机器学习工程师纳米学位毕业项目

猫狗大战

高云飞

2017年6月3日

1 定义

1.1 项目概述

项目是根据一系列猫狗的图片，训练一个模型，识别出图片中是狗的概率。

项目属于机器视觉范畴的图像识别技术，该技术是信息时代的一门重要技术，其目的是为了让计算机代替人类处理大量的物理信息。比如说让计算机从人群中迅速识别出通缉犯，对癌症准确侦测，自动驾驶对于周围环境的探测等。

项目使用的模型是卷积神经网络(Convolutional Neural Network，以下简称CNN)。该模型早在1997年便用来解决字符识别问题，随着近年来硬件性能的提高，其对复杂图像的处理也成为了可能。2012年，Hinton课题组为了证明深度学习的潜力，首次参加ImageNet[1]图像识别比赛，其通过构建的CNN网络AlexNet一举夺得冠军，且碾压第二名（SVM方法）的分类性能。也正是由于该比赛，CNN吸引到了众多研究者的注意。

项目中使用到的相关数据是来自于kaggle的Dogs vs. Cats Redux: Kernels Edition项目。数据是JPG格式的图片，分为train和test两部分。train部分数据为一系列通过文件名加过标签的图片，共有25000张，用来训练模型，其中猫和狗的图片各有12500张，这些图片没有统一尺寸，有些图片过小或者过于模糊，但是尺寸都在512 x 512像素以内，多数场景比较简单，为一只猫或者狗，也有少数为多只猫狗，甚至有一些图片中没有猫狗，需要在训练开始前，去除掉这些异常数据；test部分数据为一系列未添加标签的图片，共有12500张，用来测试模型性能。

1.2 问题陈述

项目要解决的问题是，让计算机具备判断一张图片是狗的图片的概率的能力。

项目将采用CNN，对训练集中的图片进行训练。先将test数据下的异常数据剔除，将剩余的数据进行resize操作，统一尺寸，然后打乱数据，再分为train set和validation set两部分，其中train set用于对模型的训练，而validation set用于对模型的验证。

期望通过训练，能够得出一个有效的模型，对test中的图片，能够得出较为准确的判断图片为狗的概率。

1.3 评价指标

模型性能好坏，kaggle通过一个公式来评价

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

其中的参数含义如下：

- n 数据集的大小
- $\hat{y}_i$ 通过模型预测出的图片为狗的概率
- y_i 该图片若为狗则取值为1，否则为0
- $log()$ 以e为底的自然对数函数

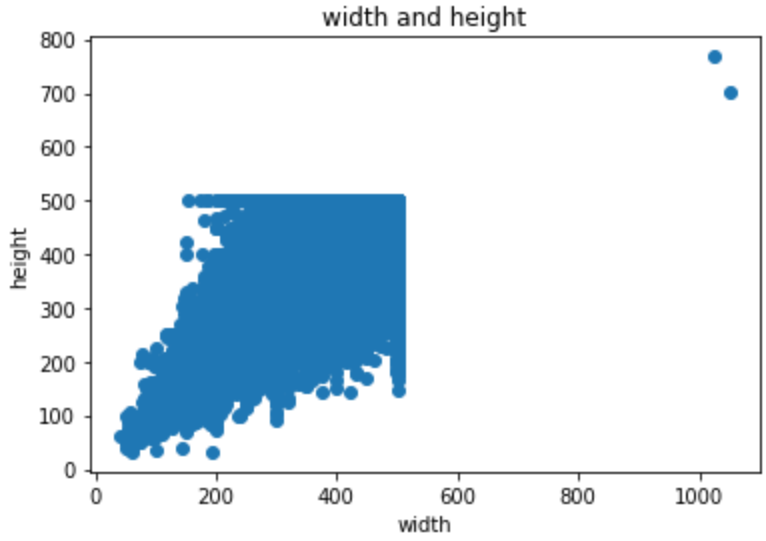
LogLoss值越小，则模型性能越好。

该对模型性能的验证方法，可以用在验证集上，通过对验证集的性能测试，可以得知模型性能好坏，然后再对测试集的数据进行预测，结果提交至kaggle。

2 分析

2.1 数据的探索

我们先将train数据集的图片尺寸的散点图绘制出来，如下：



1. 我们将图片的尺寸分布，通过散布图绘制出来。可以看出，大多数图都在500*500像素左右大小以内，只有两个图片远超过此范围。

2. 随机挑选若干张图片，发现有些图片是不适合用来训练的，因为有些图片过于模糊、场景过于复杂、不包含猫狗或者同时包含猫狗，所以图片并不都都能用来训练。

鉴于这两点原因，所以需要对数据进行预处理，处理方式如下：

1. 统一尺寸。

2. 剔除异常数据。

完成以上操作后，需要把剩余的合理数据，打乱后分成两部分，即train set和validation set，其中train set占9/10，用于训练，validation set占1/10用于验证。

