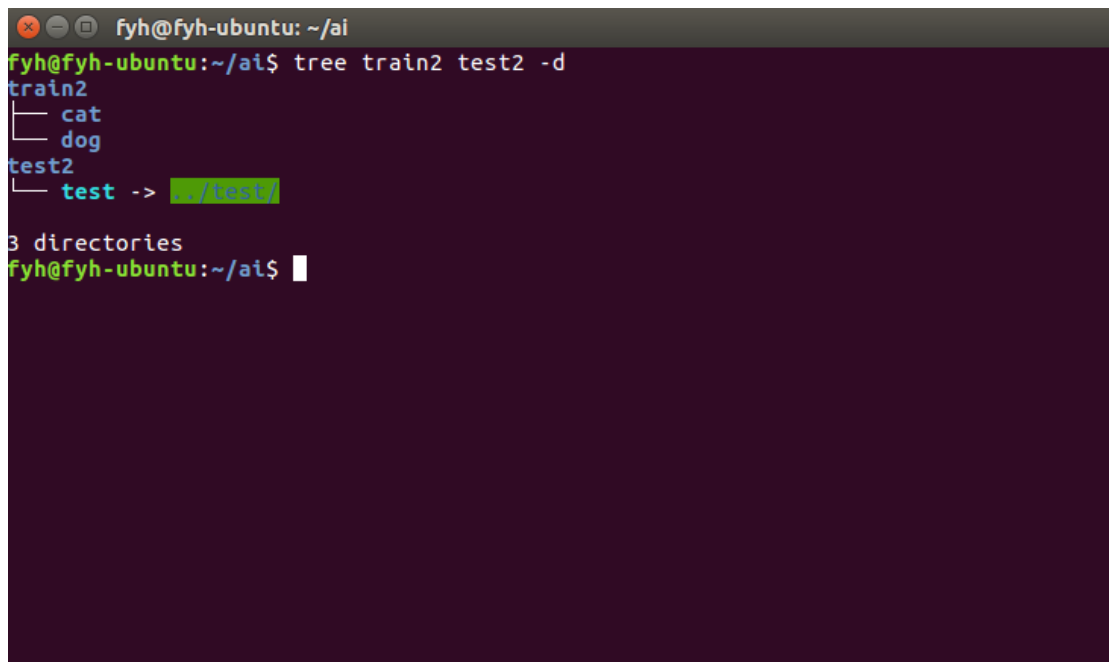
rmrf_mkdir('test2')
os.symlink('../test/', 'test2/test')

for filename in train_cat:
    os.symlink('../../train/'+filename, 'train2/cat/'+filename)

for filename in train_dog:
    os.symlink('../../train/'+filename, 'train2/dog/'+filename)

```

图 10 数据预处理代码



```

fyh@fyh-ubuntu: ~/ai
fyh@fyh-ubuntu:~/ai$ tree train2 test2 -d
train2
├── cat
└── dog
test2
└── test -> ../test/

3 directories
fyh@fyh-ubuntu:~/ai$

```

图 11 处理之后的文件夹结构

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。

## 执行过程

### 1. 导出特征向量

使用 Keras 的预训练模型提取特征，本项目将要进行迁移的候选网络模型包括:ResNet50、Xception、InceptionV3、DenseNet201[6]、InceptionResNetV2。提取不同的网络输出的特征向量后进行保存，以便后续训练。

如下图 12，为了复用代码，我们定义了一个 `write_gap` 函数，接收 3 个参数：输入模型，输入图片的大小，以及预处理函数，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, InceptionV3，DenseNet201 和 InceptionResNetV2。通过这些模型导出的特征向量，可以高度概括一张图片有哪些内容。

```
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)
    test = model.predict_generator(test_generator)
    with h5py.File("gap_%.h5"%MODEL.__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))
```

```
Found 25000 images belonging to 2 classes.  
Found 12500 images belonging to 1 classes.
```

```
write_gap(InceptionV3, (299, 299), inception_v3.preprocess_input)
```

```
Found 25000 images belonging to 2 classes.  
Found 12500 images belonging to 1 classes.
```

