机器学习纳米学位毕业项目
猫狗大战 App 实现

何伟华

2019 年 8 月 10 日

1

目录

I. 问题的定义 2
1.1 项目概述 2
1.2 实验环境 2
1.3 问题陈述 3
1.4 评价指标 3
II. 分析 4
2.1 数据探索及预处理 4
2.1.1 绘制训练集图片的尺寸散点分布图 5
2.1.2 获取训练集中，高度或者宽度小于 50 的图片 7
2.1.3 通过对图片中的色彩-像素比进行 IQR 分析 7
2.1.4 采用预处理模型查找最可能不是猫狗图片 8
2.2 卷积网络基本原理 12
2.3 基准指标 15
III. 方法实现 15
3.1 迁移学习 15
3.1.1 模型选择 15
3.1.2 数据预处理及模型调整 16
3.2 实现 16
3.2.1 生成迁移学习特征向量 17
3.2.2 载入特征向量 17
3.2.3 构建模型 18
3.2.4 训练模型 18
3.2.5 预测测试集 21
3.2.6 多种模型融合结果生成 22
3.3 改进 22
IV. App 应用实现部分： 25
4.1 加载数据集 25
4.2 搭建和训练分类器 26
4.3 搭建分类器模型和 CAM 模型 27
4.4 保存模型、可视化 29
4.5 导出 mlmodel 模型文件，在 iOS 实现运行 32
V. 结果分析 34
VI. 总结感想 36
VII. 参考文献 37

I. 问题的定义

1.1 项目概述

Cats vs. Dogs 来源于 Kaggle 大数据竞赛的一道赛题（娱乐型竞赛项目）：https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition，利用给定的数据集，用算法实现猫和狗的识别。

我们的目的需要用训练集对模型进行训练，然后在测试集上“考试”，提交 kaggle 取得高分查看考试结果。本项目使用卷积神经网络识别一张图片是猫还是狗，这是一个二分类问题。1 表示分类结果是狗，0 表示分类结果是猫。

项目背景：kaggle 一共举行过两次猫狗大战的比赛，第一次是在 2013 年，那个时候使用的是正确率作为评估标准，而在 2017 年第二次举办猫狗大战的比赛时，使用的是 log 损失函数。这么做是因为现在深度学习的发展到一定阶段有更好的方法，而深度学习尤其适合处理图像方面的问题，如果依旧是使用正确率作为评估标准，那么大多数选手的模型都是 99%的正确率，不能明显地区分开。如果使用 log 损失函数，不仅仅需要分类正确，还需要对结果有一个较高的可信度，这样就能明显地区分各个模型的分类效果，尤其是 Top 模型的分类效果。 因此参赛者需要训练一个机器学习模型，输入测试集中的图片，输出一个概率，概率越接近 1，表示该图片分类结果是狗的概率越高；概率越接近 0，表示该图片分类结果是猫的概率越高。

本项目数据集是 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)。

项目最终需要训练基于 CNN 的机器学习模型，对测试样本进行分类，并将最终结果上传 kaggle 进行最终评判。同时也实现了使用 Keras 和 Xcode 实现猫狗大战 App，可以通过摄像头或者照片点选的一张猫或者狗的图片预测是猫或者狗的概率。

- 输入：一张彩色图片
- 输出：是猫还是狗的概率

1.2 实验环境

本项目使用 Anaconda 搭建环境。mac os、jupyter notebook、python3、keras、xcode 等，模型训练使用云服务。

1.3 问题陈述

猫狗大战是 Kaggle 娱乐型竞赛项目，我们的目的需要用训练集对模型进行训练，然后在测试集上“考试”，提交 kaggle 取得高分查看考试结果，现在是 2019 年，这个是 2 年前的比赛，是不能查看自己排名。本项目使用卷积神经网络识别一张图片是猫还是狗，这是一个二分类问题。给定一张图片，算法需要预测出图片属于预先定义类别中的哪一类。在计算机视觉领域，目前解决这类问题是深度学习（Deep Learning），特别针对图像类型的数据，是深度学习中的卷积神经网络 CNN 架构，针对图像识别特别棒。

数据集中大部分图片是正常的，有少部分异常图片和低分辨率图片，图像分辨率差异较大，质量参差不齐，图片中的猫和狗颜色丰富，品种多样，姿态各异，背景环境复杂，这些都增加了分类难度。

对于训练集来说这些异常数据是要剔除掉的。 数据集中的文件名是以 type.num.jpg 方式命名的，比如 cat.0.jpg。使用 Keras 的 ImageDataGenerator 需要将不同种类的图片分在不同的文件夹中。 数据集中的图像大小是不固定的，但是神经网络输入节点的个数是固定的。所以在将图像的像素作为输入之前，需要将图像的大小进行 resize。要想取得好的名次，需要选择合理的分类器。

在处理图像的过程中，实际存在两个子过程:特征提取和对象分类。在特征提取环节， 模型以模型原始像素点为基础，从中提取边缘、纹理、对象轮廓、色彩组合，等不同抽象级 别的特征。在对象分类部分，需要将抽取的特征输入到一个分类其中，得到分类的结果，常见的分类器有 Softmax，SVM 等。

1.4 评价指标

对数损失（Log loss）亦被称为逻辑回归损失（Logistic regression loss）或交叉熵损失（Cross-entropy loss）。 交叉熵是常用的评价方式之一，它实际上刻画的是两个概率分布之间的距离，是分类问题中使用广泛的一种损失函数。 本文实际上是二分类问题， 因此可以采用 logloss 损失函数作为评价指标， 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 是测试集中图片数量
- $\hat{y}_i$ 是图片预测为狗的概率
- y_i 如果图像是狗，则为1，如果是猫，则为0
- $log()$ 是自然（基数 e）对数

采用交叉熵作为损失函数可以有效的解决梯度消失和梯度爆炸的问题。 交叉熵损失越小，代表模型的性能越好。上述评估指标可用于评估该项目的解决方案以及基准模型。

II. 分析

2.1 数据探索及预处理

下载 kaggle 猫狗数据集解压后分为 3 个文件 train.zip、 test.zip 和 sample_submission.csv。 数据集由训练数据和测试数据组成，训练数据包含猫和狗各 12500 张图片，测试数据包含 12500 张猫和狗的图片。命名规则根据“type.num.jpg” 方式命名。test 测试集包含了 12500 张猫狗的图片，没有标定是猫还是狗，每张图片命名规则根据 “num.jpg”，需要注意的是测试集编号从 1 开始，而训练集的编号从 0 开始。 sample_submission.csv 需要将最终测试集的测试结果写入 submission.csv 文件中，上传至 kaggle 进行打分。

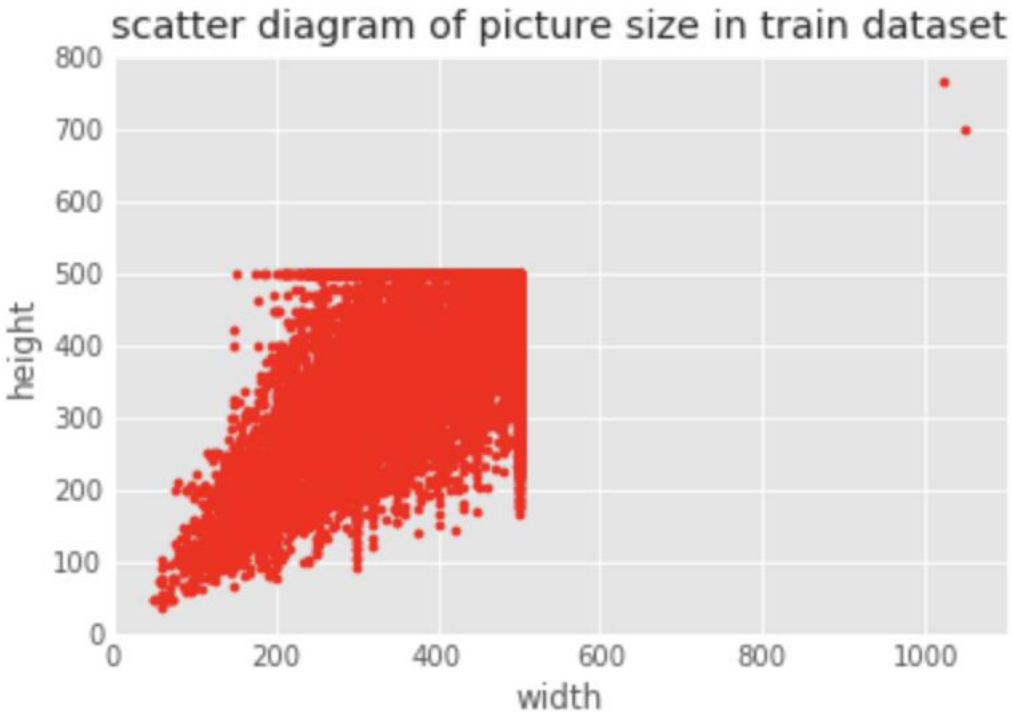
具体步骤： 从 Dogs vs. Cats Redux: Kernels Edition 下载训练数据到 image 目录并解压到当前目录。 从训练集中提取图片可视化显示。