经以上处理，现在共有三个数据集：train set, validation set和test set。

2.2 算法和技术

2.2.1 分类算法

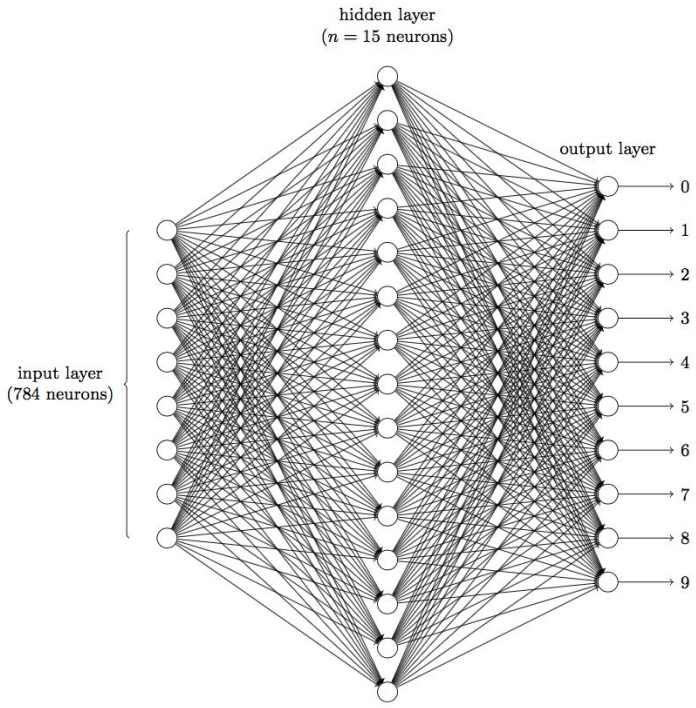
通过以上的分析，可以看出，本项目面对的问题，本质就分类问题。用于分类的算法有很多，比如Bayes, Decision Tree, SVM, KNN, Logistic Regression等。

分类算法的一般流程是，先有一定数量的的feature和label后，再用这些数据来训练模型，模型训练好以后，在用这个模型在新的feature上预测新的label。但是以上方法面临着一个问题就是，他们必须有明确的feature和label，这里所谓的“明确的”，是指特征的数量，以及每个特征能取哪些值等。

但是这样的条件，在图片分类上是不现实的，你无法看出一张图片有哪些特征。所以，以上提到的传统分类方法，在图片分类上，是不能适用的。

那么为什么要用卷积神经网络(CNN)而不是普通的神经网络呢？

我们知道，图像是由一个个像素点构成的，每个像素点又有RGB三个颜色通道，如果一个图像的尺寸是28*28的灰度值图像，假如我们使用全连接的网络结构，即，网络中的神经与与相邻层上的每个神经元均连接，那就意味着我们的网络有28 * 28 =784个神经元，hidden层采用了15个神经元，那么简单计算一下，我们需要的参数个数(w和b)就有：784*15*10+15+10=117625个，这个参数太多了，随便进行一次反向传播计算量都是巨大的，从计算资源和调参的角度都不建议用传统的神经网络。



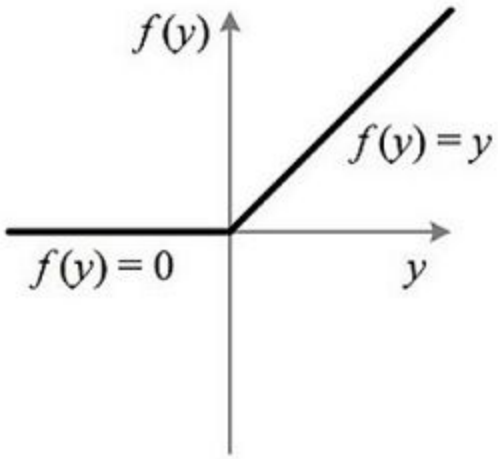
我们已经看到了传统神经网络的劣势，而我们要用到的CNN算法与以上算法不同。因为图像本身具有“二维空间特征”，即有局部特性，通俗的说，假如我们有一只猫的图片，可能我们看到眼睛、鼻子、耳朵等部位，就能确定这是一只猫的图片，并不需要把图片上的每个像素点都要严格的看一遍，而CNN就能做到，通过一层一层的卷积，提取到最典型的特征，进行识别，CNN提取的特征可以达到更好的效果，而且CNN不需要将特征提取和分类训练两个过程分开，CNN在训练时，就自动提取了最有效的特征。因为具备诸多优点，在图像分类领域，CNN已经成为了主流的算法。

在了解完卷积神经网络后，再来了解以下几个概念：激活函数、池化、Dropout

激活函数

激活函数，是每一个神经元中，决定是否向下一个神经元传递的函数。

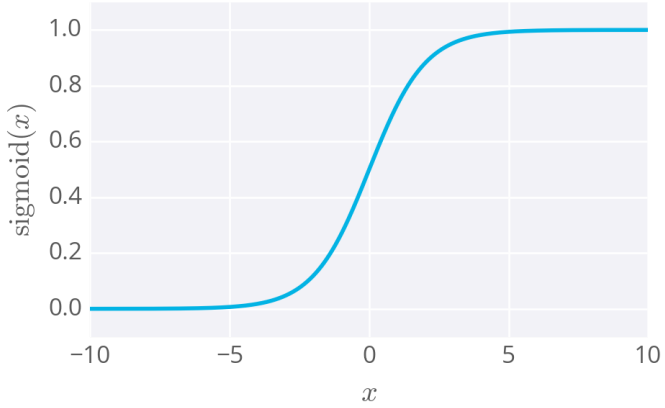
在CNN的每一个卷积层中，都会使用一个激活函数，这里我们使用ReLU函数，其函数公式如下：



$$ReLU(x) = \begin{cases} x & if\, x > 0 \\ 0 & if\, x \le 0 \end{cases}$$

ReLU函数是一个分段函数，在x<=0时，返回0，而在x>0时，返回x。

而另外一个较为常用的激活函数为sigmoid函数，其公式如下：



$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

Sigmoid函数在整个定义域内都是按照一个函数sigmoid返回的，sigmoid函数，对中央区的信号增益较大，对两侧区的增益小。这很像神经元的状态，中央区为神经元的兴奋态，两侧区为神经元的抑制区。

