

# 机器学习工程师纳米学位毕业项目

## 猫狗大战

王烨

2018 年 8 月 30 日

## 目录

- 目录
- 1.定义
  - 1.1 项目概述
  - 1.2 问题陈述
  - 1.3 评价指标
- 2.分析
  - 2.1 数据探索
  - 2.2 算法和技术
    - 2.2.1 深度神经网络
    - 2.2.2 卷积神经网络 (CNN)
  - 2.3 基准指标
  - 2.4 技术实现
- 3.具体方法
  - 3.1 迁移学习
    - 3.1 模型选择
  - 3.2 实现
    - 3.2.1 数据预处理
    - 3.2.2 模型的训练
    - 3.2.3 参数调优
- 4.结果](#4结果)
  - 4.1 模型评价与验证
  - 4.2 结果分析
    - 4.2.1 模型结果
    - 4.2.2 最终成绩
- 5.结论
- 6.不足
- 参考文献

# 1.定义

## 1.1 项目概述

猫狗大战（Dog vs Cat）是Kaggle举办的一个娱乐型的竞技比赛。在这个比赛中需要编写计算机算法来实现分类图像中是猫还是狗。Kaggle举办这个比赛的原因是，原来在很多服务器网站上为了防止机器人注册而使用了基于动物识别的验证码，这个验证是利用人能够很容易的区分猫狗，而机器不能，但是在人工智能发展之后，机器对猫狗的识别率已经大大提高了。本比赛是为了挑战计算机对猫狗的识别率。

本项目是使用卷积神经网络（Convolutional Neural Network，CNN），来对图像进行识别。

卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。

项目选择的数据集是 Kaggle 竞赛提供的数据，训练集包括 12500 张被标记为猫的图片 和 12500 张被标记为狗的图片，测试集包括 12500 张未标记图片。对于每一张测试集中的图 像，模型需要预测出是狗图像的概率（1 代表狗，0 代表猫）。

## 1.2 问题陈述

1. 首先，这是一个分类问题。
2. 项目提供的数据是来源某收留流浪猫狗网站的真实的照片，背景复杂，分辨率不同，还有一些异常图片混在其中。

根据以上的因素，作者将选用卷积神经网络，来分类猫狗图片。

## 1.3 评价指标

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

n 是测试集中的图像数量

$\hat{y}_i$  是图像是狗的预测概率

$y_i$  图像如果是狗为1，如果是猫为 0

$\log()$ 是以e为底的自然对数

越小的log loss越好

# 2.分析

## 2.1 数据探索

从Kaggle下载的数据打开后观察，是由两个文件夹train和test组成，其中\_train\_文件夹是训练数据其文件的命名格式为 物种.序号.jpg，文件夹中猫狗各占12500张，一共25000张照片，\_test\_文件夹是测试数据只用序号.jpg

来命名，共有数据12500条。

```
1 .
2 |— test
3 |   |— 1.jpg
4 |   |— 2.jpg
5 |   |— ...
6 |   |— 9999.jpg
7 |— train
8 |   |— cat.0.jpg
9 |   |— cat.1.jpg
10 |   |— cat.2.jpg
11 |   |— ...
12 |   |— cat.12499.jpg
13 |   |— dog.0.jpg
14 |   |— dog.1.jpg
15 |   |— dog.2.jpg
16 |   |— ...
17 |   |— dog.12499.jpg
```

首先，简单观察一下训练数据有哪些类型，照片来自真实的场景，可以看到场景丰富。有一只猫，关在笼子后的猫，多只猫，人和猫，猫和狗 同时现出现的；  
还有少量的异常值。



## 2.2 算法和技术

基于前面的数据探索与分析可知，猫狗识别问题其实是一个分类问题。输入一张图片得到这张图片是猫还是狗的概率。

分类算法就我们知道的有 决策树、支持向量机、逻辑回归、贝叶斯、神经网络 等方法。基于经验，神经网络 尤其适合图像的识别。

2.2.1 深度神经网络

深度神经网络（Deep Neural Networks, DNN）是一种判别模型，可以使用反向传播算法进行训练。权重更新可以使用下式进行随机梯度下降法求解：

$$\Delta w_{ij}(t+1) = \Delta w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}}$$

