

机器学习工程师纳米学位毕业项目  
猫狗大战

蔡炜

2019 年 8 月 7 日

# 1 定义

## 1.1 项目概述

本项目出自于 kaggle 的一个娱乐型竞赛项目——Dogs vs Cats Redux: Kernels Edition，是一个计算机视觉领域中典型的图像分类问题，训练集包含了 25000 张猫狗图片，其中各有一半被标记为猫或狗，项目要求竞赛者训练一个模型来对测试集中的 12500 张猫狗图片进行分类识别。

项目模型的构建主要基于卷积神经网络（Convolutional Neural Networks, CNN），这是一类包含卷积数学计算且具有深度结构的神经网络<sup>[1]</sup>，是深度学习的代表性算法之一，广泛应用于计算机视觉领域。

经过学习 udacity 的机器学习课程，本人深刻体会到计算机视觉领域的乐趣，该领域也是人工智能的热点，在现实生活中有广泛应用。

## 1.2 问题陈述

训练集图片已打好标签，标签类别仅有两种，因此该项目属于监督学习（Supervised Learning）中的二分类问题。

不难发现，训练集中的图像画质参差不齐，猫狗种类繁多，图像背景复杂，这些因素都直接影响到模型的分类预测。此外，仔细观察后还能发现，其中有些图片根本不属于猫狗，这会误导我们的模型，因此我们首先要对训练集中的图片进行“清洗”，筛选掉这些“不友好”因素。

接着，我们可以使用一些知名的预训练模型如 Xception、ResNet、

Inception 等对图像进行特征提取，从而抽象出图像相邻像素点之间不同级别的组合信息，这会大大有助于模型的学习与训练。

最后，我们将提取出的图像特征输入到经过训练的模型中，经过分类函数得到分类预测结果。由于是二分类，使用 sigmoid 分类函数映射到(0, 1)区间即可，接近 0 代表分类结果是猫，接近 1 代表分类结果是狗。

### 1.3 评价指标

本项目的评价指标采用对数损失 (LogLoss)，也称交叉熵 (Cross Entropy) 损失，它起源于信息论，主要用于度量两个概率分布间的差异性信息。使用交叉熵作为损失函数有一个好处就是使用如 sigmoid 激活函数在梯度下降时能避免像使用均方误差等其它损失函数时存在学习速降低的问题，因为学习速率可以被输出的误差所控制<sup>[2]</sup>。对于二分类问题，LogLoss 的计算公式如下：

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

其中  $N$  为测试集样本数， $y_i$  代表第  $i$  个样本的真实标签，如果是狗则  $y_i = 1$ ，否则  $y_i = 0$ ， $\hat{y}_i$  表示模型预测第  $i$  个样本为狗的概率。

由 LogLoss 的计算公式可以看出，模型预测样本对应其真实标签的概率要尽可能大，即对于真实标签为猫样本，模型输出的概率值要尽可能接近 0，而对于真实标签为狗的样本，模型输出的概率值要尽可能接近 1。另外，在采用多模型整合时若采用投票或平均值的方式则不利于 LogLoss 减小。

## 2 分析

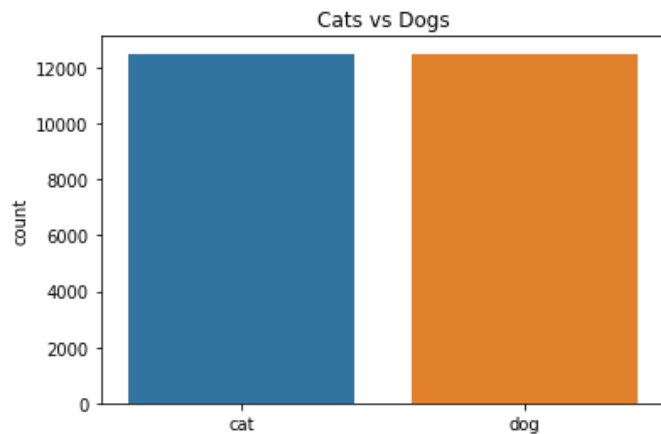
### 2.1 数据研究与可视化

本项目数据集由 kaggle 提供，分为训练集和测试集，分别包含 25000 张和 12500 张猫狗图片，其中，训练集的图片通过文件名称来标记其所属类别是猫还是狗。



训练集图片示例

可以看到，图片中猫狗姿态各异，背景复杂多样，这为我们模型的识别增加了一定难度。另外，根据图片文件名称来统计，训练集中猫狗图片各占一半，如下所示：



进一步探索，我们会发现，部分图片的标注错误，即猫的图片被标注为狗（或者相反），还有部分图片既不是猫也不是狗，或者在某些图片中，同时存在猫和狗，又或者图片中有人或者其它复杂背景，而猫狗在图片中所占比例太小，这样会导致模型在提取特征时主要学习到了非猫狗的其它因素的特征，因此我们需要对这些异常进行处理。



dog.4334.jpg (猫的图片被标注为狗)



cat.2159.jpg (图片中同时存在猫狗)



dog.2614.jpg (非猫狗图片)



dog.1194.jpg (非猫狗图片)