但是ReLU相比sigmoid函数，有些以下优点：

1. sigmoid函数，计算量大，尤其是反向传播求误差梯度时，而采用ReLU函数，整个过程的计算量节省很多；
2. 对于深层网络，sigmoid函数反向传播时候，很容易出现梯度小时的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失），从而无法完成深层网络的训练，而ReLUz则能保证按照一定速率收敛；
3. ReLU会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生；

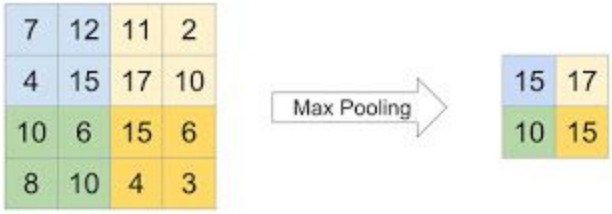
因而我们在卷积层中，使用ReLU函数作为激活函数。

池化

池化是降低特征维度，保留有效信息，一定程度上避免过拟合的一种手段。

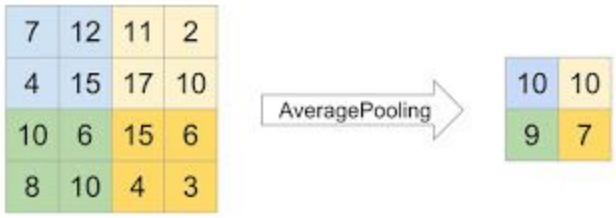
池化有多种方式，比如最大池化、平均池化等。

最大池化



如上图所示，假设有一个2x2的滤波器，以步长2移动，则每次取出最大的那个值，其他值丢弃。

平均池化



如上图所示，与最大池化不同，平均池化取的是平均值。

除去这两个常用的池化以外，还有一种池化方法，叫做全局平均池化。

全局平均池化

全局平均池化，是对每一个特征图，进行全局均值池化，每张特征图只得到一个输出，这样做，可以大大减小网络参数，避免过拟合，让卷积结构更简单。

Dropout

Dropout是一项非常重要的正则化技术，在防止过拟合上，有不凡效果。

Dropout是通过再深度学习网络的训练过程中，对神经网络单元，按照一定的概率，将其暂时从网络中丢弃来实现的。

在CNN算法中，有很多成熟的模型可供选择，比如最早的LeNet5, AlexNet, VGGNet, InceptionNet, ResNet。

LeNet5

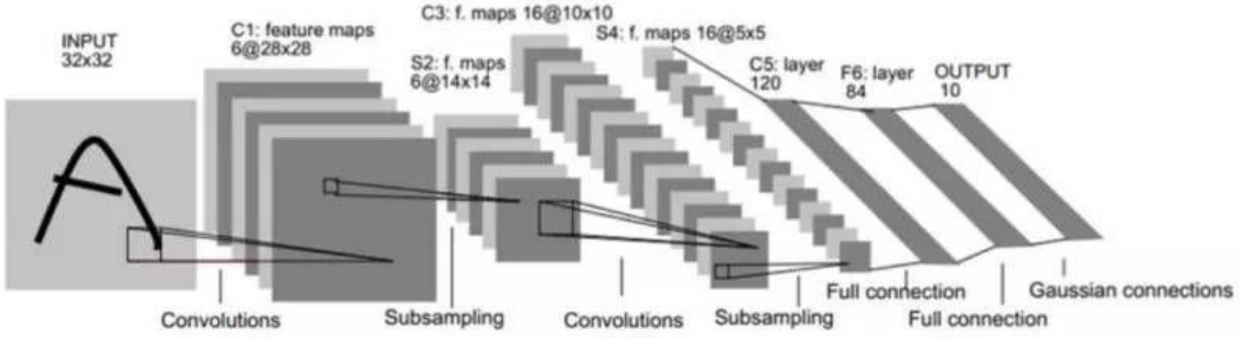
LeNet5诞生于1994年，是最早的深层卷积神经网络之一，并推动了深度学习的发展。LeNet5认为，可训练参数的卷积层是一种用少量参数在图像的多个位置上提取相似特征的有效方式，这和直接把每个像素作为多层神经网络的输入不同。像素不应该被使用在输入层，因为图像具有很强的空间相关性，而使用图像中独立的像素直接作为输入则利用不到这些相关性。

LeNet5当时的特性有如下几点。

- 每个卷积层包含三个部分：卷积、池化和非线性激活函数
- 使用卷积提取空间特征
- 降采样（Subsample）的平均池化层（Average?Pooling）
- 双曲正切（Tanh）或S型（Sigmoid）的激活函数
- MLP作为最后的分类器