从上图中可看出，图片丰富多样性。大部分图片质量和分辨率是好的，个别图像分辨率过小或者质量差。比如上图第三排最后一张图片，背景比较复杂，可能会影响模型的学习。

另外，由于猫狗所占比重一样，在训练是不需要调整类别权重。

探索查找异常数据（train 数据集）

2.1.1 绘制训练集图片的尺寸散点分布图



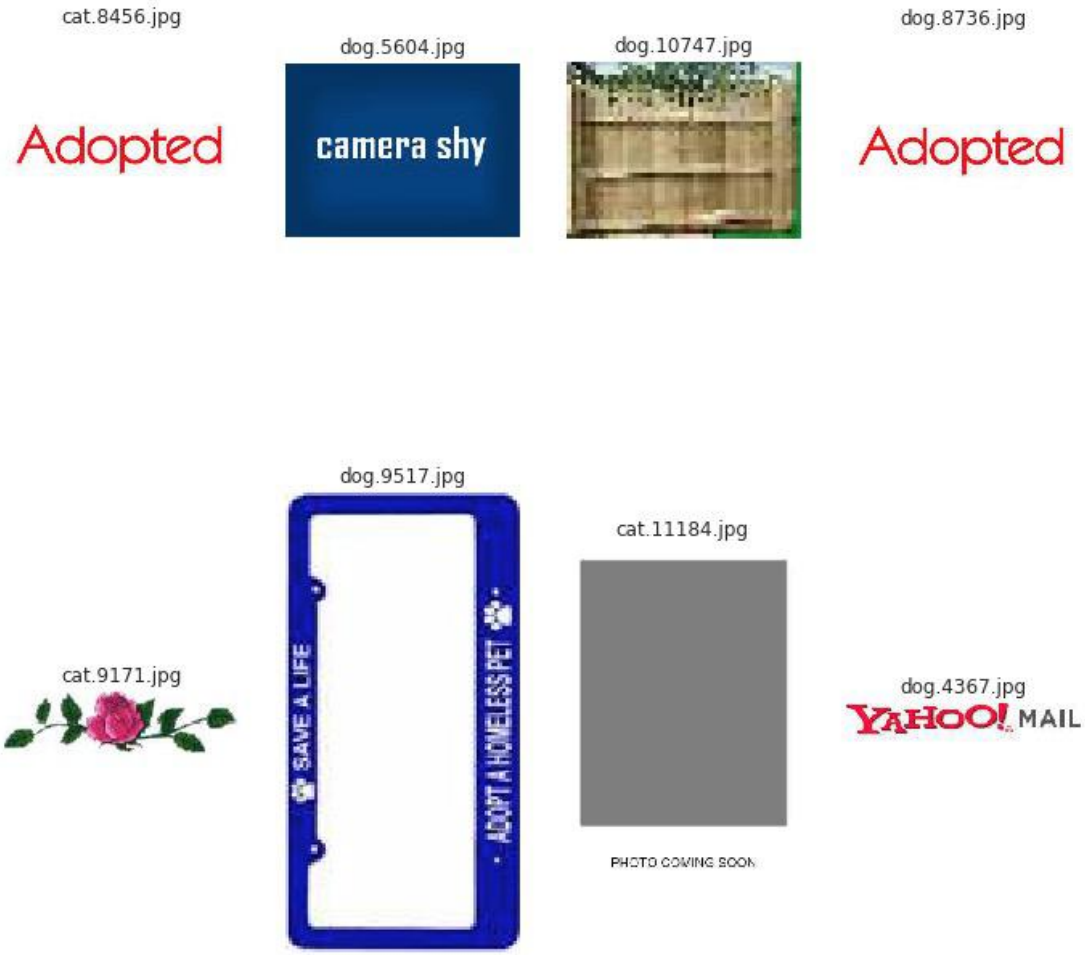
可看出大部分图片尺寸是一致，有两种图片特别大，有小部分图片比较小，训练是需要将图片大小要做归一化。

2.1.2 获取训练集中，高度或者宽度小于 50 的图片

结论，非猫狗异常图片：[‘dog.10747.jpg’，‘cat.9171.jpg’，‘dog.4367.jpg’]

2.1.3 通过对图片中的色彩-像素比进行 IQR 分析

可找出一些异常图片，如下图所示：
['cat.8456.jpg', 'dog.5604.jpg', 'dog.10747.jpg', 'dog.8736.jpg', 'cat.9171.jpg', 'dog.9517.jpg', 'cat.11184.jpg', 'dog.4367.jpg']



2.1.4 采用预处理模型查找最可能不是猫狗图片

使用 Xception 得出结论：

总共 122 张可疑图片，其中猫 89 张、狗 33 张。
手动挑选出异常图片 22 张：
['dog.1773.jpg','dog.6475.jpg','dog.10237.jpg','dog.12376.jpg','cat.10712.jpg',
'dog.10747.jpg','dog.11299.jpg','cat.8456.jpg','dog.5604.jpg', 'dog.8736.jpg',
'cat.9171.jpg', 'cat.7564.jpg', 'dog.9517.jpg', 'dog.2614.jpg', 'dog.1043.jpg',
'cat.5351.jpg', 'dog.10801.jpg','cat.7377.jpg', 'cat.5418.jpg', 'cat.4338.jpg',
'cat.11184.jpg','dog.4367.jpg']

使用 InceptionV3 得出结论：

总共 191 张可疑图片，其中猫 152 张、狗 39 张。

手动挑选出异常图片 23 张：
['cat.1773.jpg','dog.6475.jpg','dog.10237.jpg', 'dog.12376.jpg', 'cat.10712.jpg',
'dog.11299.jpg','cat.8456.jpg','dog.5604.jpg','dog.10747.jpg','dog.8736.jpg',
'cat.9171.jpg','dog.1194.jpg','cat.7564.jpg','dog.9517.jpg','dog.2614.jpg',
'dog.1043.jpg','cat.5351.jpg','dog.10801.jpg','cat.7377.jpg','cat.5418.jpg',
'cat.4338.jpg', 'cat.11184.jpg', 'dog.4367.jpg']

使用 ResNet50 得出结论：

总共 250 张可疑图片，其中猫 191 张、狗 59 张。
手动挑选出异常图片 22 张：
['dog.1773.jpg','dog.6475.jpg','dog.12376.jpg','dog.10237.jpg', 'cat.10712.jpg',
'dog.10747.jpg','dog.11299.jpg','cat.8456.jpg','dog.5604.jpg', 'dog.8736.jpg',
'cat.9171.jpg', 'cat.7564.jpg', 'dog.9517.jpg', 'dog.2614.jpg', 'dog.1043.jpg',
'cat.5351.jpg', 'dog.10801.jpg', 'cat.7377.jpg', 'cat.5418.jpg', 'cat.4338.jpg',
'cat.11184.jpg','dog.4367.jpg']

使用 InceptionResNetV2 得出结论：

总共 271 张可疑图片，其中猫 236 张、狗 35 张。
手动挑选出异常图片 23 张：
['dog.1773.jpg', 'dog.6475.jpg', 'dog.10237.jpg', 'dog.12376.jpg', 'cat.10712.jpg',
'dog.11299.jpg', 'cat.8456.jpg', 'dog.5604.jpg', 'dog.10747.jpg', 'dog.8736.jpg',
'cat.9171.jpg', 'dog.1194.jpg', 'cat.7564.jpg', 'dog.9517.jpg', 'dog.2614.jpg',
'dog.1043.jpg', 'cat.5351.jpg', 'dog.10801.jpg', 'cat.7377.jpg', 'cat.5418.jpg',
'cat.4338.jpg', 'cat.11184.jpg', 'dog.4367.jpg']

综合上面查找的结果，非猫狗异常图片 23 张：

['dog.1773.jpg', 'dog.6475.jpg', 'dog.10237.jpg', 'dog.12376.jpg', 'cat.10712.jpg',
'dog.11299.jpg', 'cat.8456.jpg', 'dog.5604.jpg', 'dog.10747.jpg', 'dog.8736.jpg',
'cat.9171.jpg', 'dog.1194.jpg', 'cat.7564.jpg', 'dog.9517.jpg', 'dog.2614.jpg',
'dog.1043.jpg', 'cat.5351.jpg', 'dog.10801.jpg', 'cat.7377.jpg', 'cat.5418.jpg',
'cat.4338.jpg', 'cat.11184.jpg', 'dog.4367.jpg']

可疑的图片图片不要挑出来，因为在真实的场景下也会有这种图片，要想模型学到真本事，这些图片应该保留下来。

数据集中的文件名是以 type.num.jpg 方式命名的，比如 cat.0.jpg。使用 Keras 的 ImageDataGenerator 需要将不同种类的图片分在不同的文件夹中。对数据集进行预处理参考的是[杨培文的 Blog](http://www.zhiding.cn/techwalker/documents/J9UpWRDfVYHE5WsOEHbyx4eM8fBcpHYEW_b72QCUihQ)创建符号链接(symbol link)的方法，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('img_train')
os.mkdir('img_train/cat')
os.mkdir('img_train/dog')

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

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

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

def rmrf_mkdir(dirname):
    if os.path.exists(dirname):
        shutil.rmtree(dirname)
    os.mkdir(dirname)

def rmrf_mkdir(dirname):
    if os.path.exists(dirname):
        shutil.rmtree(dirname)
    os.mkdir(dirname)
```

图像文件分类后的路径如下：