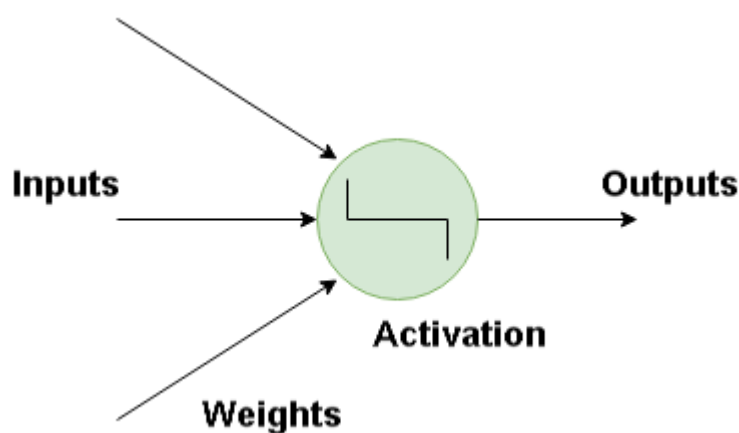
dog.2877.jpg (一只卡通猫被标注为狗)

## 2.2 算法和技术

### 2.2.1 深度学习与神经网络

深度学习(DL, Deep Learning)是机器学习(ML, Machine Learning)领域中一个研究方向，它使用包含复杂结构的多个处理层对数据进行高层抽象的算法，这些处理层被称为神经网络(Neural Networks) <sup>[3]</sup>。

神经网络是一组模仿生物大脑神经元结构而设计的算法，它有许多隐藏层共同组成，每层由许多计算单元构成，这些计算单元类似于大脑神经元，它们会对输入信号进行加权计算并且相互交互，而后被激活最终产生输出信号<sup>[4]</sup>。



神经元处理信号

在深度学习中，神经网络主要分为 3 层：

1) 输入层

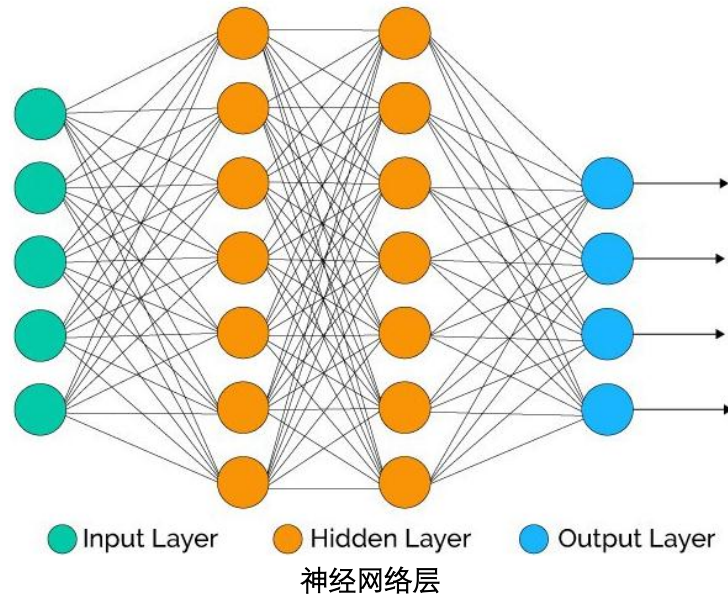
接收输入数据，神经元的数量通常对应于数据的特征数。

2) 隐层

负责对数据进行线性加权计算并激活（非线性计算），一张神经网络可以有多个隐层，每个隐层有由多个神经元组成。

3) 输出层

基于隐层的计算结果产生输出，作为预测结果。



可以看出，深度学习是学习数据的内在规律和表示层次，通过多层处理，将初始特征表示转化为高度抽象特征后利用模型完成学习任务，因此，通常将深度学习纳入表征学习（Representation Learning）<sup>[5]</sup> 范畴。

### 2.2.2 卷积神经网络

卷积神经网络（Convolutional Neural Networks, CNN）是一类包含卷积数学计算且具有深度结构的前馈神经网络（Feedforward Neural Networks），是深度学习的代表性算法之一。

在传统神经网络中，每层的每个神经元都与下层的每个神经元相连，这种连接关系称为全连接（Full Connected）。以处理图像数据为例，那么输入就是每个单位像素，假如一张黑白图像尺寸是  $10^3 \times 10^3$ ，则输入层需要  $10^6$  个结点，假如后面隐层有  $10^4$  个结点，那么需要的权重系数就达到了  $10^{10}$  个。若隐层的结点数量更多，那么计算量就会更大，而且参数过多还容易导致过拟合。卷积神经网络则是通过局部感知（local awareness）也称稀疏交互（sparse interactions）来解决

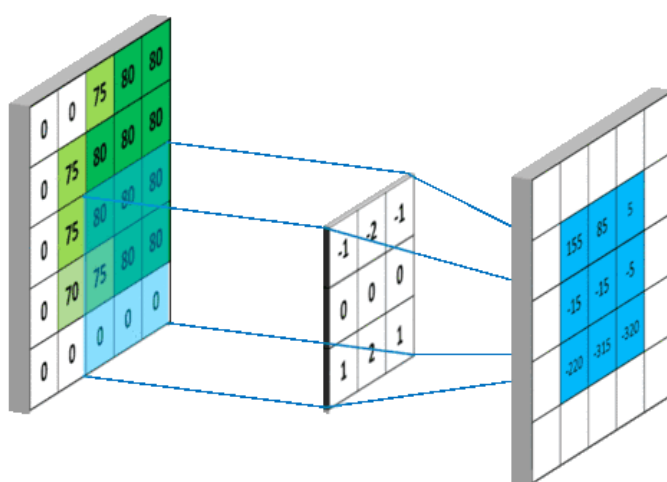


这一问题，也就是每个神经元只与上一层的部分神经元相连交互，获得局部信息，最终在高层将所有局部信息综合起来得到全局信息，这是受启发于生物视觉系统，对外界的认知从局部到全局<sup>[6]</sup>。