- 层与层之间的稀疏连接减少计算复杂度

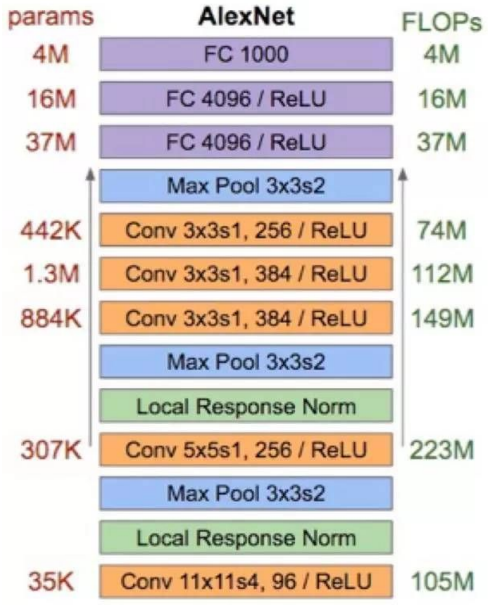
LeNet5中的诸多特性现在依然在state-of-the-art卷积神经网络中使用，可以说LeNet5是奠定了现代卷积神经网络的基石之作。Lenet-5的结构下图所示。



AlexNet

AlexNet是现代深度CNN的奠基之作。可以算是LeNet的一种更深更宽的版本。AlexNet中包含了几个比较新的技术点，也首次在CNN中成功应用了ReLU、Dropout和LRN等Trick。同时AlexNet也使用了GPU进行运算加速。

AlexNet包含了6亿3000万个连接，6000万个参数和65万个神经元，拥有5个卷积层，其中3个卷积层后面连接了最大池化层，最后还有3个全连接层。AlexNet以显著的优势赢得了竞争激烈的ILSVRC 2012比赛，top-5的错误率降低至了16.4%，相比第二名的成绩26.2%错误率有了巨大的提升。



VGGNet

VGGNet是牛津大学计算机视觉组（Visual?Geometry?Group）和Google?DeepMind公司的研究员一起研发的的深度卷积神经网络。VGGNet探索了卷积神经网络的深度与其性能之间的关系，通过反复堆叠3*3的小型卷积核和2*2的最大池化层，VGGNet成功地构筑了16~19层深的卷积神经网络。VGGNet相比之前state-of-the-art的网络结构，错误率大幅下降，并取得了ILSVRC 2014比赛分类项目的第2名和定位项目的第1名。

VGGNet论文中全部使用了3*3的卷积核和2*2的池化核，通过不断加深网络结构来提升性能。下图所示为VGGNet各级别的网络结构图，和每一级别的参数量，从11层的网络一直到19层的网络都有详尽的性能测试。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

InceptionNet

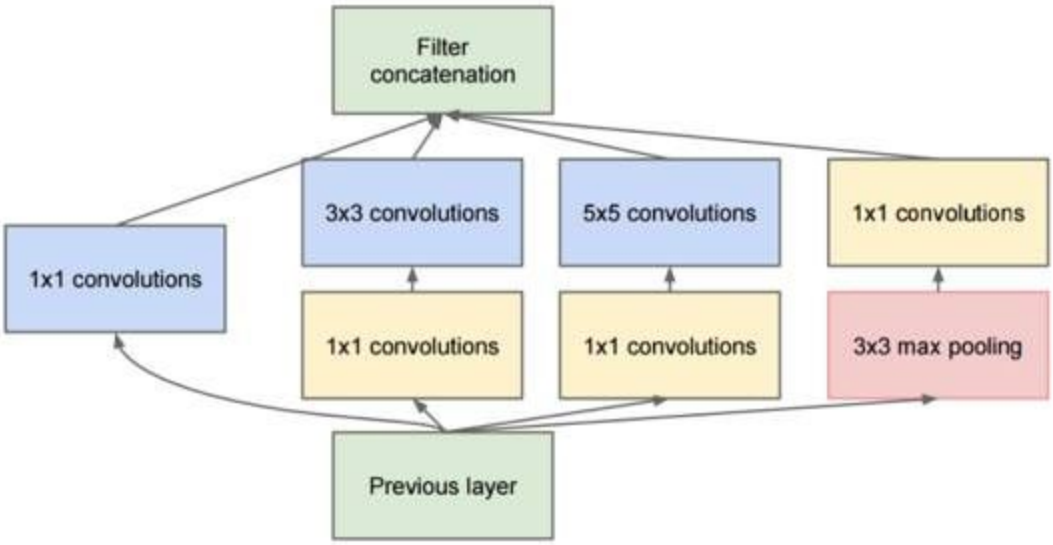
Google Inception Net首次出现在ILSVRC?2014的比赛中（和VGGNet同年），就以较大优势取得了第一名。那届比赛中的Inception Net通常被称为Inception V1，它最大的特点是控制了计算量和参数量的同时，获得了非常好的分类性能——top-5错误率6.67%，只有AlexNet的一半不到。