```
image
├── test
├── train
├── img_train
│   ├── cat
│   └── dog
```

可视化数据集：

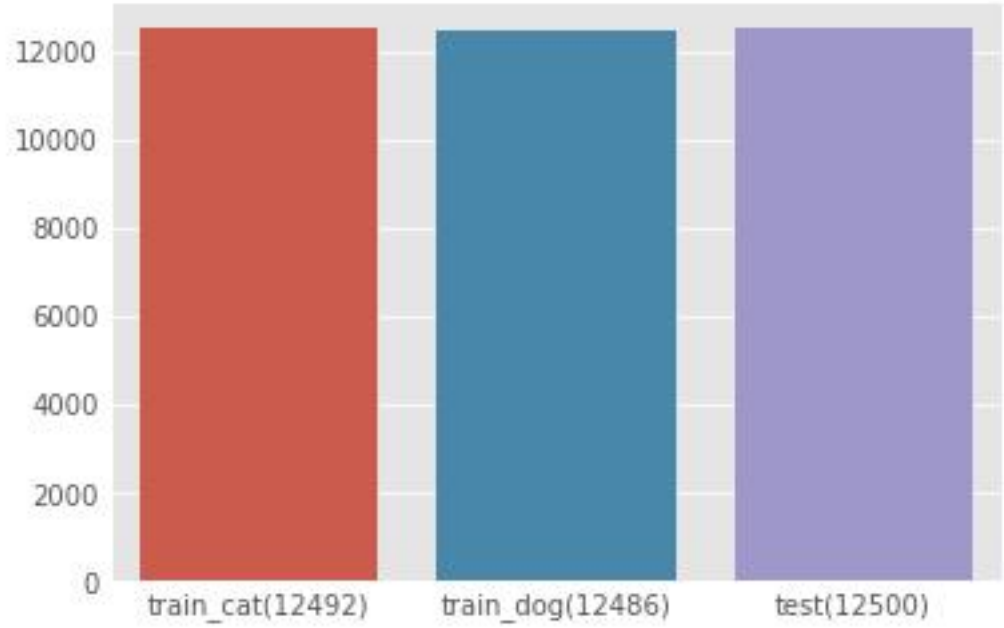
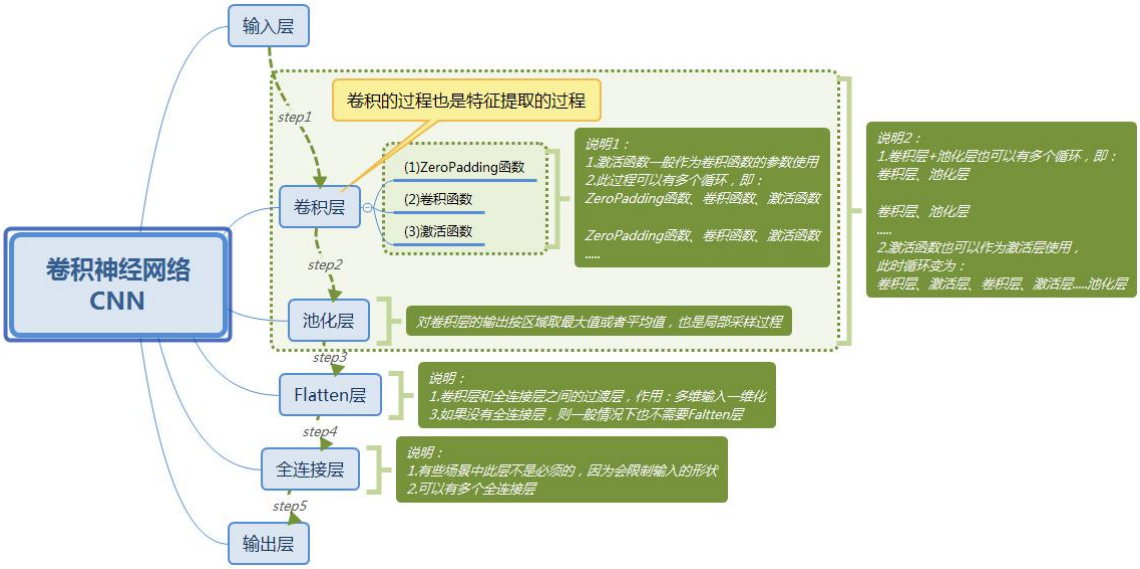


image 数据集中，猫的数量：12492，狗的数量：12486，测试集图片数量：12500

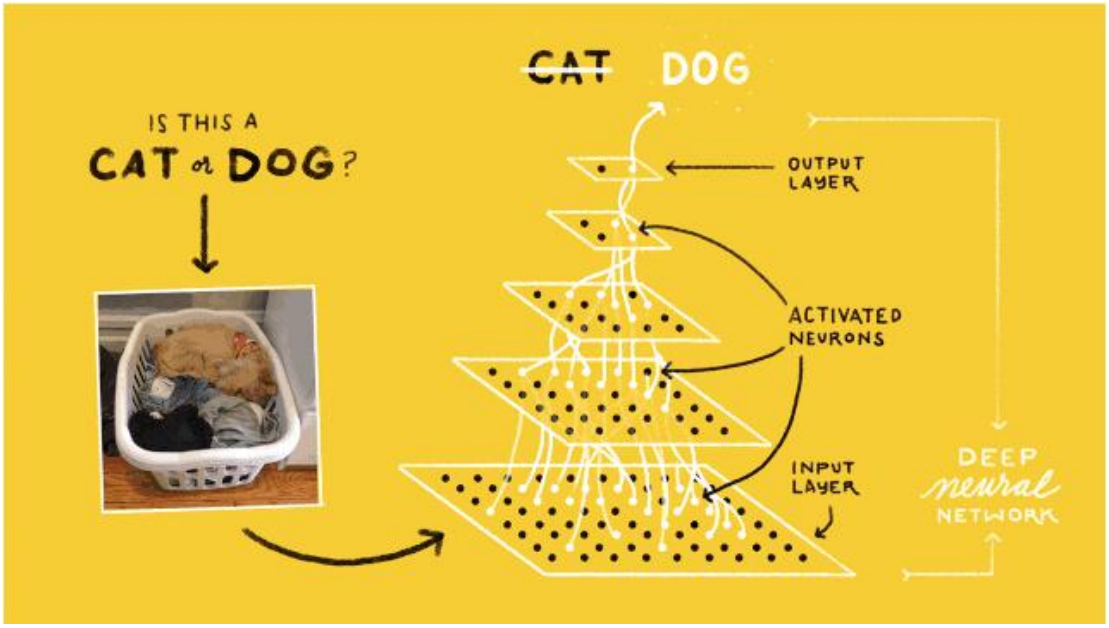
2.2 卷积网络基本原理

在给定一张图片活视频流的一帧图片，模型预测出图像属于预先定义类别中的哪一类。在计算机视觉领域，目前解决这类问题的核心技术框架是深度学习（Deep Learning），特别针对图像类型的数据，是深度学习中的卷积神经网络

（Convolutional Neural Networks, ConvNets）架构。常见的卷积神经网络架构如下：

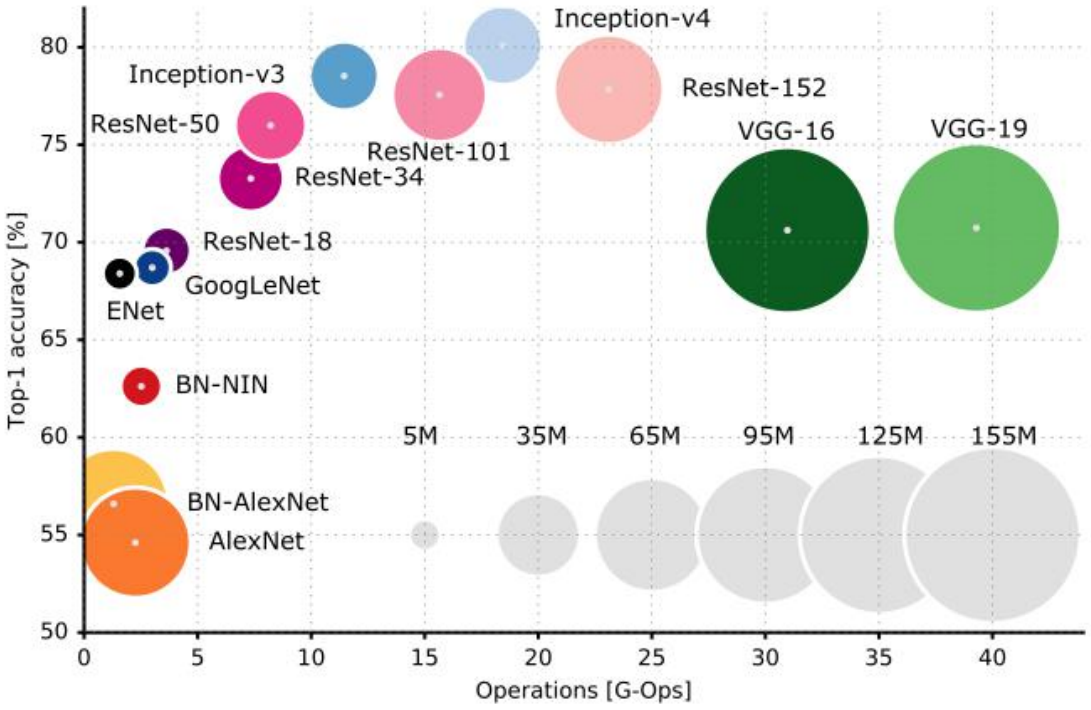


总的来说，卷积神经网络是一种特殊的神经网络结构，即通过卷积操作可以实现对图像特征的自动学习，选取那些有用的视觉特征以最大化图像分类的准确率。



上图给出了一个简单的猫狗识别的卷积神经网络结构，在最底下（同时也是最大的）的点块表示的是网络的输入层（Input Layer），通常这一层作用是读入图像作为网络的数据输入。在最上面的点块是网络的输出层（Output Layer），其作用是预测并输出读入图像的类别，在这里由于只需要区分猫和狗，因此输出层只有 2 个神经计算单元。而位于输入和输出层的，都称之为隐含层（Hidden Layer），图中有 3 个隐含层，图像分类的隐含层都是由卷积操作完成的，因此这样的隐含层也成为卷积层（Convolutional Layer）。因此，输入层、卷积层、输出层的结构及其对应的参数就构成了一个典型的卷积神经网络。当然，在实际中使用的卷积

神经网络要比这个示例的结构更加复杂，自 2012 年的 ImageNet 比赛起，几乎每一年都会有新的网络结构诞生，已经被大家认可的常见网络有 AlexNet, VGG-Net, GoogLeNet, Inception V2-V4, ResNet 等等。这些卷积神经网络都是在 ImageNet 数据集上表现非常优异的神经网络，具体准确率和模型大小如下图所示。



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

卷积神经网络中卷积层和池化层主要是对图片的几何特征进行抽取，比如浅层的卷积池化层可以抽取出一些直线，角点等简单的抽象信息，深层的卷积池化层可以抽取人脸等复杂的抽象信息，最后的全连接层才是对图片分类的关键处理。因

此可以利用已经训练好的卷积神经网络提取图片中复杂的几何特征，即将原始图片用已经训练好的卷积神经网络处理之后的输出，作为新的输入，然后加上自己的全连接层，去进行分类。在模型训练的过程中，只改变新加的全连接层的权重。

由于每一种神经网络提取的特征都不一样，因此本项目将多个神经网络处理的结果拼接，作为最后一层全连接层的输入，这样做可以有效地降低方差。

2.3 基准指标

项目目使用Xception, InceptionV3，InceptionResNetV2 三个模型完成。本项目目的最低要求是 kaggle Public Leaderboard 前 10%。在 kaggle 上，总共有 1314 只队伍参加了了比比赛，所以需要最终的结果排在 131 位之 前，131 位的得分是 0.06127，所以目目标是模型预测结果要小小于 0.06127。

III. 方法实现

3.1 迁移学习

3.1.1 模型选择

CNN 的核心是对图像特征的表示，从最基本的边缘、纹路，到较高抽象层次的对象。CNN 通过各层卷积核及其对应的权重来记录这些特征，同时，通常情况下利用最后一层的 softmax 进行概率分配，从而得出分类结果。由此可见，大部分卷积神经网络是用来做特征 提取的，只有最后的很少几层是用来得出分类结果的。
从零开始训练一个卷积神经网络，需要进行细致的网络结果设计，并调整大量的参数进 行优化。幸运的是，在知名计算机视觉竞赛 ImageNet 中，类别就含有猫和狗，同时，许多 优秀的网络结构都提供了在 ImageNet 数据集上的训练结果。由此，利用现有的模型以及预 训练出的权重，就能很好的表征猫和狗的特征，再辅以一个简单的分类器就能够获得一个相对理想的结果。

根据在 imagenet 数据集上的预测准确率，排名比较好的的是 InceptionResNetV2、Xception、InceptionV3 考虑用这三个模型进行融合。

3.1.2 数据预处理及模型调整

在 keras 文档中的预处理函数，不同的模型对于输入数据都有各自的默认值，比如在输入图片大小维度上：Xception、InceptionV3、InceptionResNetV2 默认输入图片大小是 299x299，ResNet50 默认输入图片大小是 224x224；在输入数值维度上，Xception、InceptionV3、InceptionResNetV2 默认输入数值在(-1, 1)范围内。当要输入与默认图片大小不同的图片时，只需传入当前图片大小即可。ResNet50 需要对图片进行中心化处理，由于载入的 ResNet50 模型是在 ImageNet 数据上训练出来的，所以在预处理中每个像素点都要减去 ImageNet 均值。当要输入与默认图片大小不同的图片时，只需传入当前图片大小即可。当输入数值不符合默认要求时，使用每个模型的预处理函数 preprocess_input 即可将输入图片处理成该模型的标准输入。

常见的卷积神经网络结构在前面的若干层都是卷积池化层及其各种变种，后面几层都是全连接层，这些全连接层之前的网络层被称为瓶颈层 (bottleneck). 将新的图片通过训练好的卷积神经网络直到瓶颈层的过程可以看做是对图像进行特征提取的过程。一般情况下，为了减少内存的消耗， 加快计算的过程，再将瓶颈层的结果输入全连接层之前，做一次全局平均池化，比如 ResNet50 瓶颈层输出结果是 7x7x2048，如果直接输入到全连接层，参数会非常多，所以进行一次全局平均池化，将输出矩阵调整为 1x1x2048，这么做还有一个好处，那就是可以降低过拟合的程度。

在 Keras 中载入模型并进行全局平均池化，只需要在载入模型的时候，设置 include_top=False, pooling='avg'. 每个模型都将图片处理成一个 1x2048 的行向量，将这四个行向量进行拼接，得到一个 1x8192 的行向量，作为数据预处理的结果。

3.2 实现

3.2.1 生成迁移学习特征向量