卷积神经网络的结构与普通神经网络结构相似，不同的是，其隐层通常包含卷积层（convolutional layer）和池化层（pooling layer）：

### 1) 卷积层

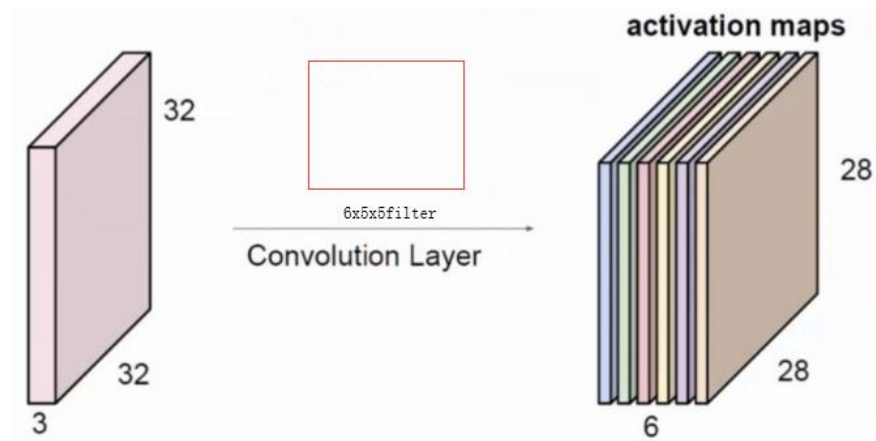
卷积层采用一个或多个滤波器（filter）也称卷积核（kernel）对图像的像素矩阵进行卷积运算，给一个卷积核设定大小尺寸和步长后，通过卷积核在像素矩阵上不断扫描移动来完成整个运算过程，每移动完一步便完成一次计算，对应到隐层的一个结点，从而将原始图像尺寸映射到另一个尺寸，而卷积核的数量则将图像通道数映射到另一个通道数。



5x5 原始图像尺寸被 3x3 尺寸卷积核映射到 3x3 尺寸

可以看出，经过相同卷积核局部感知的隐层结点的权重系数是相同的，这被称为权值共享（weight sharing）或参数共享（parameter sharing），这样设计是因为图片的底层特征与

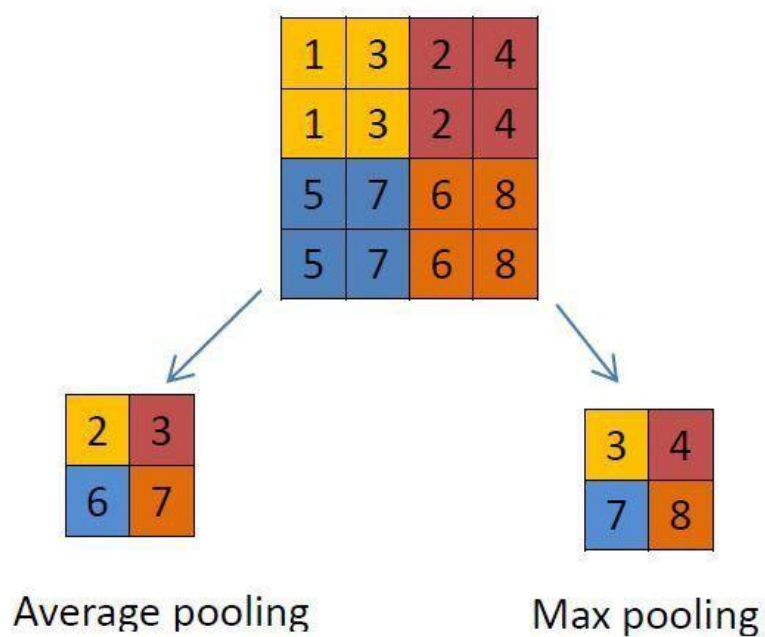
特征在图片中的位置无关，经过一个卷积核局部感知后获得的是一组特征映射，因此通常需要多个卷积核来获得多组特征，最终在上层将它们组合起来获得全局视野<sup>[7]</sup>。



32x32 尺寸的 3 通道图像被 6 个 5x5 尺寸的卷积核映射为 28x28x6 的特征图

## 2) 池化层

池化层主要起到降维作用，通常有最大池化 (max pooling) 和平均池化 (average pooling) 两种方式，最大池化是直接去 filter 区域内最大值作为映射输出，平均池化则是取平均值。



4x4 矩阵经过 2x2 filter 的两种池化方式得到的结果











### 2.2.3 技术实现

本项目实现采用开源机器学习库 Keras，其作为对 Tensorflow 框架的封装，使得开发过程更加便捷。此外，由于数据量和计算量较大，因此借助于亚马逊的 EC2（Amazon Elastic Compute Cloud，Amazon EC2）服务平台进行 GPU 加速计算。

从头开始构建一个 CNN 网络来训练并进行调参优化非常复杂耗时，而在 ImageNet 中，已有不少优秀的开源预训练模型，因此项目采用迁移学习(Transfer Learning)的方式进行模型构建与训练，从而简化训练过程和提高效果。预训练模型主要负责对图像进行特征提取，而后添加自定义的少数基层全连接层来对特征进行分类识别。

## 2.3 基准指标

本项目要求模型的 LogLoss 在 kaggle pubic leaderboard 中排入前 10%，该竞赛共有 1314 位参与者，因此需要排位在 131 名以前，即  $\text{LogLoss} < 0.06127$ 。