```
write_gap(Xception, (299, 299), xception.preprocess_input)
```

```
Found 25000 images belonging to 2 classes.  
Found 12500 images belonging to 1 classes.
```

```
write_gap(DenseNet201, (224, 224), densenet.preprocess_input)
```

```
Found 25000 images belonging to 2 classes.  
Found 12500 images belonging to 1 classes.
```

```
write_gap(InceptionResNetV2, (299, 299), inception_resnet_v2.preprocess_input)
```

```
Found 25000 images belonging to 2 classes.  
Found 12500 images belonging to 1 classes.
```

图 12 导出特征向量

## 2. 载入特征向量

载入上一步生成的特征向量，合成一条特征

在这里，选用 ResNet50，Xception 和 InceptionV3 进行模型融合。另外 X 和 y 需要打乱（使用 shuffle 函数），不然之后我们设置 validation\_split 的时候会出问题。这是因为这里有个陷阱是，程序是先执行 validation\_split，再执行 shuffle 的，所以会出现这种情况：假如你的训练集是有序的，比方说正样本在前负样本在后，又设置了 validation\_split，那么你的验证集中很可能将全部是负样本。同样的，这个东西不会有任何错误报出来，因为 Keras 不可能知道你的数据有没有经过 shuffle，保险起见如果你的数据是没 shuffle 过的，最好手动 shuffle 一下



```

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)

```

图 13 载入特征向量

### 3. 构建模型

调用 Keras 的 API 构建模型

```

from keras.models import *
from keras.layers import *

input_tensor = Input(X_train.shape[1:])
x = input_tensor
x = Dropout(0.5)(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(input_tensor, x)

model.compile(optimizer='adadelta',
              loss='binary_crossentropy',
              metrics=['accuracy'])

```

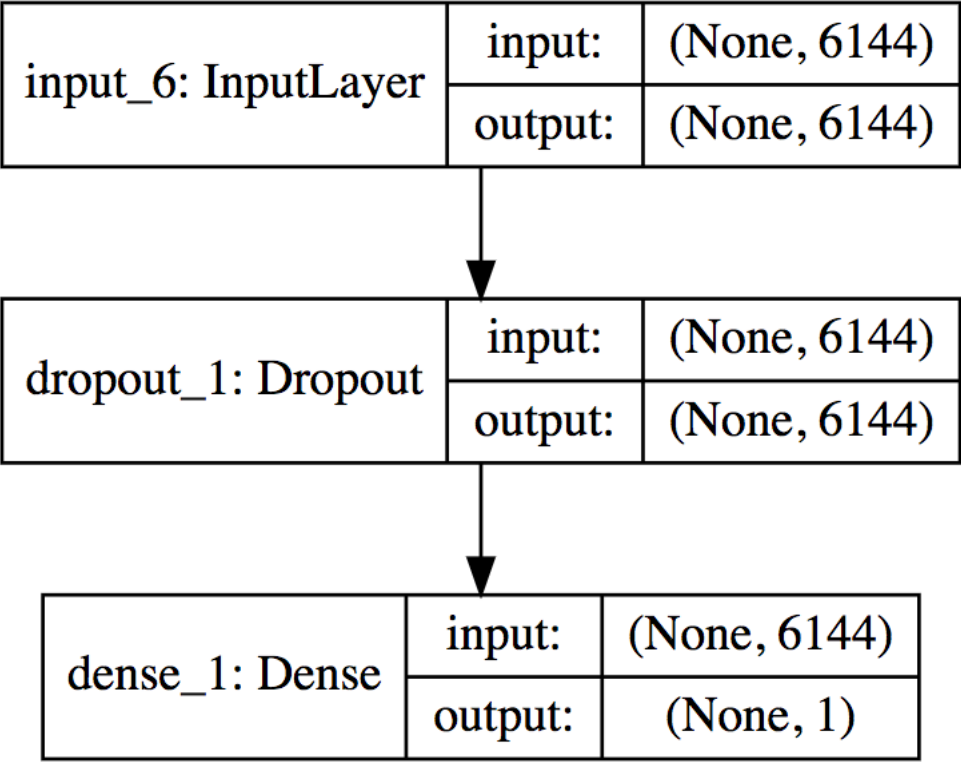


图 14 构建模型

4. 训练模型

分割训练集验证集， 然后进行训练。

validation\_split 用于在没有提供验证集的时候， 按一定比例从训练集中取出一部分作为验证集。这里设置为先前提到训练集 8:验证集 2 分割比例， 即 0.2