```
from keras.models import *
from keras.layers import *
from keras.applications import *
from keras.applications.inception_resnet_v2 import InceptionResNetV2,preprocess_input
from keras.preprocessing.image import *
import h5py

train_sample_counts = len(os.listdir('img_train'))
test_sample_counts = len(os.listdir('img_test'))

def write_gap(MODEL, image_size, func=None):
    width = image_size[0]
    height = image_size[1]
    inputs = Input((height, width, 3))
    x = inputs
    if func:
        x = Lambda(func)(x)#增加预处理函数层

    base_model = MODEL(input_tensor=x, weights='imagenet', include_top=False)
    model = Model(base_model.input, GlobalAveragePooling2D()(base_model.output))

    gen = ImageDataGenerator()
    train_ge = gen.flow_from_directory("img_train", image_size, shuffle=False, batch_size=16)
    test_ge = gen.flow_from_directory("img_test", image_size, shuffle=False, batch_size=16, class_mode=None)
    train = model.predict_generator(train_ge, train_sample_counts)
    test = model.predict_generator(test_ge, test_sample_counts)


    with h5py.File("gap_%s.h5"%MODEL.__name__) as h:
        h.create_dataset("train", data=train)
        h.create_dataset("test", data=test)
        h.create_dataset("label", data=train_ge.classes)
```

```
write_gap(ResNet50, (224, 224), resnet50.preprocess_input)
write_gap(InceptionV3, (299, 299), inception_v3.preprocess_input)
write_gap(Xception, (299, 299), xception.preprocess_input)
write_gap(InceptionResNetV2, (299, 299), inception_resnet_v2.preprocess_input)
```

3.2.2 载入特征向量

经过上面的代码以后，我们获得了四个特征向量文件，分别是：

- gap_ResNet50.h5
- gap_InceptionV3.h5
- gap_Xception.h5
- gap_InceptionResNetV2.h5

这里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，不然之后设置 validation_split 的时候会出问题。这里设置了 numpy 的随机数种子为 2019。

```
import h5py
import numpy as np
from sklearn.utils import shuffle
np.random.seed(2019)

X_train = []
X_test = []

for filename in ["gap_ResNet50.h5", "gap_Xception.h5", "gap_InceptionV3.h5", "gap_InceptionResNetV2.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)
```

3.2.3 构建模型

载入预处理的数据之后，先进行一次概率为 0.5 的 dropout，减少参数减少计算量，防止过拟合，然后直接连接输出层，激活函数为 Sigmoid，优化器为 Adadelta，输出一个零维张量，表示某张图片中有狗的概率。

```
from keras.models import *
from keras.layers import *
from keras.regularizers 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'])
```

3.2.4 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% 。

其中批处理参数 batch_size，太小时可能会使学习过于随机，虽然训练速率很快，但收敛震动且得到一个不可靠模型。在合理的范围内，增大 batch_size 可以提高内存利用率，大矩阵乘法的并行化效率提高；跑完一次 epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快；在适当范围内，batch_size 越大，其确定的下降方向越准，引起训练波动越小，这里 batch_size=128 是一个比较合适的经验值（在云平台上训练）。

```
from keras.callbacks import *
# model_history = model.fit(X_train, y_train, batch_size=128, nb_epoch=8, verbose=1, validatio
n_split=0.2, callbacks = [TensorBoard(log_dir='./Graph')])

from keras.callbacks import ModelCheckpoint
checkpointer = ModelCheckpoint(filepath='weights.best.cat_vs_dog.hdf5', verbose=1, save_best_o
nly=True)
model_history = model.fit(X_train, y_train, epochs=8, batch_size=128, validation_split=0.2, ca
llbacks=[checkpointer], verbose=1)
```