124	Chase		0.06026	6	3y
125	John Vial		0.06037	18	2y
126	tmuzap		0.06054	21	3y
127	icodingc		0.06068	12	3y
128	Wilson Sun		0.06082	23	2y
129	Jeremy Howard		0.06086	11	3y
130	RaviKiranK		0.06114	35	2y
131	Reziproke		0.06127	4	3y
132	mathieuzaradzki		0.06149	32	2y
133	Mouatez		0.06240	39	3y

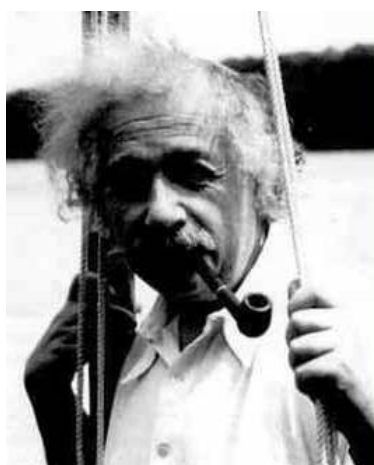
kaggle public leaderboard

## 3 方法

### 3.1 数据预处理

#### 3.1.1. 异常数据清理

在上述数据探索部分，我们已经剖析了 kaggle 提供的数据集中存在部分异常，因此，在模型利用这些数据进行训练之前，我们需要清理掉这些异常数据。



异常数据示例 dog.1773.jpg



异常数据示例 dog.9517.jpg

关键问题来了，训练集总共包含 25000 张图片，我们几乎不可能用肉眼一一分辨出这些异常数据，那么该怎么办呢？

众所周知，ImageNet 项目是一个用于视觉对象识别软件研究的大型

可视化数据库，其中超过 1400 万张图像被标注，其中恰恰包含了猫和狗，官方每年举办的大规模视觉识别挑战赛（ILSVRC）诞生了许多优秀的视觉识别模型，因此本项目采用的方法是使用已在 ImageNet 中预训练的模型对训练集中的图片进行分类识别，最终根据模型预测结果是否在 ImageNet 数据集的猫狗类别中来判断是否是异常数据。

具体地，本项目使用了 5 种预训练模型，分别是 ResNet50、Xception、InceptionV3、InceptionResNetV2 以及 NASNetLarge，将它们的预测结果取并集，并且加上 kaggle 讨论区中人们已被发现的异常数据，最终作为需要清理的异常数据。

```
def find_out_cheat_files(model, preprocess_func, decode, img_size):  
    """  
    use pretrained model(on imagenet) to find out images that may confuse my model in later training step.  
    """  
  
    cheat_files = []  
    for fname in fnames:  
        category = cats_code if fname.split('.')[0] == 'cat' else dogs_code  
        img_path = FOLDER_TRN / fname  
        img = image.load_img(img_path, target_size=img_size)  
        arr = image.img_to_array(img)  
        tensor = np.expand_dims(arr, axis=0)  
        tensor = preprocess_func(tensor)  
        preds = model.predict(tensor)  
        preds_decode, __, __ = zip(*decode(preds, top=30)[0])  
  
        if not np.intersect1d(preds_decode, category).size:  
            cheat_files.append(fname)  
  
    return cheat_files
```

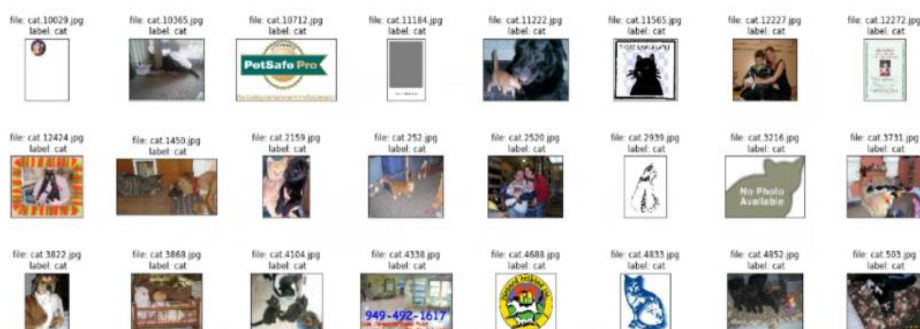
定义方法 使用预训练模型识别异常数据



```
# Cheat files aggregation
cheatfiles_all = np.union1d(cheat_files_by_models, arr_cheat)
print("From the above, there are {} cheat files in all, as below:".format(len(cheatfiles_all)))
pprint(cheatfiles_all)

From the above, there are 428 cheat files in all, as below:
array(['cat.10029.jpg', 'cat.10037.jpg', 'cat.10107.jpg', 'cat.10121.jpg',
      'cat.10209.jpg', 'cat.10220.jpg', 'cat.10266.jpg', 'cat.10270.jpg',
      'cat.10365.jpg', 'cat.10425.jpg', 'cat.10471.jpg', 'cat.10521.jpg',
      'cat.10532.jpg', 'cat.10536.jpg', 'cat.10539.jpg', 'cat.10579.jpg',
      'cat.10609.jpg', 'cat.10634.jpg', 'cat.10636.jpg', 'cat.10700.jpg',
      'cat.10712.jpg', 'cat.10743.jpg', 'cat.10807.jpg', 'cat.10863.jpg',
      'cat.10864.jpg', 'cat.10893.jpg', 'cat.10912.jpg', 'cat.10932.jpg',
      'cat.10946.jpg', 'cat.11018.jpg', 'cat.11039.jpg', 'cat.11062.jpg',
      'cat.11141.jpg', 'cat.11168.jpg', 'cat.11184.jpg', 'cat.11222.jpg',
      'cat.11231.jpg', 'cat.11255.jpg', 'cat.11256.jpg', 'cat.11281.jpg',
      'cat.11297.jpg', 'cat.1139.jpg', 'cat.11399.jpg', 'cat.114.jpg',
      'cat.11432.jpg', 'cat.11544.jpg', 'cat.11562.jpg', 'cat.11565.jpg',
      'cat.11607.jpg', 'cat.11608.jpg', 'cat.11634.jpg', 'cat.11661.jpg',
      'cat.11675.jpg', 'cat.11683.jpg', 'cat.11724.jpg', 'cat.11726.jpg',
      'cat.11777.jpg', 'cat.11823.jpg', 'cat.11870.jpg', 'cat.11879.jpg',
      'cat.11923.jpg', 'cat.11941.jpg', 'cat.11968.jpg', 'cat.11980.jpg',
      'cat.12126.jpg', 'cat.12182.jpg', 'cat.12189.jpg', 'cat.12219.jpg',
      'cat.12222.jpg', 'cat.12227.jpg', 'cat.12239.jpg', 'cat.12252.jpg',
```