```
In [11]: model.fit(X_train, y_train, batch_size=128, epochs=8, validation_split=0.2)

Train on 20000 samples, validate on 5000 samples
Epoch 1/8
20000/20000 [=====] - 7s 336us/step - loss: 0.0810 - acc: 0.9722 - val_loss: 0.0185 - val_acc: 0.9956
Epoch 2/8
20000/20000 [=====] - 1s 49us/step - loss: 0.0235 - acc: 0.9932 - val_loss: 0.0144 - val_acc: 0.9950
Epoch 3/8
20000/20000 [=====] - 1s 48us/step - loss: 0.0178 - acc: 0.9949 - val_loss: 0.0144 - val_acc: 0.9946
Epoch 4/8
20000/20000 [=====] - 1s 49us/step - loss: 0.0162 - acc: 0.9950 - val_loss: 0.0111 - val_acc: 0.9970
Epoch 5/8
20000/20000 [=====] - 1s 49us/step - loss: 0.0141 - acc: 0.9955 - val_loss: 0.0097 - val_acc: 0.9968
Epoch 6/8
20000/20000 [=====] - 1s 51us/step - loss: 0.0141 - acc: 0.9956 - val_loss: 0.0094 - val_acc: 0.9966
Epoch 7/8
20000/20000 [=====] - 1s 50us/step - loss: 0.0120 - acc: 0.9961 - val_loss: 0.0101 - val_acc: 0.9962
Epoch 8/8
20000/20000 [=====] - 1s 49us/step - loss: 0.0118 - acc: 0.9963 - val_loss: 0.0106 - val_acc: 0.9960
```

图 15 训练模型

## 5. 预测测试集

对测试集进行预测，导出 csv，然后上传到 kaggle 相关页面查看得分

```
In [6]: y_pred = model.predict(X_test, verbose=1)
y_pred = y_pred.clip(min=0.005, max=0.995)

12500/12500 [=====] - 0s 21us/step
```

```
In [7]: import pandas as pd
from keras.preprocessing.image import *

df = pd.read_csv("sample_submission.csv")

image_size = (224, 224)
gen = ImageDataGenerator()
test_generator = gen.flow_from_directory("test2", image_size, shuffle=False,
                                         batch_size=16, class_mode=None)

for i, fname in enumerate(test_generator.filesnames):
    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)
```

Found 12500 images belonging to 1 classes.

/home/fyh/anaconda3/envs/tf/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py  
cated and will be removed in a future release. Please use .at[] or .iat[] acc