```
Train on 19914 samples, validate on 4979 samples
Epoch 1/8
19712/19914 [============================>.] - ETA: 0s - loss: 0.1051 - acc: 0.9698Epoch 0000
1: val_loss improved from inf to 0.02436, saving model to weights.best.cat_vs_dog.hdf5
19914/19914 [==============================] - 8s 384us/step - loss: 0.1044 - acc: 0.9701 - v
al_loss: 0.0244 - val_acc: 0.9952
Epoch 2/8
19200/19914 [===========================>..] - ETA: 0s - loss: 0.0205 - acc: 0.9950Epoch 0000
2: val_loss improved from 0.02436 to 0.01567, saving model to weights.best.cat_vs_dog.hdf5
19914/19914 [==============================] - 1s 48us/step - loss: 0.0209 - acc: 0.9949 - va
l_loss: 0.0157 - val_acc: 0.9952
Epoch 3/8
19328/19914 [============================>.] - ETA: 0s - loss: 0.0146 - acc: 0.9961Epoch 0000
3: val_loss improved from 0.01567 to 0.01464, saving model to weights.best.cat_vs_dog.hdf5
19914/19914 [==============================] - 1s 47us/step - loss: 0.0145 - acc: 0.9961 - va
l_loss: 0.0146 - val_acc: 0.9958
Epoch 4/8
19456/19914 [============================>.] - ETA: 0s - loss: 0.0128 - acc: 0.9961Epoch 0000
4: val_loss improved from 0.01464 to 0.01459, saving model to weights.best.cat_vs_dog.hdf5
19914/19914 [==============================] - 1s 48us/step - loss: 0.0127 - acc: 0.9961 - va
l_loss: 0.0146 - val_acc: 0.9954
Epoch 5/8
19328/19914 [============================>.] - ETA: 0s - loss: 0.0111 - acc: 0.9968Epoch 0000
5: val_loss improved from 0.01459 to 0.01395, saving model to weights.best.cat_vs_dog.hdf5
19914/19914 [==============================] - 1s 48us/step - loss: 0.0110 - acc: 0.9968 - va
l_loss: 0.0140 - val_acc: 0.9958
Epoch 6/8
19456/19914 [============================>.] - ETA: 0s - loss: 0.0109 - acc: 0.9969Epoch 0000
6: val_loss did not improve
19914/19914 [==============================] - 1s 47us/step - loss: 0.0107 - acc: 0.9969 - va
l_loss: 0.0146 - val_acc: 0.9958
Epoch 7/8
19456/19914 [============================>.] - ETA: 0s - loss: 0.0098 - acc: 0.9972Epoch 0000
7: val_loss did not improve
19914/19914 [==============================] - 1s 46us/step - loss: 0.0097 - acc: 0.9972 - va
l_loss: 0.0159 - val_acc: 0.9958
Epoch 8/8
19584/19914 [============================>.] - ETA: 0s - loss: 0.0098 - acc: 0.9972Epoch 0000
8: val_loss did not improve
19914/19914 [==============================] - 1s 46us/step - loss: 0.0096 - acc: 0.9972 - va
l_loss: 0.0145 - val_acc: 0.9960
```

训练过中的 loss 和 accuracy 如下：

```
import matplotlib.pyplot as plt
%matplotlib inline
# 画图
def plot_training(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    epochs = range(len(acc))
    plt.plot(epochs, acc, 'b')
    plt.plot(epochs, val_acc, 'r')
    plt.legend(["acc", "val_acc"], loc='best')
    plt.title('Training and validation accuracy')
    plt.figure()

    loss = history.history['loss']
    val_loss = history.history['val_loss']
    plt.plot(epochs, loss, 'b')
    plt.plot(epochs, val_loss, 'r')
    plt.legend(["loss", "val_loss"], loc='best')
    plt.title('Training and validation loss')
    plt.show()

# 训练的acc_loss图
plot_training(model_history)
```

运行出结果:

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 99.60%的准确率，这相当于一千张图只错了 4 张，非常不错的成绩了。

3.2.5 预测测试集

使用训练好的模型中对处理过的测试集数据进行分类，得到每张图片中有狗的概率。在提交的 kaggle 之前，先使用 numpy.clip()函数做一个截断处理，将所有图片的概率值限制在[0.005,0.995]之间，这样可以稍微降低 loss。

kaggle 官方的评估标准是 LogLoss，下面的表达式就是二分类问题的 LogLoss 定义。

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

$|log(1)| = 0.000$

$|log(0.995)| = 0.0050$

$lim_{x\to 0}|log(x)| = \infty$

$|log(0.005)| = 5.2983$

不过由于 kaggle 本身就做了一次截断，限制了最小值为 1e−15，又|log(1e−15)|=34.5388。所以如果把预测正确的概率从 1 截断到 0.995，单张图片的 loss 最多增加 0.0050，但是如果把预测错误的概率从 1e−15 截断到 0.005，那么会极大地降低 loss。

由于 kaggle 是根据 id 来确定每张图片的类别的，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。

```
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("img_test", image_size, shuffle=False,
                                         batch_size=16, class_mode=None)

y_pred = model.predict(X_test, verbose=1)
y_pred = y_pred.clip(min=0.005, max=0.995)

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('submission.csv', index=None)
df.head(10)
```

3.2.6 多种模型融合结果生成

Xception-InceptionResNetV2 融合模型-adam
loss：0.0111 - acc：0.9970 - val_loss：0.0165 - val_acc：0.9958

ResNet50-InceptionV3-Xception 融合模型-adam
loss：0.0087 - acc：0.9975 - val_loss：0.0140 - val_acc：0.9964

ResNet50-InceptionV3-InceptionResNetV2 融合模型-adam
loss：0.0086 - acc：0.9974 - val_loss：0.0134 - val_acc：0.9958

InceptionV3-Xception-InceptionResNetV2 融合模型-adam
loss：0.0119 - acc：0.9967 - val_loss：0.0137 - val_acc：0.9960

InceptionV3-Xception-InceptionResNetV2 融合模型-adadelta
loss：0.0130 - acc：0.9960 - val_loss：0.0146 - val_acc：0.9956

ResNet50-InceptionV3-Xception-InceptionResNetV2 融合模型-adam
loss：0.0104 - acc：0.9971 - val_loss：0.0139 - val_acc：0.9962

3.3 改进

从上一步结果可看出：

1. 三个模型融合比两个模型效果要好；
2. 当四个模型融合，效果没有明显的提升，反而运输量和复杂度成本提高了；
3. 优化器 adam 比 adadelta 速度更快，且精度更高；

优化器 adam、adadelta 比较：

Adadelta 是对 Adagrad 的扩展，最初方案依然是对学习率进行自适应约束，但是进行了计算上的简化。Adagrad 会累加之前所有的梯度平方，而 Adadelta 只累加固定大小的项，并且也不直接存储这些项，仅仅是近似计算对应的平均值。即：

$$n_t = \nu * n_{t-1} + (1-\nu) * g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

在此之后 Adadelta 其实还依赖于全局学习率，但是作者做了一定处理，经过近似牛顿迭代法之后：

$$E|g^2|_t = \rho * E|g^2|_{t-1} + (1-\rho) * g_t^2$$

$$\Delta x_t = -\frac{\sqrt{\sum_{r=1}^{t-1} \Delta x_r}}{\sqrt{E|g^2|_t + \epsilon}}$$

其中，E 代表求期望。

此时，可以看出 Adadelta 已经不用依赖全局学习率了。

特点：

- 训练中期，加速效果不错，很快
- 训练后期，反复在局部最小值抖动

Adam 本质上是带有动量项的 RMSprop,它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam 的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳，公式如下：

$$m_t = \mu * m_{t-1} + (1-\mu) * g_t$$

$$n_t = \nu * n_{t-1} + (1-\nu) * g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\mu^t}$$

$$\hat{n}_t = \frac{n_t}{1-\nu^t}$$

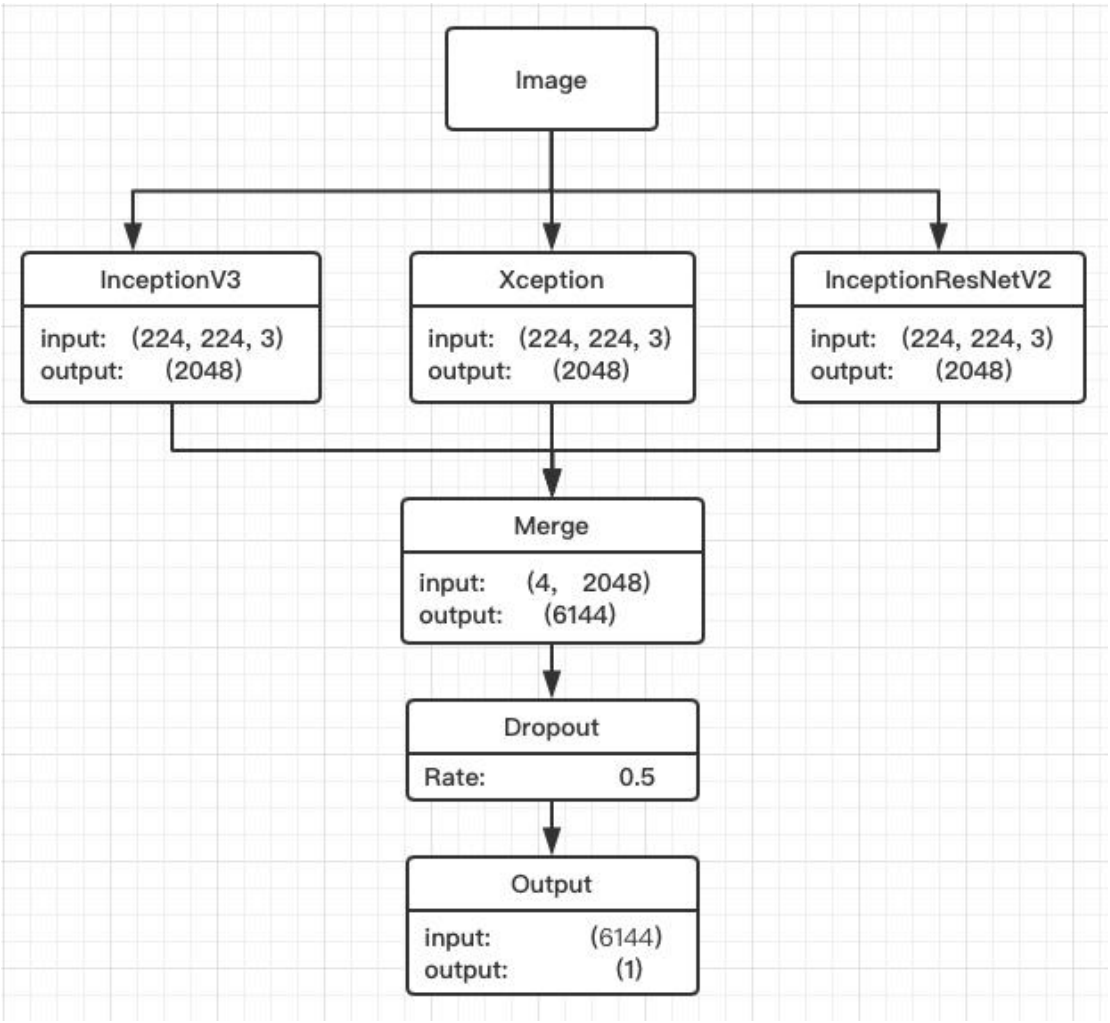
$$\Delta\theta_t = -\frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon} * \eta$$

特点：

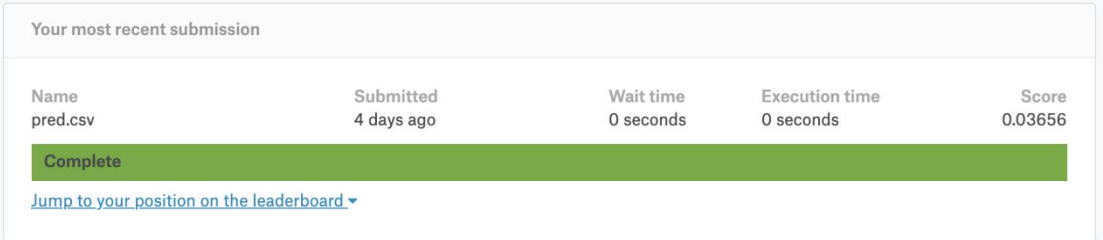
- 结合了 Adagrad 善于处理稀疏梯度和 RMSprop 善于处理平稳目标的优点
- 对内存需求较小
- 为不同的参数计算不同的自适应学习率
- 也适用于大多非凸优化-适用于大数据集和高维空间

所以，这里做一些调整：InceptionV3-Xception-InceptionResNetV2 使用这三个模型进行融合，实践证明使用 adam 优化器效果更好。

整个迁移学习的神经网格结构如下所示。



将测试集的处理结果提交到 kaggle 上，loss 为 0.03556，和验证集的 loss 近似。



IV. App 应用实现部分：

为了在 App 上运行模型效果，使项目落地，通过下面的步骤来实现。
这里使用单模型 RetNet50 来训练的模型预测训练集可视化模型

4.1 加载数据集

1. 这里的图片 224x224 裁剪，不是从图片中心向四周裁剪，如果图片超出规定尺寸裁剪，最后可能只剩下中间区域一部分，这可能是狗的躯干，头或者尾巴没了，这样训练肯定会影响模型的学习结果；而是对图片进行缩放，这样就不会丢掉猫狗重要部分的了。

2. 这里要注意一下，做普通的猫狗识别 y 是一维的，导出分类模型 y 变成两个维度，因为需要把猫编码成【1，0】，狗编码成【0，1】，才能针对分类进行 CAM 可视化。

3. 如果是一个神经元 sigmoid 做二分类，模型只会把狗对应的权值加大，对模型来说，猫和其他背景没有差别，因此需要弄两个维度，然后用 sofmax 激活函数，这样模型不仅仅能区分狗和非狗了，因为如果把猫也看作背景，sigmoid 之后没办法让猫的输出比狗大。