综合模型预测和 kaggle 讨论区的结果作为最终需要清理的异常数据



部分异常数据展示

### 3.1.2. 数据增强 (Data Augmentation)

删除异常数据后，将测试集按标签分配比例进一步划分为测试集和验证集，验证集用于训练过程中交叉验证，从而评估每个周期训练结果的好坏。

接着，项目使用了数据增强，具体包括随机旋转的度数范围设置为 25、随机缩放范围设置为 0.2、剪切强度设置为 0.1、宽度和高度转换范围设置为 0.1 以及使用随机水平翻转，这增加数据样本的同时通常



能够提高模型的泛化能力，考虑到本项目使用的 4 种预训练模型对图像处理的默认尺寸不完全同，因此生成了 299x299 和 331x331 两种尺寸的生成器实例，并且批次大小分别为 16 和 32。

```
# Image size
SIZE_299 = (299, 299)
SIZE_331 = (331, 331)

def generator_flow(
    datagen, df, target_size, batch_size,
    directory=FOLDER_TRN, x_col='file', y_col='label',
    mode='binary', shuffle=True
):
    gen = datagen.flow_from_dataframe(
        df,
        directory=directory,
        x_col=x_col,
        y_col=y_col,
        target_size=target_size,
        class_mode=mode,
        batch_size=batch_size,
        shuffle=shuffle
    )
    return gen

datagen_trn = ImageDataGenerator(
    rotation_range=25,

    width_shift_range=.1,
    height_shift_range=.1,
    zoom_range=.2,
    shear_range=.1,
    horizontal_flip=True
)
```

### 数据增强

```
# Data generator of training data & validation data with different size
generator_trn_331 = generator_flow(datagen_trn, df_trn, SIZE_331, BATCH_SIZE_331)
generator_val_331 = generator_flow(datagen_val, df_val, SIZE_331, BATCH_SIZE_331)
generator_trn_299 = generator_flow(datagen_trn, df_trn, SIZE_299, BATCH_SIZE_299)
generator_val_299 = generator_flow(datagen_val, df_val, SIZE_299, BATCH_SIZE_299)
```

```
Found 18429 images belonging to 2 classes.
Found 6143 images belonging to 2 classes.
Found 18429 images belonging to 2 classes.
Found 6143 images belonging to 2 classes.
```

```
# Class indices
print(generator_trn_331.class_indices)
print(generator_trn_299.class_indices)
print(generator_val_331.class_indices)
print(generator_val_299.class_indices)
```

```
{'cat': 0, 'dog': 1}
{'cat': 0, 'dog': 1}
{'cat': 0, 'dog': 1}
{'cat': 0, 'dog': 1}
```

### 生成器实例及标签映射

### 3.2 特征提取

通常，特征提取是解决图像分类与识别问题的必要步骤，CNN 的卷积核恰是在其中发挥了重要作用，卷积层的各卷积核通过设置不同的权重能够提取出图像的边缘、纹理等甚至其它高层次的特征。

本项目使用了 4 种在 ImageNet 上预训练的模型，分别是 Xception、InceptionV3、NASNetLarge、InceptionResNetV2，将它们最后的全连接层去掉，权重设置为在 ImageNet 上训练的结果。先让图像分别经它们各自的预处理方法进行处理，然后分别对应输入到这些模型中，达到提取图像特征的效果。此外，还为它们都设置了平均池化，用于压缩提取的特征，防止过拟合。