Inception V1有22层深，比AlexNet的8层或者VGGNet的19层还要更深。但其计算量只有15亿次浮点运算，同时只有500万的参数量，仅为AlexNet参数量（6000万）的1/12，却可以达到远胜于AlexNet的准确率，可以说是非常优秀并且非常实用的模型。

Inception V1降低参数量的目的有两点：第一，参数越多模型越庞大，需要供模型学习的数据量就越大，而目前高质量的数据非常昂贵；第二，参数越多，耗费的计算资源也会更大。

Inception V1参数少但效果好的原因除了模型层数更深、表达能力更强外，还有两点：一是去除了最后的全连接层，用全局平均池化层（即将图片尺寸变为1*1）来取代它。全连接层几乎占据了AlexNet或VGGNet中90%的参数量，而且会引起过拟合，去除全连接层后模型训练更快并且减轻了过拟合。

二是Inception V1中精心设计的Inception?Module提高了参数的利用效率，其结构如图10所示。这一部分也借鉴了Network?In?Network的思想，形象的解释就是Inception?Module本身如同大网络中的一个小网络，其结构可以反复堆叠在一起形成大网络。



经过多年的不断改进迭代，InceptionNet已经进化出了多个版本。包括上面提到的Inception V1(top-5错误率6.67%)，Inception V2(top-5错误率4.8%)，Inception V3(top-5错误率3.5%)，Inception V4(top-5错误率3.08%)。

ResNet

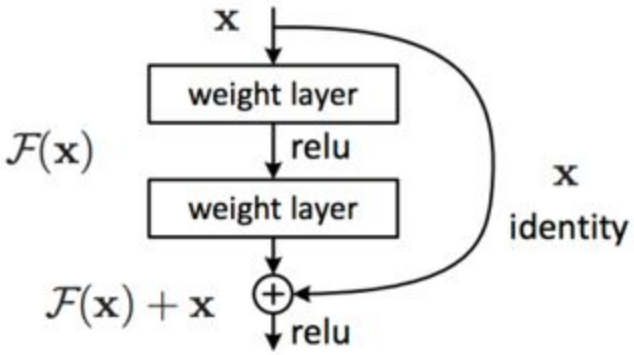
ResNet（Residual Neural Network）由微软研究院的Kaiming He等4名华人提出，通过使用Residual Unit成功训练152层深的神经网络，在ILSVRC 2015比赛中获得了冠军，取得3.57%的top-5错误率，同时参数量却比VGGNet低，效果非常突出。

ResNet的结构可以极快地加速超深神经网络的训练，模型的准确率也有非常大的提升。

ResNet最初的灵感出自这个问题：在不断加神经网络的深度时，会出现一个Degradation的问题，即准确率会先上升然后达到饱和，再持续增加深度则会导致准确率下降。

这并不是过拟合的问题，因为不光在测试集上误差增大，训练集本身误差也会增大。假设有一个比较浅的网络达到了饱和的准确率，那么后面再加上几个的全等映射层，起码误差不会增加，即更深的网络不应该带来训练集上误差上升。

而这里提到的使用全等映射直接将前一层输出传到后面的思想，就是ResNet的灵感来源。假定某段神经网络的输入是x，期望输出是H(x)，如果我们直接把输入x传到输出作为初始结果，那么此时我们需要学习的目标就是F(x)=H(x)-x。



2.2.2 算法选择

经过以上介绍，可以解决本项目问题的CNN模型有很多，而且在图像识别上的表现，InceptionNet与ResNet无论是在参数数量还是在性能上，都明显好于VGGNet、AlexNet等，所以

在使用的模型上，从InceptionNet与ResNet中选择。由于InceptionNet错误率表现稍好于ResNet，则选择InsceptionNet作为项目中使用的模型。

我们选择使用InceptionNet模型，并进行**迁移学习**，在猫狗的新数据上进行训练，修改分类器，使之从输出多个种类的概率的分类器，变成只输出狗的概率。

那么什么是**迁移学习**？迁移学习(Transfer learning) 顾名思义就是就是把已学训练好的模型参数迁移到新的模型来帮助新模型训练。考虑到大部分数据或任务是存在相关性的，所以通过迁移学习我们可以将已经学到的模型参数（也可理解为模型学到的知识）通过某种方式来分享给新模型从而加快并优化模型的学习效率不用像大多数网络那样从零学习（starting from scratch, tabula rasa）。

由于我们的猫狗图片数据集，是ImageNet数据集的子集，所以用迁移学习这种方式，利用现有的成熟的模型来训练新的模型是非常便利的。

2.2.3 框架选择

机器学习的框架有很多，比Tensorflow, Caffe, Theano, PyTorch等。

Tensorflow是谷歌的开发的机器学习框架，并于2015年开源。Tensorflow对于快速执行基于图形的计算非常有用，灵活的API可以通过GPU支持的架构在多个设备之间部署模型。Tensorflow由C++编写，提供用户友好的Python接口。另外，Tensorflow拥有强大的用户社区和网络教程，这对于使用Tensorflow提供了非常有利的条件。

Caffe是由伯克利人工智能研究小组和伯克利视觉和学习中心开发。内核由C++编写，提供Python和Matlab接口。虽然Caffe对训练或者微调前馈分类模型非常有用，但是在研究中使用的并不多。