```
import cv2
import numpy as np
from tqdm import tqdm


img_size = 224
def load_train(img_size):

    train = os.listdir('train')
    n = len(train)
    X = np.zeros((n, img_size, img_size, 3), dtype=np.uint8)
    y = np.zeros((n, 2), dtype=np.uint8) #one-hot编码 0为 (1,0) , 1为 (0,1)
    #y = np.zeros((n, 1), dtype=np.uint8) #one-hot编码 0为cat, 1为dog
#     for i, filename in zip(tqdm(range(n)), train):
    for i, filename in zip(range(n), train):
        name = filename[0: 3]
        if(name == 'cat'):
            y[i] = [1, 0]
        elif(name == 'dog'):
            y[i] = [0, 1]
        else:
            return
        X[i] = cv2.resize(cv2.imread('train/%s' % filename), (img_size, img_size))

    return X,y
X_train, y_train = load_train(img_size)
print("训练样本加载完成，训练集图片数量：{}".format(len(X_train)))
```

提取特征

```
from keras.layers import *
from keras.models import *
from keras.applications import *
from keras.optimizers import *
from keras.regularizers import *
from keras.applications.resnet50 import preprocess_input


# 数据预处理方法是减去训练集每个像素点颜色的平均值
def preprocess_input(x):
    return x - [103.939, 116.779, 123.68]

def get_features(MODEL, data=X_train):

    cnn_model = MODEL(include_top=False, input_shape=(224, 224, 3), weights='imagenet')
    inputs = Input((224, 224, 3))
    x = inputs
    x = Lambda(preprocess_input, name = 'preprocessing')(x) #preprocess_input函数因预训练模型而异
    x = cnn_model(x)
    x = GlobalAveragePooling2D()(x)
    cnn_model = Model(inputs, x)

    features = cnn_model.predict(data, batch_size = 64, verbose=1)
    return features

features = get_features(ResNet50)
```

```
24893/24893 [==============================] - 1609s 65ms/step
```

4.2 搭建和训练分类器

构建模型，参考：

https://github.com/fchollet/keras/blob/master/keras/applications/resnet50.py

分为训练集和验证集：

http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
inputs = Input(features.shape[1:])
x = inputs
x = Dropout(0.5)(x)
x = Dense(2, activation = 'softmax', kernel_regularizer = l2(1e-4), bias_regularizer = l2(1e-4
))(x)
model = Model(inputs, x, name = 'prediction')
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
h = model.fit(features, y_train, batch_size = 128, epochs = 8, validation_split = 0.2)
# 模型概括
model.summary()
```

```
Train on 19914 samples, validate on 4979 samples
Epoch 1/8
19914/19914 [==============================] - 57s 3ms/step - loss: 0.1131 - acc: 0.9544 - va
l_loss: 0.0276 - val_acc: 0.9894
Epoch 2/8
19914/19914 [==============================] - 1s 47us/step - loss: 0.0474 - acc: 0.9842 - va
l_loss: 0.0246 - val_acc: 0.9912
Epoch 3/8
19914/19914 [==============================] - 1s 46us/step - loss: 0.0412 - acc: 0.9868 - va
l_loss: 0.0254 - val_acc: 0.9916
Epoch 4/8
19914/19914 [==============================] - 1s 48us/step - loss: 0.0372 - acc: 0.9875 - va
l_loss: 0.0266 - val_acc: 0.9906
Epoch 5/8
19914/19914 [==============================] - 1s 47us/step - loss: 0.0334 - acc: 0.9883 - va
l_loss: 0.0241 - val_acc: 0.9920
Epoch 6/8
19914/19914 [==============================] - 1s 47us/step - loss: 0.0322 - acc: 0.9881 - va
l_loss: 0.0248 - val_acc: 0.9918
Epoch 7/8
19914/19914 [==============================] - 1s 46us/step - loss: 0.0313 - acc: 0.9898 - va
l_loss: 0.0293 - val_acc: 0.9875
Epoch 8/8
19914/19914 [==============================] - 1s 46us/step - loss: 0.0356 - acc: 0.9878 - va
l_loss: 0.0263 - val_acc: 0.9910
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_46 (InputLayer)        (None, 2048)              0
_________________________________________________________________
dropout_18 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_18 (Dense)             (None, 2)                 4098
=================================================================
Total params: 4,098
Trainable params: 4,098
Non-trainable params: 0
_________________________________________________________________
```

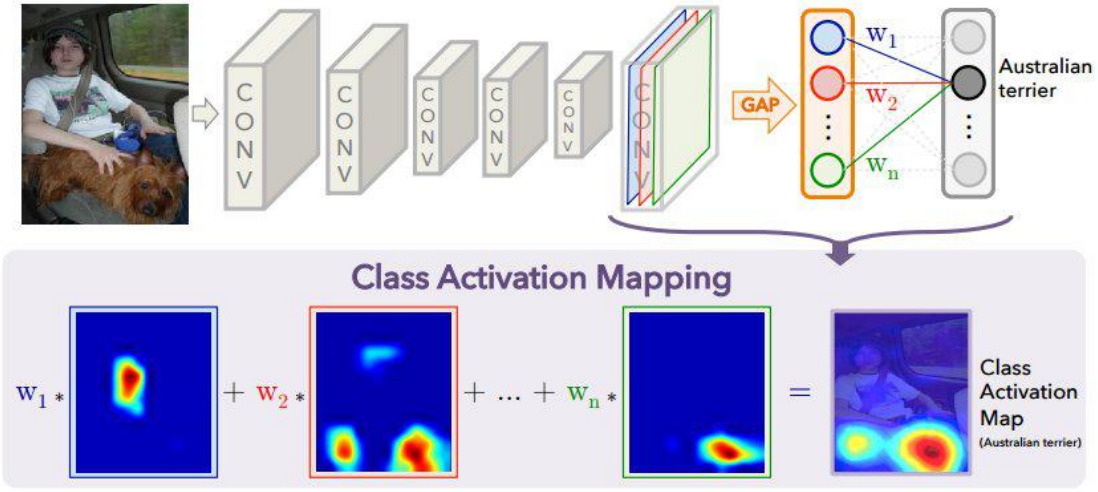
4.3 搭建分类器模型和 CAM 模型

ResNet50 最后一层带有一个（7，7）的池化层，但是需要输出卷积层的原始数据，而不是把每个激活图压缩成一个点，因此取倒数第二层搭建成一个新的 cnn_model，再去搭建 CAM 模型。