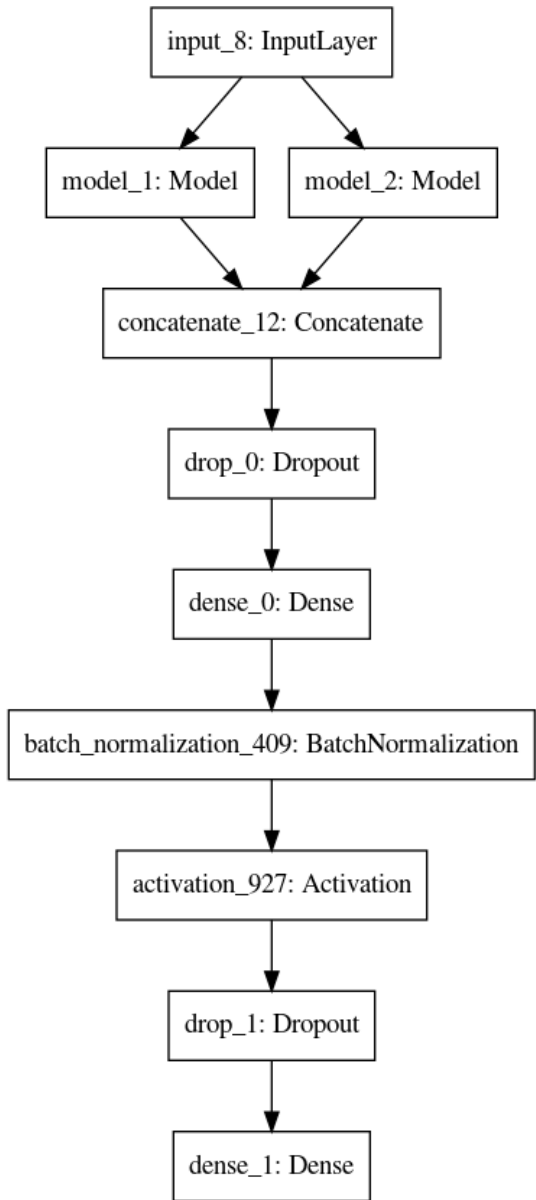
### 3.3 模型构建

将几种预训练模型提取出的特征进行融合，具体方法是使用 Keras 提供的 Concatenate 层，它可以将多个张量按照指定的维度进行拼接，默认是最后一个维度。

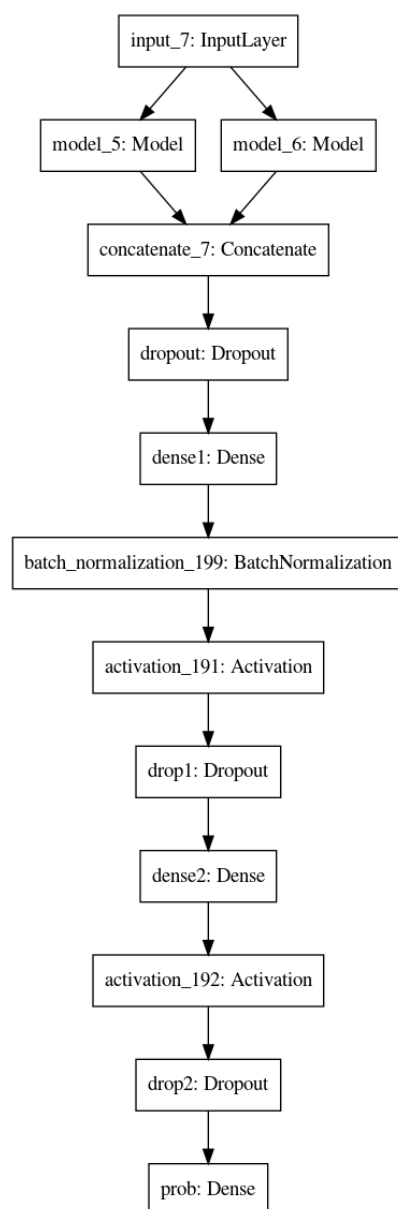
特征融合后，只需再使用少数几层全连接层和丢弃层，激活函数使用 ReLu，在反向传播求误差梯度时，其相对于 sigmoid 等其它激活函数计算量较小，在深层网络中也不容易出现梯度消失。此外，它会将部分输出映射为 0，为网络增加了稀疏因素，减少了参数相互依存的关系，一定程度上缓解了过拟合问题。最后一个全连接层使用 sigmoid 激活函数，它可以将输出映射到(0,1)的区间，从而作为二分类的结果。

由于本项目使用的 4 种预训练模型对图像的预处理方式以及默认输入尺寸不完全相同，其中 NASNetLarge 的默认输入尺寸为 331x331，而

Xception、InceptionV3 和 InceptionResNetV2 为 299x299, 本项目将 Xception 和 InceptionV3 这 2 种模型提取的特征进行融合, 而将 NASNetLarge 和 InceptionResNetV2 另外 2 种模型提取的特征进行融合, 接着分别对两个融合的特征添加少数几层网络, 最终构建出 2 种模型。



基于 InceptionResNetV2 和 NASNetLarge 构建的模型



基于 Xception 和 InceptionV3 构建的模型

### 3.4 模型训练

模型训练使用的损失函数为二分类交叉熵 `binary_crossentropy`，与该二分类问题匹配，度量为精度 `accuracy`，针对基于 Xception 和 InceptionV3 构建的模型，批次大小设置为 32，优化器使用了 RMSprop，参数为默认参数；而针对基于 InceptionResNetV2 和 NASNetLarge 构建的模型，则使用带动量的 SGD，可以加速收敛并且

抑制震荡，每次参数更新后学习率衰减值设为  $1e-3$ ，由于批次大小为 16 比较小，因此初始的学习率设置为 0.009，比默认的 0.01 小，避免陷入局部最优。

模型总共训练 10 个周期，训练批次大小 batch\_size 分别为 16 和 32 个样本，根据 val\_loss，训练过程会自动保存最优权重，此外，还设置了早期停止 EarlyStopping，在连续 3 个周期模型表现没有改进时便会停止训练。

```
Epoch 1/10
575/575 [=====] - 700s 1s/step - loss: 0.1087 - acc: 0.9682 - val_loss: 0.0674 - val_acc: 0.9910

Epoch 00001: val_loss improved from inf to 0.06740, saving model to model299.0823.1218.weights.best.hdf5
Epoch 2/10
575/575 [=====] - 673s 1s/step - loss: 0.0840 - acc: 0.9773 - val_loss: 0.0154 - val_acc: 0.9974

Epoch 00002: val_loss improved from 0.06740 to 0.01544, saving model to model299.0823.1218.weights.best.hdf5
Epoch 3/10
575/575 [=====] - 674s 1s/step - loss: 0.0736 - acc: 0.9793 - val_loss: 0.0225 - val_acc: 0.9967

Epoch 00003: val_loss did not improve from 0.01544
Epoch 4/10
575/575 [=====] - 674s 1s/step - loss: 0.0742 - acc: 0.9808 - val_loss: 0.0395 - val_acc: 0.9943

Epoch 00004: val_loss did not improve from 0.01544
Epoch 5/10
575/575 [=====] - 674s 1s/step - loss: 0.0709 - acc: 0.9812 - val_loss: 0.0113 - val_acc: 0.9977

Epoch 00005: val_loss improved from 0.01544 to 0.01134, saving model to model299.0823.1218.weights.best.hdf5
Epoch 6/10
575/575 [=====] - 674s 1s/step - loss: 0.0665 - acc: 0.9819 - val_loss: 0.0240 - val_acc: 0.9959

Epoch 00006: val_loss did not improve from 0.01134
Epoch 7/10
575/575 [=====] - 674s 1s/step - loss: 0.0603 - acc: 0.9837 - val_loss: 0.0330 - val_acc: 0.9931

Epoch 00007: val_loss did not improve from 0.01134
```

### 基于 Xception 和 InceptionV3 的模型训练

```
Epoch 00001: val_loss improved from inf to 0.02225, saving model to model331.weights.best.hdf5
Epoch 2/10
1151/1151 [=====] - 2585s 2s/step - loss: 0.0409 - acc: 0.9848 - val_loss: 0.0191 - val_acc: 0.9936

Epoch 00002: val_loss improved from 0.02225 to 0.01910, saving model to model331.weights.best.hdf5
Epoch 3/10
1151/1151 [=====] - 2582s 2s/step - loss: 0.0377 - acc: 0.9854 - val_loss: 0.0208 - val_acc: 0.9938

Epoch 00003: val_loss did not improve from 0.01910
Epoch 4/10
1151/1151 [=====] - 2580s 2s/step - loss: 0.0359 - acc: 0.9862 - val_loss: 0.0227 - val_acc: 0.9925

Epoch 00004: val_loss did not improve from 0.01910
Epoch 5/10
1151/1151 [=====] - 2581s 2s/step - loss: 0.0399 - acc: 0.9861 - val_loss: 0.0151 - val_acc: 0.9953

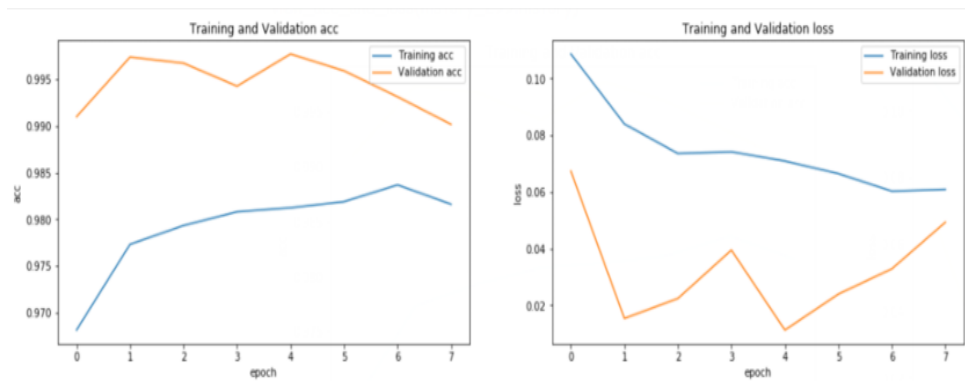
Epoch 00005: val_loss improved from 0.01910 to 0.01512, saving model to model331.weights.best.hdf5
Epoch 6/10
1151/1151 [=====] - 2581s 2s/step - loss: 0.0339 - acc: 0.9873 - val_loss: 0.0173 - val_acc: 0.9951

Epoch 00006: val_loss did not improve from 0.01512
Epoch 7/10
1151/1151 [=====] - 2581s 2s/step - loss: 0.0359 - acc: 0.9884 - val_loss: 0.0299 - val_acc: 0.9904

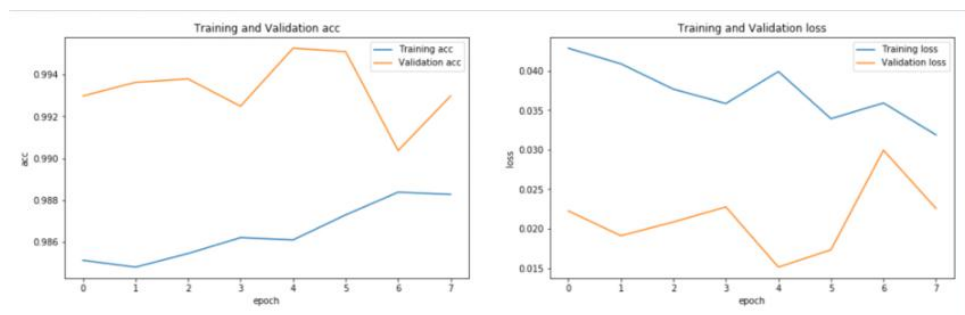
Epoch 00007: val_loss did not improve from 0.01512
Epoch 8/10
1151/1151 [=====] - 2580s 2s/step - loss: 0.0319 - acc: 0.9883 - val_loss: 0.0226 - val_acc: 0.9930

Epoch 00008: val_loss did not improve from 0.01512
```