```
Out[7]:
```

	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

图 16 预测测试集

如图 16，这里我们对结果分值进行了 clip 操作，把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](#)，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。

pred.csv	0.03891
an hour ago by fanyaohua	
ResNet50, Xception, InceptionV3 merge	

图 17 提交 kaggle 查看得分

最终的预测结果提交到 kaggle，如图 17，分数为 0.03891，排名 14 左右，成绩还行，但主要是前人经验，还有提升的空间。

## 完善

为了进一步提升分类效果，作者准备重新选择模型进行融合。

### 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 18 keras 提供的预训练模型

作者选择 Top-1 及 Top-5 排行均前三的模型进行融合: InceptionResNetV2, Xception, InceptionV3。

pred.csv  
an hour ago by fanyaohua

0.03647

InceptionResNetV2, Xception, InceptionV3 merge

图 19 kaggle 得分

该模型的最终测试数据得分为 0.03647。排名已升至第七 (top 0.5%)，比之前的结果又有了提升。

## IV. 结果

### 模型的评价与验证

采用的是三个模型(InceptionResNetV2、Xception、InceptionV3)融合提取特征的方法作为最终的项目方案，提取特征后就可以训练模型了，最终提交的成成绩是 0.03647。

各模型训练结果：

Model	Logloss	Ranking
ResNet50	0.07635	131
ResNet50, Xception, InceptionV3 融合	0.03891	14
InceptionResNetV2, Xception, InceptionV3 融合	0.03647	7

## 合理性分析

经过不断的测试，可以看到多模型融合的效果是最好的。三种模型 (InceptionResNetV2、 Xception、 InceptionV3) 融合的训练效果好于其他模型组合，也更好于单一基准模型。因为这三种模型的准确率都比较高，默认输入图片的大小都是  $299 \times 299$ ，所以将它们综合到一起，可以很好的获取图片的信息。多层特征融合操作时可直接将不同层的网络特征级联,不同层的特征富含的语义信息可以相互补充。

## V. 项目结论

---

### 结果可视化



图 20 部分测试样本的预测结果

图 20 给出了部分测试样本的预测结果。可以看出，最终模型对于这些样本的预测结果基本是正确的。从这些结果可以看出，该模型对于猫狗的识别能力是非常强的，即使图片中出现了很多其他物体，如人体（No.11）、笼子(No.5)等等，都能做出很好的分辨。

综上所述，作者得到的模型是可信的。

## 对项目的思考

整个项目的操作步骤大致分为:数据预处理、使用预训练的模型提取特征向量、构建模型并训练、使用测试集来预测结果并提交。

预测结果要有好的表现，其中最重要的是提取特征向量。对于特征向量的提取，直接使用了预训练好的模型，既能节省时间又能得到较好的结果。通过不同的模型组合来训练，三种模型(InceptionResNetV2、Xception、InceptionV3)的效果是最佳的。

## 需要作出的改进

首先，正如前文提到的，作者在项目中并没有对训练数据进行清洗，在此假设了所有训练数据都是正确的。然而事实并非如此，之前在数据的探索中提到了训练数据中实际上有一定量错误的图片，既不是猫也不是狗。

尽管未经过数据清洗，作者仍然获得了相当好的分类模型。这再次证明了所构造模型的鲁棒性。当然，如果能将全部错误的训练数据从数据集中去除，相信能够达到更好的分类效果。

其次，在候选融合模型选择过程中，作者并没有做太多的尝试。只试了 Top 前三的模型融合，如果精心挑选组合，想必能获得更好的结果。

再次，没有尝试 Fine-tuning 技术。如果对选定的模型进行 Fine-tuning，想必还能提高最终得分。

最后，由于时间进度原因，作者没有完成模型的 web 部署，即一个用户上传照片识别猫狗的 web 应用。在完成论文后，将着手开发。



## 参考文献

- [1] LeNet5. <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>
- [2] Xception: Deep Learning with Depthwise Separable Convolutions.  
<https://arxiv.org/abs/1610.02357>
- [3] Rethinking the Inception Architecture for Computer Vision.  
<http://arxiv.org/abs/1512.00567>
- [4] He et.al "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification" ICCV 2015
- [5] Deep Residual Learning for Image Recognition.  
<https://arxiv.org/abs/1512.03385>
- [6] DenseNet. <https://github.com/liuzhuang13/DenseNet>

## 参考资料

- [https://github.com/ypwhs/dogs\\_vs\\_cats](https://github.com/ypwhs/dogs_vs_cats)
- [https://github.com/fountainhead-gg/dogs\\_vs\\_cats](https://github.com/fountainhead-gg/dogs_vs_cats)
- [https://github.com/juncas/udacity\\_p7\\_cat\\_vs\\_dog](https://github.com/juncas/udacity_p7_cat_vs_dog)
- [https://github.com/zhuoli91/Dogs\\_vs\\_Cats](https://github.com/zhuoli91/Dogs_vs_Cats)
- [https://github.com/tymcmf/dogs\\_cats](https://github.com/tymcmf/dogs_cats)
- <https://blog.csdn.net/u012905422/article/details/52463324>