其中， $\eta$  为学习率， $C$ 为代价函数。

这一函数的选择与学习的类型（例如监督学习、无监督学习、增强学习）以及激活函数相关。

例如，为了在一个多分类问题上进行监督学习，通常的选择是使用ReLU作为激活函数，而使用交叉熵作为代价函数。

Softmax函数定义为  $p_j = \frac{\exp(x_j)}{\sum_k \exp(x_k)}$ ，其中  $p_j$  代表类别  $j$  的概率，而  $x_j$ 和 $x_k$  分别代表对单元  $j$  和  $k$  的输入。交叉熵定义为  $C = - \sum_j d_j \log(p_j)$ ，其中  $d_j$  代表输出单元  $j$  的目标概率， $p_j$  代表应用了激活函数后对单元  $j$  的概率输出。

2.2.2 卷积神经网络（CNN）

卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。

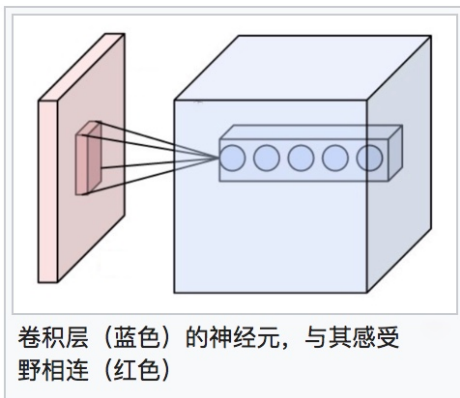
卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。这一模型也可以使用反向传播算法进行训练。相比较其他深度、前馈神经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构。

卷积神经网络一般结构如下：

1. 卷积层（Convolutional layer）

卷积层是CNN的核心构建块。图层的参数由一组可学习的过滤器（或内核）组成，这些过滤器具有较小的感知区域，但延伸到输入卷的整个深度。在前向传递期间，每个滤波器在输入体积的宽度和高度上卷积，计算滤波器的条目与输入之间的点积，并产生该滤波器的二维激活图。结果，网络学习当在输入中的某个空间位置处检测到某种特定类型的特征时激活的过滤器。

沿深度维度堆叠所有滤波器的激活图形成卷积层的完整输出体积。因此，输出体积中的每个条目也可以被解释为神经元的输出，其观察输入中的小区域并与同一激活图中的神经元共享参数。



2. 线性整流层 (Rectified Linear Units layer, ReLU layer)

线性整流层使用 $f(x) = \max(0, x)$ 作为这一层神经的 激活函数。它可以增强判定函数和整个神经网络的非线性特性，而本身并不会改变卷积层。

事实上，其他的一些函数也可以用于增强网络的非线性特性，如双曲正切函数 $f(x) = \tanh(x)$ 或者Sigmoid函数 $f(x) = (1 + e^{-x})^{-1}$ 。

相比其它函数来说，ReLU函数更受青睐，这是因为它可以将神经网络的训练速度提升数倍，而并不会对模型的泛化准确度造成显著影响。

3. 池化层(Pooling Layer)

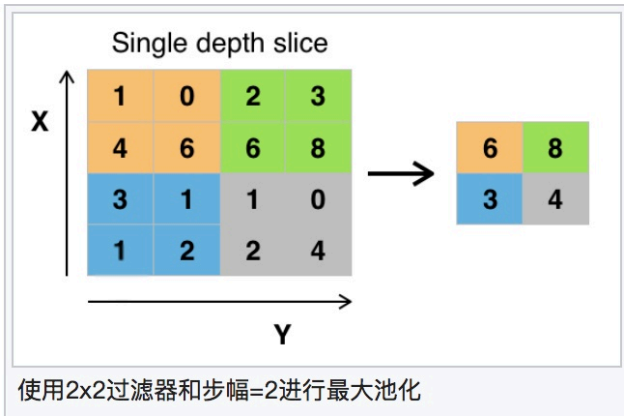
每隔2个元素进行的2x2最大池化

池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。直觉上，这种机制能够有效地原因在于，在发现一个特征之后，它的精确位置远不及它和其他特征的相对位置的关系重要。池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。通常来说，CNN的卷积层之间都会周期性地插入池化层。

池化层通常会分别作用于每个输入的特征并减小其大小。目前最常用形式的池化层是每隔2个元素从图像划分出 $2 \times 2$ 的区块，然后对每个区块中的4个数取最大值。这将会减少75%的数据量。

除了最大池化之外，池化层也可以使用其他池化函数，例如“平均池化”甚至“L2-范数池化”等。过去，平均池化的使用曾经较为广泛，但是最近由于最大池化在实践中的表现更好，平均池化已经不太常用。

由于池化层过快地减少了数据的大小，目前文献中的趋势是使用较小的池化滤镜，甚至不再使用池化层。

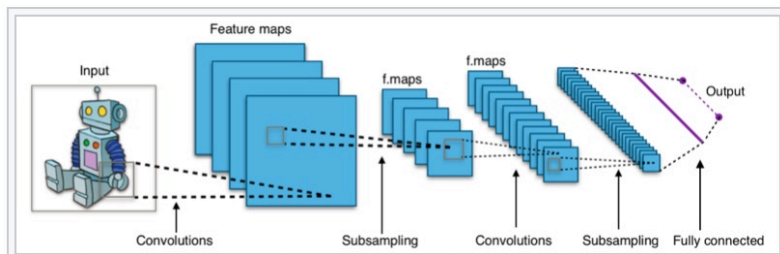


4. 完全连接的层 (Fully connected layer)

最后，在几个卷积和最大池化层之后，神经网络中的高级推理是通过完全连接的层完成的。完全连接层中的神经元与前一层中的所有激活具有连接，如常规神经网络中所见。因此，可以通过矩阵乘法后跟偏置偏移来计算它们的激活。

5. 损失函数层 (loss layer)

损失函数层用于决定训练过程如何来“惩罚”网络的预测结果和真实结果之间的差异，它通常是网络的最后一层。各种不同的损失函数适用于不同类型的任务。例如，Softmax交叉熵损失函数常常被用于在K个类别中选出一个，而Sigmoid交叉熵损失函数常常用于多个独立的二分类问题。欧几里德损失函数常常用于结果取值范围为任意实数的问题。



典型的CNN架构

## 2.3 基准指标

此次的要求是在Kaggle上取得前10%的排名，截止目前2018/09/06为止，目前有4441名提交的成绩，按得分排序找到第444名，得分如下：

443	388025	Yujie	2017-02-13 14:24:33	0.05602
444	368495	Bojan Tunguz	2016-09-15 12:26:08	0.05603

也就是说这次项目的得分要小于等于 0.05603

## 2.4 技术实现

1. 根据项目的性质，作者决定采用开源框架 *TensorFlow* 和 *Keras*。
2. 由于使用CNN处理大分辨率全彩色图片所需的计算量非常的大，项目使用了GPU进行计算加速。在安装 *TensorFlow* 和 *Keras* 的同时还需要安装 *CUDA* 和 *cuDNN*。

基于以上两点，作者直接申请了AWS的GPU计算型 p3.2xlarge 并直接加载了 Deep Learning AMI (Ubuntu) Version 13.0 - ami-09994a343440ce0cd 镜像。

# 3.具体方法

## 3.1 迁移学习

### 3.1 模型选择

由于项目需要在kaggle排名前10%，自己创建并训练一个神经网络需要细致的模型设计并进行大量的调参优化，需要大量的算力和时间。

好消息是，Keras提供了在ImageNet上预训练过的用于图像分类的模型 Xception、VGG16、VGG19、ResNet50、InceptionV3、InceptionResNetV2、MobileNet、DenseNet、NASNet。阅读相关的说明文档后，作者决定选取其中的Xception 和 InceptionV3模型。