Theano是一个快速数值计算的python库，由蒙特利尔大学开发并开源，与NumPy紧密结合，可以在CPU或者GPU上运行。但是该框架已经与2017年11月停止维护，不在开发。

PyTorch是由Facebook人工智能研究组开发的。PyTorch是为了解决其前身Torch框架的编程语言困境应运而生的。因为Torch编程语言为比较小众的Lua，严重阻碍了该框架的发展，因此，PyTorch的开发人员采用了Python命令编程风格。同时它还支持动态计算图，这一特性使得它对做时间序列以及自然语言处理数据相关工作的研究人员和工程师很有吸引力。

由于Theano不再继续被开发，Torch是以不为许多人熟悉的Lua语言编写的，Caffe还处于它的早熟阶段，TensorFlow和PyTorch成为大多数深度学习实践者的首选框架。虽然两个框架都

使用Python，但是仍然存在一些差异性，比如PyTorch的功能并不如Tensorflow完善，而且社区丰富程度也不如Tensorflow，这就意味着当编程过程中遇到问题，Tensorflow会更容易找到解决方案，在扩展性上，Tensorflow要更胜一筹。所以，项目中将使用Tensorflow。

Tensorflow也不尽全是优点，由于其使用繁琐，我们将采用Keras设置Tensorflow后端的形式，间接的使用Tensorflow。

那么什么是Keras呢？

Keras是一个高层神经网络API，Keras由纯Python编写而成并基Tensorflow、Theano以及CNTK后端。Keras 为支持快速实验而生，能够把你的idea迅速转换为结果。而且Keras的API设计的十分简洁友好，模块化设计使得拥有极强的扩展性。因此，在简易和快速实现的项目上，Keras成了首选。

2.3 基准指标

在ImageNet比赛中，历代模型冠军的结果衡量指标是以top-5错误率为基准的，但是因为本项目只需要预测图片为狗的概率，所以，按照top-5的错误率衡量是不合适的。

在**1.3 评价指标**中，提到了一个LogLoss函数，我们以这个函数作为评价指标，这也是Kaggle所要求的。当我们训练好模型，在test set上做出预测，将预测结果提交到kaggle上，能进入Leaderboard排名的前10%便可以认为模型足够优秀。

3 方法

3.1 数据预处理

读取训练数据，需要对数据进行以下处理：

1. 统一数据尺寸；
2. 正则化数据；
3. 剔除异常数据；
4. 分离训练集train set和验证集validation set。

3.1.1 统一数据尺寸并正则化数据

针对每一张图片，我们通过下面代码读取与操作。

```
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input,
decode_predictions

def readImage(path):
    img = image.load_img(path, target_size=(image_size, image_size))
    x = image.img_to_array(img)
#     x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x
```

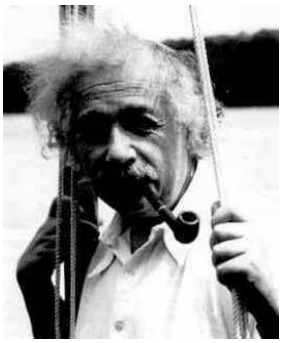
通过以上代码，将图片数据以image_size*image_size的尺寸读入内存，并通过inception_v3的preprocess_input预处理方法，做正则化处理，查看的preprocess_input源码得知，对每个像素点x的处理是x/127.5 - 1。

同样，针对每一张图片，我们通过其文件名，可以知道其是狗还是猫，如果是狗，则label为1，如果为猫，则label为0，从而得到了一个label的数组。

3.1.2 剔除异常数据

通过对数据的图片的观察，会发现，有些图片是不适合参与训练的。比如说以下图片：





以上只是举例了三张图片而已，我们可以看出，这些图片存在一些问题，场景复杂，图片模糊，甚至图片内容既不包含猫又不包含狗。

那么这些图片该怎样剔除呢？

我们可以利用预训练模型，按照top-N(N取1000以内的任意值)的方式去剔除。在以往的ImageNet竞赛中，已经有足够优秀的预训练模型已经权重文件，可以很好的帮我们识别图片是否

为异常值。以Top-5为例，通过InceptionV3这个预训练模型，加载weights为imagenet，得出某一图片最有可能的5个物体可能，并给出其编号与概率，查询这5个物体，是否在猫狗的分类集合里。

具体代码如下：

```
from keras.applications.inception_v3 import InceptionV3

inceptionV3 = InceptionV3(weights='imagenet')  # 引入InceptionV3模型
```

```
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input,
decode_predictions
import numpy as np

def getPreds(img_path): # 获取模型对某一张图片的预测top-50
    img = image.load_img(img_path, target_size=(image_size, image_size))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    preds = inceptionV3.predict(x)
    result = decode_predictions(preds, top=50)[0]
#     print('Predicted:', result)
    return result
```

```
def isCat(preds): # 判断是否为猫
    for item in preds:
        if item[0] in cats:
            return True
    return False
def isDog(preds): # 判断是否为狗
    for item in preds:
        if item[0] in dogs:
            return True
    return False
def isValid(img_path): # 判断是否为合理的图片
    label = getLabel(img_path)
```

```
    preds = getPreds(img_path)
    return {
        0:isCat(preds),
        1:isDog(preds)
    }[label]
```

通过这些代码，我们就可以提出以top-50为标准的异常值，共有164张图片被剔除掉。

3.1.3 数据集分离

将经过以上步骤的数据，分成随机两个部分，训练集train set和验证集validation set。因为train set和validation set是随机的从同一个数据集里分离出的两部分，所包含的特征是一样的，所以能够有效的通过validation set来验证模型在train set上的训练是否有效。

```
import numpy as np
from sklearn.model_selection import train_test_split

total_feature, total_label = readFeatureAndLabel(valid_files)
total_feature = np.array(total_feature)

total_label = np.array(total_label)

train_feature, validation_feature, train_label, validation_label =
train_test_split(total_feature, total_label, test_size=0.1, random_state=1)
```

我们分离出的数据命名为train_feature, validation_feature, train_label, validation_label。

3.2 执行过程

3.2.1 Inception预训练模型训练

在**2.2.1算法选择**中，我们已经提到，使用Inception模型来解决此项目中遇到的问题。Keras中已经提供了Inception V3预训练模型。

查阅Keras文档，按照其要求的方式，以InceptionV3作为基准模型，应用全局平均池化，由于我们只需要图片为狗的概率，所以将InceptionV3的输出函数设置为sigmoid函数。代码示例如下：

```
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras import backend as K

# 创建预训练模型
base_model = InceptionV3(weights='imagenet', include_top=False)
# 添加平均池化层
x = base_model.output
x = GlobalAveragePooling2D()(x)
# 由于我们只需要狗的概率，所以设置参数classes为1，输出函数为sigmoid
predictions = Dense(1, activation='sigmoid')(x)
# 创建模型
model = Model(inputs=base_model.input, outputs=predictions)

opt = Adam(lr=0.0005, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

# train the model on the new data for a few epochs
history = model.fit(train_feature, train_label, batch_size=batch_size, epochs=20,
validation_data=(validation_feature, validation_label))
```

经过上述代码运行，我们可以得到在每一轮训练中train-loss、train-acc、val-loss、val-acc四个数值。并记录在history中。其曲线变化过程如下图：

由上面两图中可以看到训练过程中loss与accuracy的走势。虽然loss与accuracy无论是训练还是验证，都是区域整体下降的趋势。训练曲线，loss能够下降到大概0.02，accuracy能提升到0.99到1.0之间；验证曲线，波动很大，loss最低没低于过0.05，acc最高在0.98，因为其并不稳定，所以该模型性能不佳。

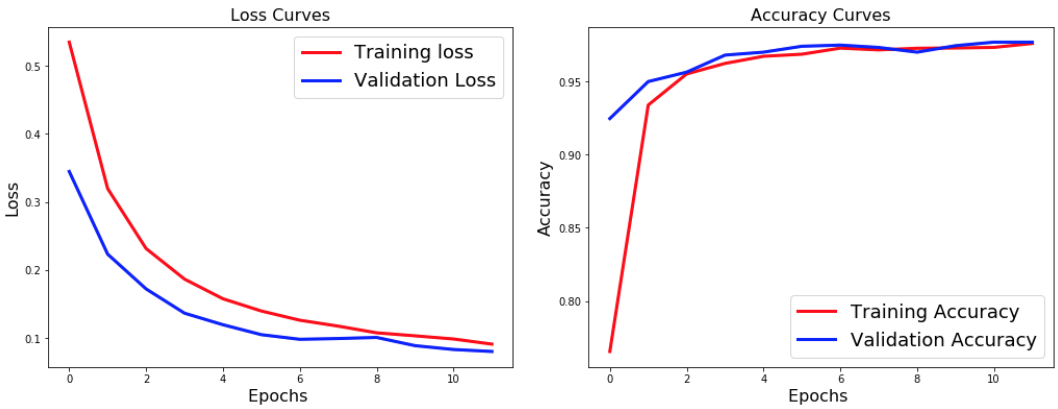
针对上面的情况，我们需要作出一些改进。

如果波形波动太大，可以通过降低学习率的方式、冻结部分层等方式来变化，使之变得曲线平滑。

在上述代码中，我们将全部层冻结，在model模型创建后，我们通过这样的代码，去冻结其中的层。

```
for layer in base_model.layers:
    layer.trainable = False
```