### 基于 InceptionResNetV2 和 NASNetLarge 的模型训练



基于 Xception 和 InceptionV3 的模型训练过程的精确度和损失曲线



基于 InceptionResNetV2 和 NASNetLarge 的模型训练过程的精确度和损失曲线

## 4. 结果

### 4.1. 模型评价与验证

项目将 Xception 和 InceptionV3 进行融合，构建出输入图像尺寸是 299x299 的一个模型，以及将 InceptionResNetV2 和 NASNetLarge 进行融合，构建出输入图像尺寸是 331x331 的另一个模型，两个模型分别进行训练后对测试集进行预测，LogLoss 分别为 0.04597 和 0.04170。

---

<a href="#">submission_299_0823rmsprop.csv</a>	0.04597
21 hours ago by cwcai	
Xception&InceptionV3(rmsprop)	

---

**基于 Xception 和 InceptionV3 的模型在测试集上的表现**

---

<a href="#">submission_331.csv</a>	0.04170
10 days ago by cwcai	
NASNetLarge&InceptionResNetV2(SGD optimizer)	

---

**InceptionResNetV2 和 NASNetLarge 的模型在测试集上的表现**

以上结果均满足  $\text{LogLoss} < 0.06127$ ，符合项目通关要求。

### 4.2. 结果分析

从以上 LogLoss 来看，利用 NASNetLarge 和 InceptionResNetV2 进行融合的模型表现较好，其中 Xception 和 InceptionV3 融合的模型在 kaggle 的 leaderboard 上排到第 36 位，而 NASNetLarge 和 InceptionResNetV2 融合的模型可以排到第 20 位，均位于 leaderboard 前 10%，即 131 名以前。

观察 Xception 和 InceptionV3 融合的模型的训练过程，发现模型在第 5 轮取得验证集上最低 loss 后，至第 7 轮训练期间，在训练集上精度

上升依然、loss 依然下降，但在验证集上却出现精度下降、loss 上升的现象，可能有过拟合的趋势；而观察 NASNetLarge 和 InceptionResNetV2 融合的模型的训练过程，模型在训练集上的精度和 loss 总体来说分别上升和下降，但是在验证集上波动较大，有可能是批次 batch\_size 设置得比较小，导致梯度下降过程每次参数更新的幅度比较大。



## 5. 结论

### 5.1. 项目思考

猫狗大战是图像分类识别问题中的经典，虽然“入手”难度不大，但是想要取得优异结果还是不容易的，在 kaggle 竞赛排名榜上，LogLoss 在 0.4 以内的仅有 15 名选手。

整个项目的完成涉及到数据的预处理、特征提取、模型选择与构建、模型训练、参数调优、预测结果分析等，其中需要仔细思考以及多次尝试实验的步骤还是不少的，经历了这次项目，简述下我的以下收获：

- 1) 通常计算机视觉问题对 PC 的显卡和内存要求较高，可以尝试在亚马逊等云服务平台上进行开发，根据需求申请和配置云服务器，这会大大有助于项目开发；
- 2) 从头至尾构建一个完成的 CNN 既耗时又耗资源，在没有充分掌握 CNN 之前，建议利用迁移学习的方式去解决问题，多尝试几种预训练模型，再结合自己构建的少数几层网络，而后多进行几次测试，通过测试结果反馈来调整参数甚至改变模型的构造方式，直至到达满意的效果；
- 3) 不要小看数据预处理，我觉得这是一门博大精深的学问！现在的人工智能可以说十分依赖数据，没有数据一切都是空谈，只有将数据处理好，包括异常数据的清洗、归一化、图像的实时增强等，才不至于误导你的模型，从而模型才能学习到正确的方向，最终获得好的结果。

## 5.2. 突破

在本项目中，我始终没有成功将 LogLoss 突破至 0.4 以内，有可能异常数据清理做得不够科学，我使用了几种模型预测结果的并集，这有可能导致部分正常的猫狗图片也被清理掉了，从而使得模型没有学习到部分特征；另外，还可以尝试选择其它预训练模型、改变自定义的几层网络层或者改变下优化器和其它参数等，又或者干脆不做模型融合，选定单一预训练模型，然后对其进行微调，或许能获得惊喜的结果。

## 参考文献

- [1] Goodfellow, I., Bengio, Y., Courville, A. . Deep learning (Vol. 1) . Cambridge : MIT press, 2016 : 326-366
- [2] 薛景浩, 章毓晋, 林行刚. 图像分割中的交叉熵和模糊散度算法 . 《 电子学报 》, 1999
- [3] 基于卷积神经网络的深度学习算法与应用研究 . 知网 [引用日期 2019-07-17]
- [4] 深度学习是什么? 它的工作原理是什么? 百度文献 [发布日期 : 2018-11-30]
- [5] 周志华 . 机器学习 . 北京 : 清华大学出版社, 2015 : 114-115
- [6] CSDN 卷积神经网络超详细介绍 [发布日期 : 2018-09-19]
- [7] 基于图像底层特征的图像聚类与检索研究[D]. 昆明理工大学, 2018.