InceptionV3 和 Xception 这两个模型的图片预处理方式为将数据从 (0, 255) 缩放到 (-1, 1) 区间内。

## 3.2 实现

### 3.2.1 数据预处理

因为图片数据大小不一、场景复杂，所以决定用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。



ImageDataGenerator.flow\_from\_directory函数要求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分类，为了减少磁盘的消耗量，作者决定采用建立软链接的方式将图片分到不到目录中。

```
1 def sort_img(dir_path):
2     '''判断图片是猫还是狗'''
3     train_filenames = os.listdir(dir_path)
4     train_cat = list(filter(lambda x:x.split('.')[0] == 'cat', train_filenames))
5     train_dog = list(filter(lambda x:x.split('.')[0] == 'dog', train_filenames))
6     return train_cat , train_dog
7
8 def create_dir(dir_path):
9     '''建立指定的目录'''
10    remove_dir(dir_path)
11    cmd = "mkdir -p " + dir_path
12    os.system(cmd)
13
14 def remove_dir(dir_path):
15     cmd = "rm -rf " + dir_path
16     os.system(cmd)
17
18 def create_symlink(ori_path,tar_apth):
19     cmd = "ln -s " + ori_path + ' ' + tar_apth
20     os.system(cmd)
21
22 def process_train_data(data_list, org_path, tar_path):
23     '''建立分猫狗分类目录及软链接'''
24     create_dir(tar_path)
25     for img in tqdm(data_list):
26         create_symlink(org_path + img,tar_path + img)
```

```
1 cat, dog = sort_img(ORG_TRAIN_DIR) #分别建立猫狗数据
2
3 process_train_data(cat,ORG_TRAIN_DIR,TRAIN_CAT_DIR)
4 process_train_data(dog,ORG_TRAIN_DIR,TRAIN_DOG_DIR)
5
6 create_dir(TEST_DIR)
7 create_symlink(ORG_TEST_DIR,TEST_DIR)
```

```
100% |████████████████████| 12500/12500 [01:00<00:00, 205.43it/s]
100% |████████████████████| 12500/12500 [01:00<00:00, 205.16it/s]
```

处理后目录结构如下：

```
(tensorflow_p36) ubuntu@ip-172-31-8-110:~/capstone/dog_vs_cat$ tree input -d
input
├── test
│   └── test -> /home/ubuntu/capstone/dog_vs_cat/all/test/ ③
└── train
    ├── cat ①
    └── dog ②
```

- ① 12500张猫
- ② 12500张狗
- ③ 验证集

### 3.2.2 模型的训练

采用策略是先将Xception 和 InceptionV3模型输出的特征向量保存下来，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，即使不使用AWS在普通的电脑上也可以进行，因为要保存两个模型的向量，而他们的代

码几乎是相同的所以为了代码的复用，写到一个保存函数里。

## I. 保存向量函数

```
1 def write_gap(MODEL, image_size, lambda_func=None):
2     '''保存向量'''
3     h5_full_path = H5PY_PATH + MODEL.__name__ + '.h5'
4     width = image_size[0]
5     height = image_size[1]
6     input_tensor = Input((height, width, 3))
7     x = input_tensor
8     if lambda_func:
9         x = Lambda(lambda_func)(x)
10
11     gen = ImageDataGenerator()
12     train_generator = gen.flow_from_directory(TRAIN_DIR, image_size, shuffle=False,
13                                             batch_size=16)
14     test_generator = gen.flow_from_directory(TEST_DIR, image_size, shuffle=False,
15                                             batch_size=16, class_mode=None)
16
17     base_model = MODEL(input_tensor=x, weights='imagenet', include_top=False)
18     plot_model(base_model, to_file=MODEL.__name__+'.png')
19     model = Model(base_model.input, GlobalAveragePooling2D()(base_model.output))
20     model.summary()
21
22     train = model.predict_generator(train_generator)
23     test = model.predict_generator(test_generator)
24     remove_dir(h5_full_path)
25
26     with h5py.File(h5_full_path) as h:
27         h.create_dataset("train", data=train)
28         h.create_dataset("test", data=test)
29         h.create_dataset("label", data=train_generator.classes)
```