再次运行代码后，我们可以得到这样的曲线图。

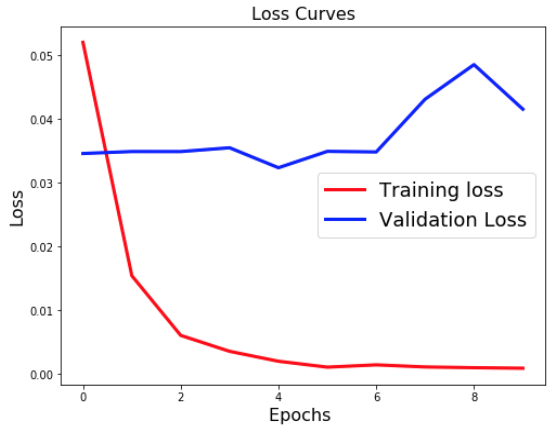


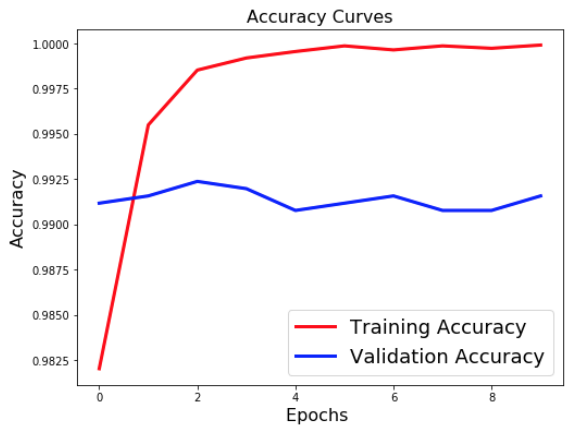
我们通过曲线图，可以看出，曲线平滑了许多，但是随之带来了一些问题，就是loss上升，accuracy下降了。训练曲线，loss会到0.1上下，accuracy在接近0.95的位置就难以攀升了；验证曲线，loss在0.05上下徘徊，accuracy在0.98上下徘徊。

对于这样的现实，我们需要继续优化。现在我们决定把部分层解除冻结，再次训练。

```
for i, layer in enumerate(base_model.layers):
    print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model.layers[:249]:
    layer.trainable = False
for layer in model.layers[249:]:
    layer.trainable = True

# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
# loss='sparse_categorical_crossentropy'
from keras.optimizers import SGD
# Adam(lr=0.001)
opt2 = SGD(lr=0.0008, momentum=0.99, decay=0.00002, nesterov=True)
model.compile(optimizer=opt2, loss='binary_crossentropy',
metrics=['accuracy'])

# we train our model again (this time fine-tuning the top 2 inception
blocks
# alongside the top Dense layers
history2 = model.fit(train_feature, train_label, batch_size=batch_size,
epochs=10, validation_data=(validation_feature, validation_label),
class_weight='auto')
```

经过这样的处理，我们得到了较为理想的曲线。





看上图，我们可以看出，训练的loss已经降低到接近于0了，而accuracy接近了1；验证的loss降低到0.04一下，accuracy提到到0.9以上。可以看出有较大提升。

经过以上的优化处理，我们在测试集上的LogLoss上的Score减小到了0.04458，排名在30左右。

3.3 完善训练

通过保存最佳状态时候的模型，保存了成绩最好的模型。

```
callback = ModelCheckpoint('best_point.h5', monitor='val_acc', verbose=0,
save_best_only=True, save_weights_only=False, mode='auto', period=1)
```

通过上面的回调，我们可以将val_acc最好状态的模型保存下来，并用于预测test数据，最终LogLoss得分0.04202，排名20上下。

4 结果

4.1 模型的评价与验证

我们已经得到了符合标准的模型，我们的模型，是通过对预训练模型的InceptionV3的迁移学习配合fine-tune完成的。替换其输出函数为sigmoid，并在新的数据上重新训练，我们得到了在loss和accuracy都比较满意的模型。

4.2 合理性分析

训练好的模型，通过LogLoss函数，在validation set上，得到了4.046025048481314e-06的分数，只是一个相较来说，比较低的一个分数了，因此，我们可以利用这个模型，对test set数据进行预测，并将预测结果提交到kaggle上验证我们的有效性。

最终得分得到了LogLoss分数为0.05833，排名110左右，已经进入前10%，符合项目的要求的及格标准，可以认为是，比较有效的模型了。

5 项目结论

5.1 对项目的思考

在做这个项目的时候，一开始是没有头绪的，因为这第一次，比较完整的做一个项目，在以往的项目中，都是有现有的文件，我们像做填空一样，只需要在关键的位置填上关键的代码就好了，这次不同，是从头到尾完整来做。

在开始项目以前，草草看了一下项目要求，本以为很简单，因为感觉跟P5项目类似，直到开始做，才发觉是没有那么容易的，本想用keras，模拟一下P5项目中的流程，开始上手以后，就发觉困难重重，也暴露出对以前学习到的一些概念，已经这些概念在实际开发用起到那些作用，认知不足的问题。

本想自己去搭建VGG的层次结构，最终还是放弃了。自己一层一层的去搭建，在自身基础薄弱的时候，这样做显然是不现实的，在导师同学的提醒下，选择了迁移学习的这条路。

虽然遇到这样那样的问题，但是还是在导师和同学的指导下，查阅大量资料，将项目完成至此地步，把看似不可能完成的任务，分解成一个个小问题，逐个去理解，查阅，询问，日拱一卒，也能把项目逐渐完成。

5.2 需要作出的改进

在我个人的能力和认知只能，暂时没有更好的改进方案。InceptionV3这个模型已经够复杂，我来改动恐怕已经超出能力范围。

在我了解的范围内，Inception V4和InceptionResNet效果不错，但是在一开始做项目时候，并不知道其效果如何，而且Inception V4预训练模型Keras并没有封装。所以并没有尝试这两种。

参考文献

《Going Deeper with Convolutions》Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich *Submitted on 17 Sep 2014*

《Dropout: A Simple Way to Prevent Neural Networks from Overfitting》 Nitish Srivastava，Geoffrey Hinton，Alex Krizhevsky，Ilya Sutskever，Ruslan Salakhutdinov Published 2014-06