在对卷积层输出的激活图进行加权平均的时候，可理解为卷积核大小为 1x1 的不带 bias 的卷积层。 分类器就简单了，直接用 GlobalAveragePooling2D 进行平均，然后用刚才训练的 model 算一下。

全局平均池化层（Global AveragePooling, GAP）通常用在卷积神经网络的最后一层，用于降维，使用 GAP 层可极大的降低特征维度，是全连接层参数不至于过多，保留了特征的强度信息，丢弃了特征的位置信息，因为是分类问题，对于位置信息不敏感，所以使用 GAP 层来降维效果很好。

我们可以对卷积层的输出加权平均，权重是 GAP 层到这个分类的权重，如下图所示：



这样就能得到一个 CAM(Class Activation Mapping)可视化图，也就是类激活图。简单来说，可以在最后一层的卷积层后面加一层。
参考链接：http://cnnlocalization.csail.mit.edu/

cam=(P−0.5)*w*outputcam=(P−0.5)*w*output

cam：类激活图 7*7

P：猫狗概率

output：卷积层的输出 2048*1

w：卷积核的权重 7*7*2048

```
#获取刚才训练模型的全连接权值
weights = model.get_weights()[0]

cnn_model = ResNet50(include_top = False, input_shape=(width, width, 3), weights = 'imagenet')
cnn_model = Model(cnn_model.input, cnn_model.layers[-2].output, name = 'resnet50')

inputs = Input((width, width, 3))
x = inputs
x = cnn_model(x)
cam = Conv2D(2, 1, use_bias = False, name = 'cam')(x)
model_cam = Model(inputs, cam)

x = GlobalAveragePooling2D(name = 'gap')(x)
x = model(x)
model_clf = Model(inputs, x)
```

载入权值的时候需要把 wieght 从（2048，2）reshape 成（1，1，2048，2），然后载入到 model_cam 的最后一个 1x1 卷积层里。

```
model_cam.layers[-1].set_weights([weights.reshape((1, 1, 2048, 2))])
```

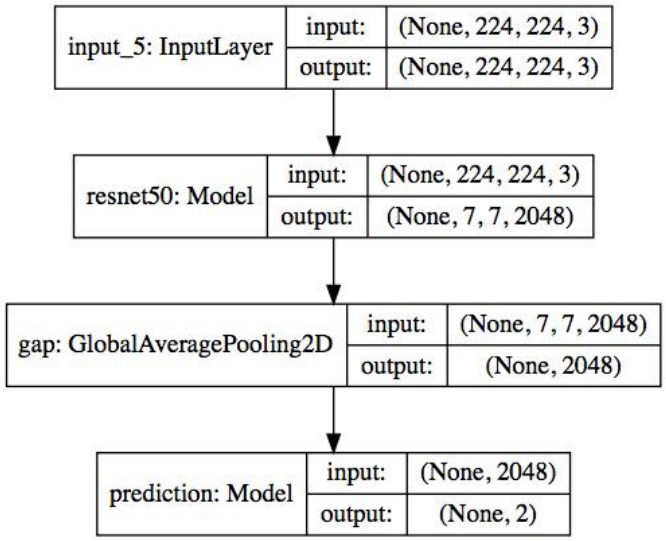
4.4 保存模型、可视化

```
model_clf.save('model_clf.h5')
model_cam.save('model_cam.h5')
```

可视化模型

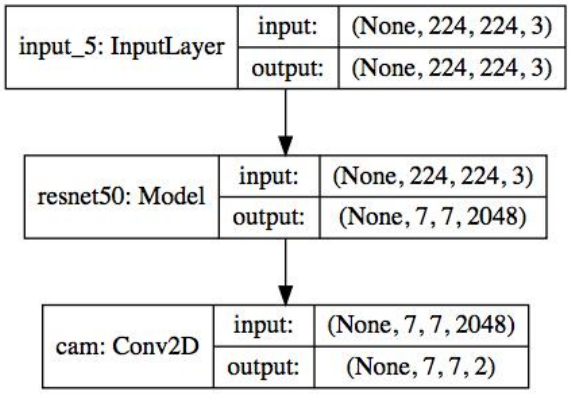
```
from keras.models import *
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

model_clf = load_model('model_clf.h5')
SVG(model_to_dot(model_clf, show_shapes=True).create(prog='dot', format='svg'))
```

```
/Users/heweihua/anaconda3/lib/python3.7/site-packages/keras/engine/saving.py:292: UserWarning
: No training configuration found in save file: the model was *not* compiled. Compile it manu
ally.
  warnings.warn('No training configuration found in save file: '
```




```
model_cam = load_model('model_cam.h5')
SVG(model_to_dot(model_cam, show_shapes=True).create(prog='dot', format='svg'))
```

```
/Users/heweihua/anaconda3/lib/python3.7/site-packages/keras/engine/saving.py:292: UserWarning
: No training configuration found in save file: the model was *not* compiled. Compile it manu
ally.
  warnings.warn('No training configuration found in save file: '
```



可视化测试

利用刚才搭建的两个模型尝试可视化，首先用 model_clf 预测这张图片是猫还是狗。 它会输出两个概率：第一个是猫的，第二个是狗的，我们取猫的概率，也就是 prediction[0,0]. 然后用 model_cam 输出两张 CAM 可视化的图，模型输出的 shape 是（1，7，7，2），可以简单的用 cam[0, :, :, 1 if prediction < 0.5 else 0]来提取对应类别的 CAM 图。 之后我们进行一些调整，首先将图片缩小 1/10，因为 CAM 图的取值范围大概在-5~30，平均值大约是 6，所以经过几次调整，除以 10

可视化效果比较好的数值，然后将数值限制在 0~1 之间。并转换为 Uint8(因为接下来的染色需要 Uint8 的格式)。 对 CAM 的染色使用 Opencv 的函数以及颜色条（见下图），我们选择了 COLORMAP_JET 样式。

参考链接：http://docs.opencv.org/trunk/d3/d50/group__imgproc__colormap.html

最后将染色的 heatmap 加载原图上，行形成最终的可视化效果图。

```
import matplotlib.pyplot as plt
import random
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# 用模型进行预测
plt.figure(figsize=(12, 14))
for i in range(16):
    plt.subplot(4, 4, i+1)
    img = cv2.imread('test/%d.jpg' % random.randint(1, 12500))
    img = cv2.resize(img, (224, 224))
    x = img.copy()
    x.astype(np.uint8)

    prediction = model_clf.predict(np.expand_dims(img, 0))
#     print (prediction) #猫编码成【1, 0】, 狗编码成【0, 1】 [9.408827e-10 1.000000e+00] = [0, 1] -
-> dog
    prediction = prediction[0, 0]
    if prediction < 0.5:
        plt.title('dog %.2f%%' % (100 - prediction*100))
    else:
        plt.title('cat %.2f%%' % (prediction*100))
    cam = model_cam.predict(np.expand_dims(img, 0))
    cam = cam[0, :, :, 1 if prediction < 0.5 else 0]

    # 调整 CAM 的范围
    cam -= cam.min()
    cam /= cam.max()
    cam -= 0.2
    cam /= 0.8
    cam = cv2.resize(cam, (224, 224))

    # 染成彩色
    heatmap = cv2.applyColorMap(np.uint8(255*cam), cv2.COLORMAP_JET)
    heatmap[np.where(cam <= 0.2)] = 0

    # 加在原图上
    out = cv2.addWeighted(img, 0.8, heatmap, 0.4, 0)

    # 显示图片
    plt.axis('off')
    plt.imshow(out[:,:,::-1])
```