代码17~19行，构建基本模型，并在后面追加一层平均池化。因为在后面代码中模型会使用全连接层，因此在这里添加一层AveragePooling,因为在多分类任务的时候，假设图中出现了一只狗和一个人。狗很大，人很小。我们应该输出狗而不是人。

代码26~29行，为保存向量到磁盘文件中。

## II. 使用InceptionV3预训练模型并保存

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

## III. 使用Xception预训练模型并保存

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

## IV. 导入两个模型权重向量文件



```

1 def load_eigenvector(path_list=[]):
2     x_train = []
3     x_test = []
4     for file_path in path_list:
5         with h5py.File(file_path, 'r') as fp:
6             x_train.append(np.array(fp['train']))
7             x_test.append(np.array(fp['test']))
8             y_train = np.array(fp['label'])
9
10    x_train = np.concatenate(x_train, axis=1)
11    x_test = np.concatenate(x_test, axis=1)
12    x_train, y_train = shuffle(x_train, y_train)
13    return x_train, y_train, x_test

```

## V. 增加dropout层和全连接层并编译模型

```

1 def full_connect_layer(x_train, dropout_rate = 0.2):
2     input_tensor = Input(x_train.shape[1:])
3     model = Model(input_tensor, Dropout(dropout_rate)(input_tensor))
4     model = Model(model.input, Dense(1, activation = 'sigmoid')(model.output))
5     # 编译模型
6     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
7     model.summary()
8     return model

```

代码3行，增加dropout层。

代码4行，增加全连接层，并将Sigmoid函数做为激活函数。

代码6行，编译模型,优化器选择 *adam* 算法。

*Adam* 是一种可以替代传统随机梯度下降过程的一阶优化算法，它能基于训练数据迭代地更新神经网络权重。Adam 最开始是由 OpenAI 的 Diederik Kingma 和多伦多大学的 Jimmy Ba 在提交到 2015 年 ICLR 论文 (Adam: A Method for Stochastic Optimization) 中提出的。

## VI. 训练模型

```

1 def model_fit(model,x_train,y_train,chk_point_file,epochs = 10,batch_size = 128):
2     remove_dir(chk_point_file)
3     history_train = model.fit(x_train, y_train, validation_split = 0.2,epochs = epochs,
4                               batch_size = batch_size, verbose=1,
5                               callbacks=[ModelCheckpoint(filepath=chk_point_file, verbose=1,
6                                                         save_best_only=True)])
7     return history_train

```

训练模型同时对训练集进行分割，`validation_split = 0.2` 20%作为验证集，80%作为训练集；训练结果通过回调函数保存成文件。

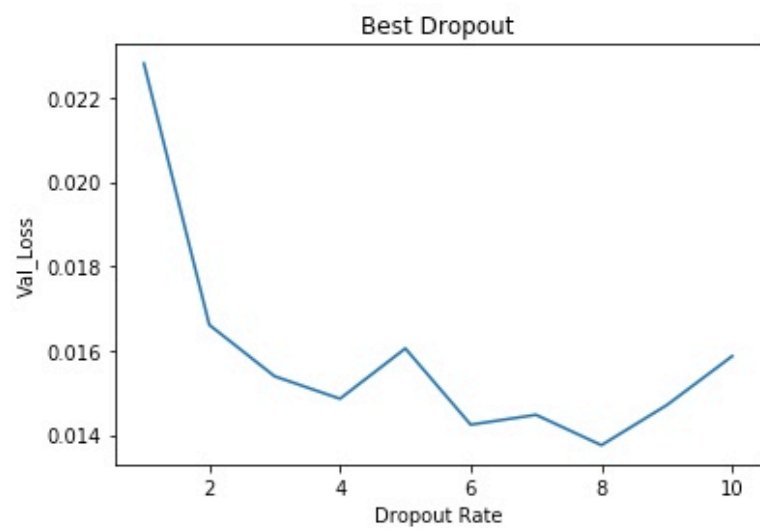
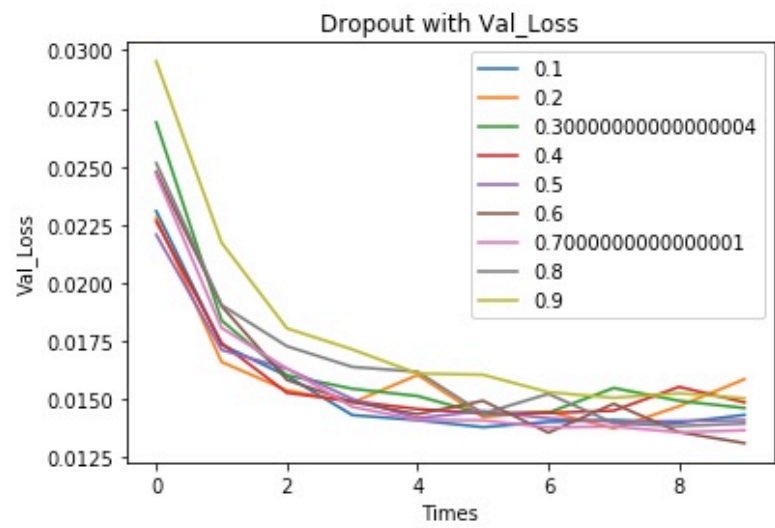
### 3.2.3 参数调优

通过对不同dropout丢弃率，从0.1~0.9尝试

```
1 epochs = 10
2 dropout_result= []
3 for rate in np.arange(0.1, 1.0, 0.1):
4     model = full_connect_layer(x_train, dropout_rate=rate)
5     train_result = model.fit(x_train, y_train, batch_size=128, epochs=epochs,
6                             validation_split=0.2)
7     dropout_result.append((rate,train_result.history['val_loss'],train_result.history['val_acc']
8                             ))
```

	Avg	Max	Min
dropout rate			
0.1	0.015517	0.023084	0.013810
0.2	0.015878	0.022812	0.013757
0.3	0.016578	0.026895	0.014370
0.4	0.015862	0.022648	0.014410
0.5	0.015565	0.022074	0.014042
0.6	0.015893	0.024777	0.013124
0.7	0.015686	0.024650	0.013588
0.8	0.016551	0.025151	0.013856
0.9	0.017931	0.029517	0.015050

从图上可看出dropout为0.1时成绩最好



在不同dropout率下的 *LogLoss* 图

## 4.结果

### 4.1 模型评价与验证

```

Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [=====] - 4s 193us/step - loss: 0.0815 - acc: 0.9764 - val_loss: 0.0234 - val_acc: 0.9946

Epoch 00001: val_loss improved from inf to 0.02337, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 2/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0227 - acc: 0.9940 - val_loss: 0.0167 - val_acc: 0.9948

Epoch 00002: val_loss improved from 0.02337 to 0.01674, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 3/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0178 - acc: 0.9945 - val_loss: 0.0138 - val_acc: 0.9952

Epoch 00003: val_loss improved from 0.01674 to 0.01377, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 4/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0154 - acc: 0.9953 - val_loss: 0.0134 - val_acc: 0.9952

Epoch 00004: val_loss improved from 0.01377 to 0.01342, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 5/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0141 - acc: 0.9958 - val_loss: 0.0126 - val_acc: 0.9952

Epoch 00005: val_loss improved from 0.01342 to 0.01259, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 6/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0132 - acc: 0.9958 - val_loss: 0.0119 - val_acc: 0.9956

Epoch 00006: val_loss improved from 0.01259 to 0.01185, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 7/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0122 - acc: 0.9960 - val_loss: 0.0120 - val_acc: 0.9956

Epoch 00007: val_loss did not improve from 0.01185
Epoch 8/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0110 - acc: 0.9967 - val_loss: 0.0144 - val_acc: 0.9944

Epoch 00008: val_loss did not improve from 0.01185
Epoch 9/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0101 - acc: 0.9965 - val_loss: 0.0117 - val_acc: 0.9956

Epoch 00009: val_loss improved from 0.01185 to 0.01168, saving model to /home/ubuntu/capstone/dog_vs_cat/saved_models/Xception_IncV3.best.h5
Epoch 10/10
20000/20000 [=====] - 1s 36us/step - loss: 0.0092 - acc: 0.9968 - val_loss: 0.0125 - val_acc: 0.9956

Epoch 00010: val_loss did not improve from 0.01168

```

本次采用了两个模型混合特征组合策略，从数据中来看，混合特征的方法效果非常的好。从上图可以看出在10次训练中准确率迅速提升并保持在0.995以上，而 $LogLoss$ 损失值仅在0.0092附近；训练集和验证集误差较接近，说明没有过拟合，同时通过预训练，大大降低了训练的时间，因为只更新了最后全连接层的权重，大大加快了训练速度。

## 4.2 结果分析

### 4.2.1 模型结果

在相同的dropout = 0.2的情况下

网络模型	得分	排名
InceptionV3	1.12465	3721
Xception	0.04092	117

混合模型特征	0.03838	12
--------	---------	----








作者没有对单模型的dropout率进行优化，统一用了0.2，从结果来看Xception模型比InceptionV3要优秀，整整提高了96.3%。

混合模型特征得分是0.03838，比Xception提高了又提高了6.2%，说明采用混合策略是可以提高水平的。

对于混合模型，考虑到不同模型的能力不同，为不同的模型设置不同的权重，让能力强的输出的数据占有更大的比重，也是可以考虑的策略。

#### 4.2.2 最终成绩

因为作者并没有将数据集中的异常数据剔除，而且从最终的测试集得到的分数来看，模型是具有的健壮性。最终在kaggle的public leaderboard得到了12名的成绩。

Name		Submitted	Wait time	Execution time	Score
pred.csv		3 minutes ago	0 seconds	0 seconds	0.03838
Complete					
<a href="#">Jump to your position on the leaderboard</a>					
5	▲19	DeepBrain		0.03518	56 2y
6	▼3	lefant		0.03580	84 2y
7	▲41	matview		0.03778	40 2y
8	▲7	Bancroftway Systems [Andy ...		0.03804	41 2y
9	new	Arvinder Chopra		0.03805	5 2y
10	new	Ranjeeta		0.03807	5 2y
11	▼6	Adarsh Tadimari		0.03838	12 2y

## 5. 结论

CNN是一个非常强大而灵活的工具，特别是CNN这种开放的结构可以把不同的模型组合其中，围绕其分类及特征，可以有很多不同的组合和应用；利用CNN进行视觉分类还是十分强大的，不过需要强大的计算力，我想通过优化和改进算法应该可以继续减少所需的算力。通过这次项目，从理论到了实践，又从实践回到了理论；让作者对人工智能的理解进一大步，对CNN和keras的使用提高了很多。

通过一周不懈的努力，最终在kaggle的得分还是比较让作者欣慰的。

## 6. 不足

- 没有尝试使用Fine-tuning

- 没有异常数据做剔除的处理
- 没有尝试更多的组合
- 由于时间的限制，没有实现IOS的应用。有时间要实现这个功能

## 参考文献

[维基百科](#)

Jeff Delaney [CatdogNet - Keras Convnet Starter](#)

Sentdex [Full Classification Example with ConvNet](#)

ypwhs [猫狗大战](#)

[面向小数据集构建图像分类模型](#)

[Xception: Deep Learning with Depthwise Separable Convolutions](#)

[Keras中文文档](#)

[https://keras-cn-docs.readthedocs.io/zh\\_CN/latest/](https://keras-cn-docs.readthedocs.io/zh_CN/latest/)

[VGG in TensorFlow](#)