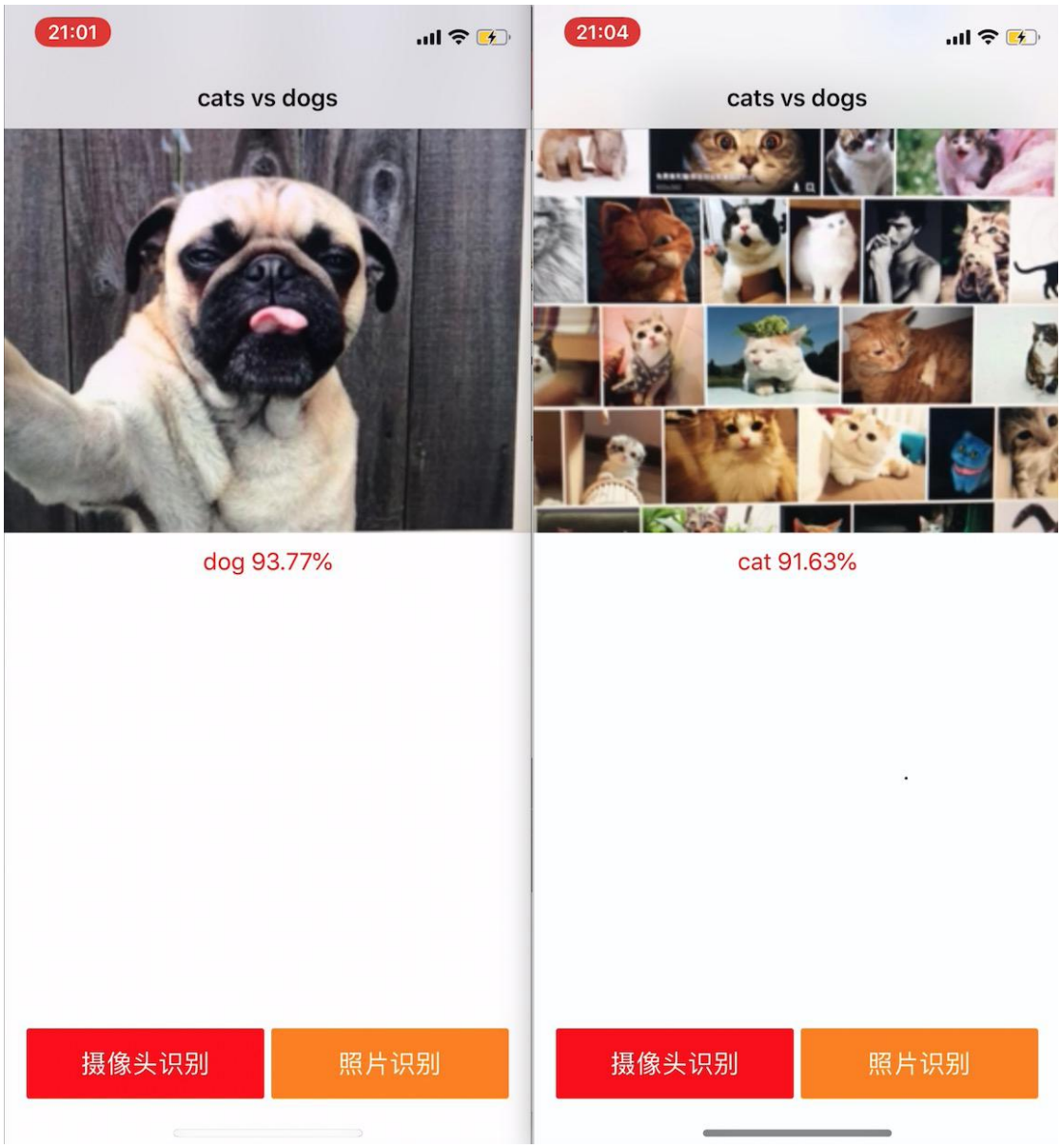
4.5 导出 mlmodel 模型文件，在 iOS 实现运行

导出 mlmodel 模型文件 在 iOS11 中，可以直接使用 Keras 模型，只需要使用苹果的模型转换库 Core ML Tools 将 Keras 以 HDF5 格式存储的模型转换为苹果使用的 mlmodel 格式即可。 参考链接：
https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml

```
from coremltools.converters.keras import convert
import coremltools

coreml_model = convert('model_clf.h5', blue_bias=103.939, green_bias=116.779, red_bias=123.68,
                       input_names=['image'], image_input_names='image', output_names='prediction')
coreml_model.author = 'weihua.he'
coreml_model.short_description = 'Dogs vs Cats'
coreml_model.license = 'MIT' #指定许可证
coreml_model.input_description['image'] = 'A 224x224 Image.'
coreml_model.output_description['prediction'] = 'The probability of Dog and Cat.'
coreml_model.save('model_clf.mlmodel')

#CAM模型
coreml_model = convert('model_cam.h5', blue_bias=103.939, green_bias=116.779, red_bias=123.68,
                       input_names=['image'], image_input_names='image', output_names='cam')
coreml_model.author = 'weihua.he'
coreml_model.short_description = 'Dogs vs Cats'
coreml_model.license = 'MIT'
coreml_model.input_description['image'] = 'A 224x224 Image.'
coreml_model.output_description['cam'] = 'The cam Image.'
coreml_model.save('model_cam.mlmodel')
```

将导出的模型，放到 xcode 新建的工程中使用即可，摄像头识别使用的是 opencv 的摄像头。 iOS 那部分代码由于篇幅问题且非重点，见 github:https://github.com/bjheweihua/cats_vs_dogs

在 iphone 手机上运行 App， 摄像头识别效果如下图：



V. 结果分析

特征组合相对单模型能够得到一些改善。相对单模型，特征提取后直接分类输出，特征 组合还叠加了一个额外的分类网络，实际参与计算的网络层要大于单模型，因此，针对特征 组合所获改善这一结果，有多少是来自于特征组合本身，有多少是来自于额外的分类网络， 还是一个有待于研究的问题。

单个 ResNet50 模型 8 次迭代训练结果：

- 训练集 loss：0.0356 - acc: 0.9878；验证集 val_loss：0.0263 - val_acc: 0.9910。

两个模型 Xception-InceptionResNetV2 融合 8 次迭代训练结果：

- 训练集 loss：0.0111 - acc: 0.9970；验证集 val_loss：0.0165 - val_acc: 0.9958。

使用 Xception，InceptionV3，InceptionResNetV2 这三个模型进行迁移学习 8 次迭代训练结果：

- 训练集 loss：0.0096 - acc: 0.9972； 验证集 val_loss：0.0170 - val_acc: 0.9960。

两个模型组合比单模型好，三个模型融合比两个模型好。

Xception 有 126 层，InceptionV3 有 159 层，InceptionResNetV2 有 572 层，不同的卷积核有各种各样的的组合，可更好抽取图片中的泛化特征；这样既提高分类的准确率，又降低模型的过拟合风险，一般情况下网络越深，准确率越高。

Xception，InceptionV3，InceptionResNetV2 这三个模型进行组合迁移学习，是一个比较好的组合，效果比先单个神经网络模型效果好。这里利用了 bagging 的思想，通过多个模型处理数据并进行组合，可以有效降低模型的方差，减少过拟合程度，提高分类准确率。

如果 ResNet50，Xception，InceptionV3，InceptionResNetV2 四个模型融合，从结果上提升效率不高，反而复杂度成本增大了。

VI. 总结感想

本项目使用迁移学习和融合模型完成，站在巨人的肩膀上可以轻松的学习到很优秀的权重参数。这过程中在自己 MAC 上折腾好几周，但是只能跑一个模型，机器发热严重风扇呼呼的一直转，感觉电脑都要冒烟了，好几次导致电脑自动关机了。然后，跑任务的时候把电脑所有的程序都关掉，跑了足足 8 多小时，终于能全部跑通。但是，使用多模型融合时就不行了，电脑直接提示内存警告，直接宕机，在网上找了 AWS、腾讯的 GPU，都不好用，腾讯的还收费那么贵。终于，通过一个同事知道公司内部有免费服务器资源，通过申请公司的 GUP 服务器才把项目完成。做这个项目，最大的困难就是使用云计算平台去训练，熟练了整个云计算平台的基本使用流程。

该项目中使用了 Xception，InceptionV3，gap_InceptionResNetV2 这三个模型进行了提取特征向量，然后将特征向量直接拼接，忽略了特征之间的位置关系。除了这三个模型，还可以增加更多新的模型，但是效果未必能提升，或者使用 stacking 的方法进行模型融合，进一步降低方差，提高分类的准确率。或者使用更强大的分类器进行训练；这过程中尝试过使用 EfficientNets，预训练好的模型 EfficientNet 提供 B0 到 B5， B5 的精确度 83.2%，B6,B7 还没提供，由于时间关系没继续走这条路。其实为了更高效的解决一些问题，应该适用一些成本比较低，但准确率高，效率更高的体积小简单一点的模型更合适，如 EfficientNets 模型从宽-高-深度动态调整一个合适的参数来提供模型的准确率。

最开始做这个项目，先用 TensorFlow 来实现，发现有很多坑，mac 尝试了很多次没能跑起来，所以放弃了。Keras 对于初学者还是很友好，使用简单而且稳定，虽然 Keras 底层实现也是 TensorFlow 框架，但是 Keras 对 TensorFlow 封装的更好，更友好。除非 Keras 没提供的一些 TensorFlow 的方法，这时我们可使用 TensorFlow 底层方法。

VII. 参考文献

[1] Karen Simonyan and Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE- SCALE IMAGE RECOGNITION. At ICLR,2015.

[2] [译] Deep Residual Learning for Image Recognition (ResNet)

[3] 手把手教你如何在 Kaggle 猫狗大战冲到 Top2%：https://yangpeiwen.com/dogs-vs-cats-2

[4] Keras 做图片分类（四）：迁移学习--猫狗大战实战：https://zhuanlan.zhihu.com/p/51889181

[5] Kaggle 猫狗大战准确率 Top 2%webapp 部署：https://www.jianshu.com/p/1bc2abe88388

[6] Keras 中文文档：https://keras.io/zh/applications

[7] 毕业设计 Dogs vs Cats For Udacity P7 (异常值检验)：https://zhuanlan.zhihu.com/p/34068451

[8] 面向小数据集构建图像分类模型：https://keras-cn-docs.readthedocs.io/zh_CN/latest/blog/image_classification_using_very_little_data

[9] 杨培文 胡博强. 深度学习技术图像处理入门. 北京：清华大学出版社，2018（2019.4 重印）.

[10] [美]Ian GoodFellow [加]Yoshua Bengio [加]Aaron Courvile 著 赵申剑等人译. 深度学习. 北京：人民邮电出版社，2017.8（2017.12 重印）.

[11] efficientnet：https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet