机器学习纳米学位

毕业项目

优达学城

2017 年 3 月 8 日

I. 问题的定义

项目概述

这个项目简单描述就是基于图片的猫狗分类，但这其中涉及的许多的领域，包括计算机视觉中的图像处理，深度学习中的卷积神经网络，以及其他与机器学习相关的邻域。

所以做这个项目可以提高我对深度学习的理解，也有利于我对计算机视觉的掌握，更不用说我现在正在学习的机器学习纳米学位[0]了。

本次的数据集来自 kaggle 上的猫狗大战项目[1]，详见引用。

问题陈述

在现实生活中，我们人类能够很清楚就能理解的东西，在计算机看来就不一样了，比如我们能很清楚的看到外面草坪上有一只猫或一只狗，但这在计算机眼里就是一连串的 1 和 0，所以我在此要解决的问题就是让计算机'看到'猫和狗，并对猫和狗进行分类，所以这个问题也被一些大神戏称为猫狗大战，

要解决这个问题呢，首先要有数据，在机器学习中，数据是必不可少的东西，有了数据以后就可以构建一个可以解决此问题的模型，最后还需要强大的运算能力对模型进行训练。在训练结束后就是对我们的模型进行评估了，评估通过的话，我们的问题就解决了。

评价指标

此次项目的评价指标为猫狗预测概率值的 log loss。

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

在这里，n 为测试集的图片数量，$\hat{y}_i$是图片预测为狗的概率值，y_i为 1 如果图片为狗，为 0 如果图片为猫，$log()$是以 e 为底的自然对数。Logloss 越小则越好。

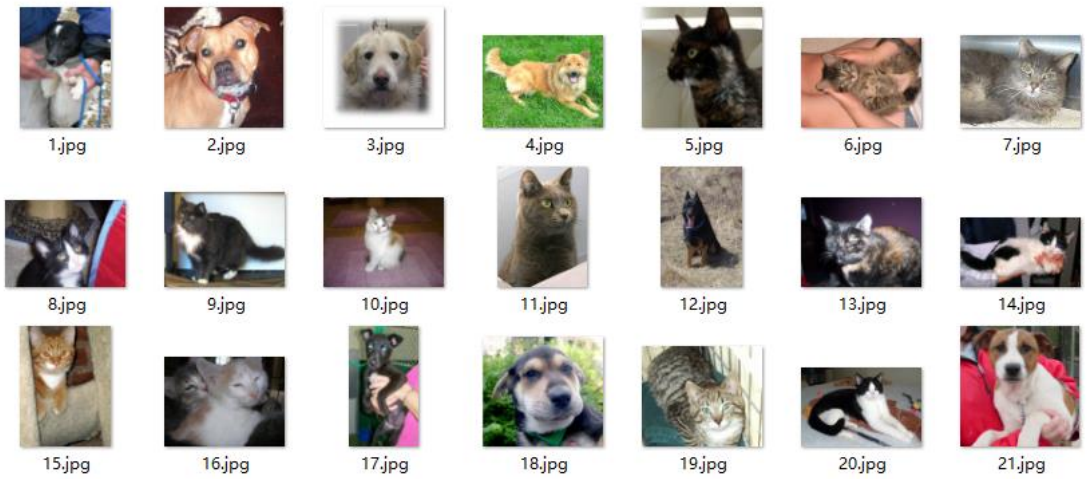
II. 分析

数据的探索

本次使用的数据在上面已经说了，是来自 kaggle 的猫狗大战数据集，它分为训练集和测试集，训练集有 25000 张图片，其中猫狗各占一半，

至于其准确度，我稍微看了一下，除了一些拍得比较差的有点难区分外，我还是没找到什么标记错误的。

而测试集有 12500 张图片，从下图就可以看出测试集的猫狗都是没有分类的，所以我也没有去数猫狗各有多少张了。



另外一些异常点，我在下面举几个例子，

 这里有 4 条狗

 这个不知道是什么的东西被标记为狗，

 这只猫在图片中比例太小，可能对分类造成困难。

这些异常点比较少，不必针对他们进行格外处理，可以通过数据提升使其对整体的影响降到最低。

探索性可视化

对于数据集，有一个缺陷就是每张图片的大小不一，这是需要处理的。

对于猫狗图片的一些探索性可视化，我实在找不到什么，我想到了把图像的 size

做一个图表，但又觉得没必要，所以此处没有可视化，不过可以参考上面数据的探索。

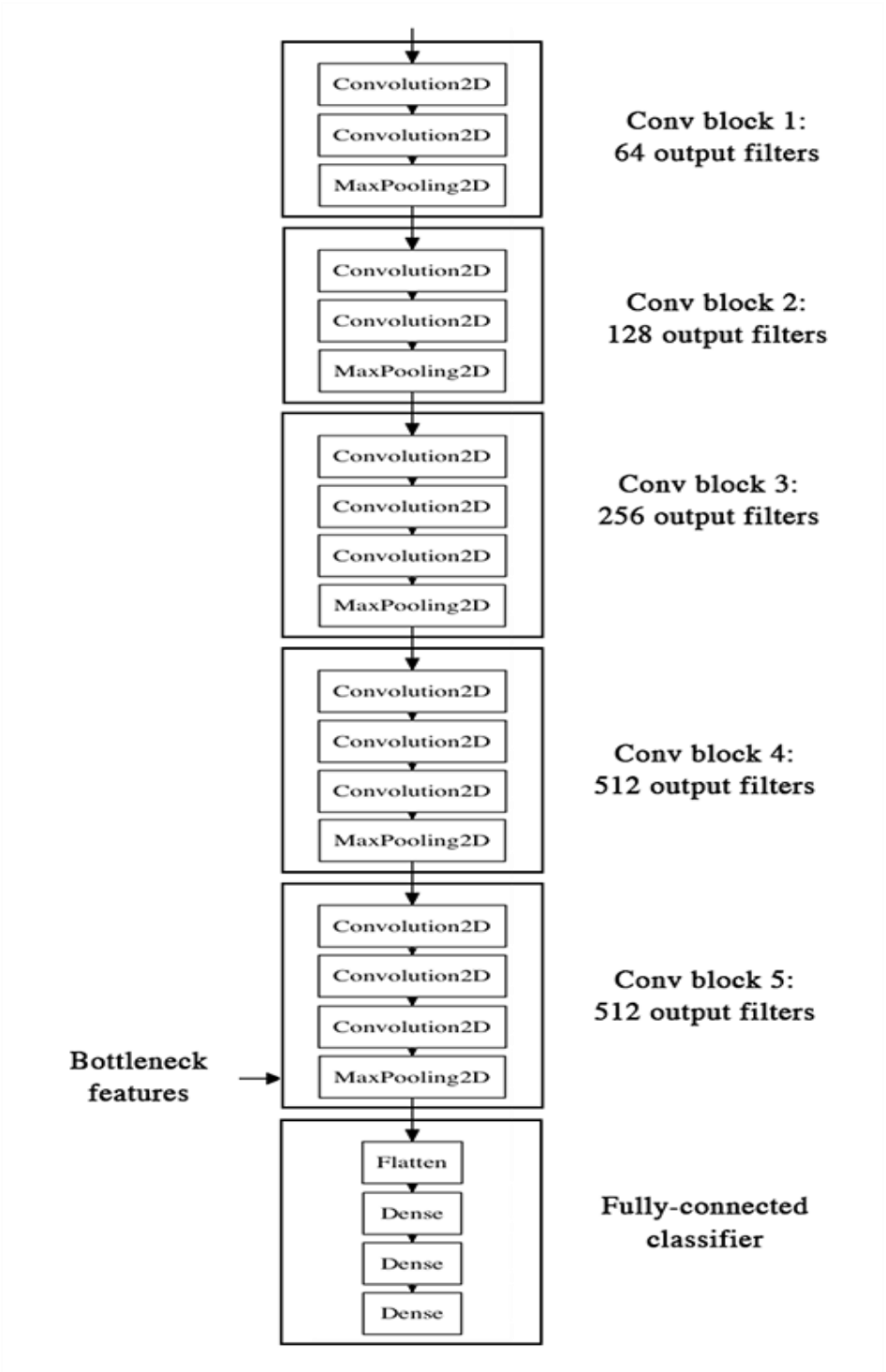
另外一件重要的是文件名，因为数据预处理的时候，需要用到，从上面的图片我们可以看到训练集的文件名为‘cat/dog’+‘.’+‘number’+‘.’+‘jpg’的形式，我们就可以据此来进行标签分类，而测试集文件名就是简单的‘number’+‘.’+‘jpg’，它的标签是需要我们进行预测的。

算法和技术

此次项目我用到的算法主要是卷积神经网络[2]，所谓卷积神经网络，它也是神经网络的一种，由一个或多个卷积层和顶端的全连接层（对应经典的神经网络）组成，同时也包括关联权重和池化层，常用于处理图像，所以这次根据图像区分猫和狗用它再适合不过了。但其本质上还是属于机器学习中的监督学习，所以我们上面需要输入猫狗的图像和区分它们的标签这两种数据。

另外我还使用了迁移学习[3]，所谓迁移学习呢，就是把已经与训练好的模型权重迁移到一个新的模型中去，因为大多数数据都存在一定相关性，所以运用已经训练好的模型的参数，这样就可以节约大量训练时间，避免从 0 开始训练。

对于此次分类，我用的 base_model 是 VGG16[4]，它是一个非常强大的卷积神经网络，详见引用，该模型的基本构架如下，

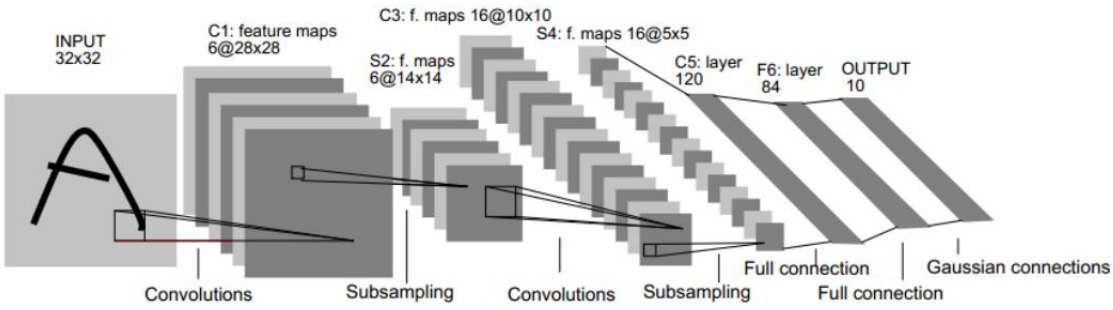


在这里我只保留它的卷积层，然后全连接是自己写的，作为我的基准模型。

如上图，VGG16 有 5 个卷积区，每个卷积区最后一个 max pool，共有卷积层 13 层， output filter 由 64 逐渐增加到 512，最后 flatten 后加上全连接。进行了这么多层的卷积，足以解决猫狗分类问题，

然后我再对一些算法细节做些说明吧，

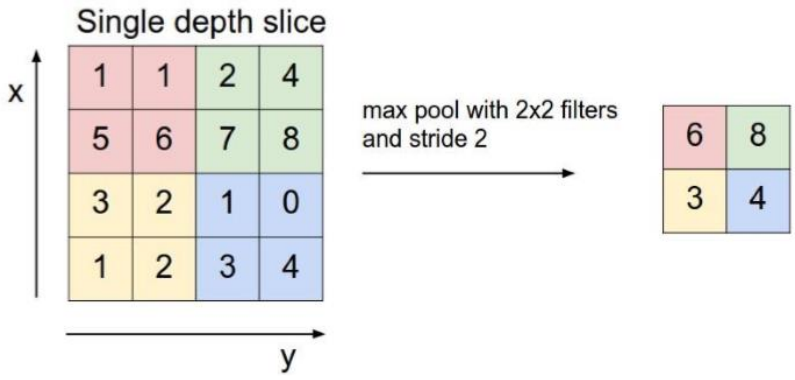
先说卷积（CNN）吧，在 CNN 中，图像的一小部分（局部感受区域）作为层级结构的最低层的输入，信息再依次传输到不同的层，每层通过一个数字滤波器去获得观测数据的最显著的特征。这个方法能够获取对平移、缩放和旋转不变的观测数据的显著特征，因为图像的局部感受区域允许神经元或者处理单元可以访问到最基础的特征，例如定向边缘或者角点。



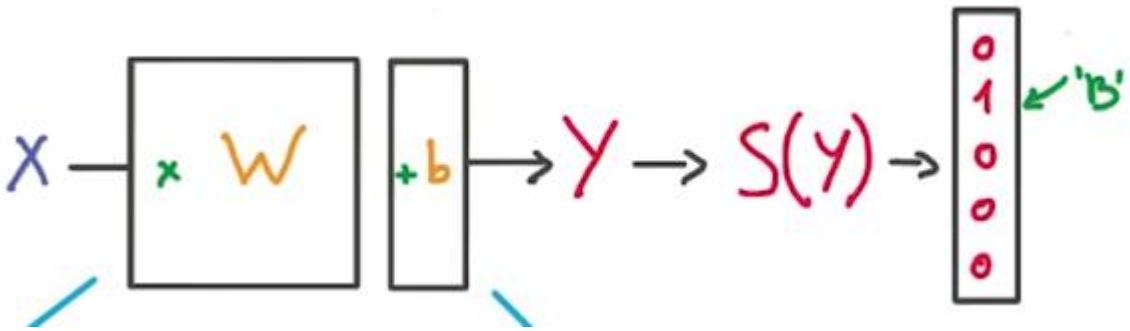
通过卷积层获得了图像的特征之后，理论上我们可以直接使用这些特征训练分类器（如 softmax），但是这样做将面临巨大的计算量的挑战，而且容易产生过拟合的现象。为了进一步降低网络训练参数及模型的过拟合程度，我们对卷积层进行池化/采样(Pooling)处理。池化/采样的方式通常有以下两种：

Max-Pooling: 选择 Pooling 窗口中的最大值作为采样值（如下图）

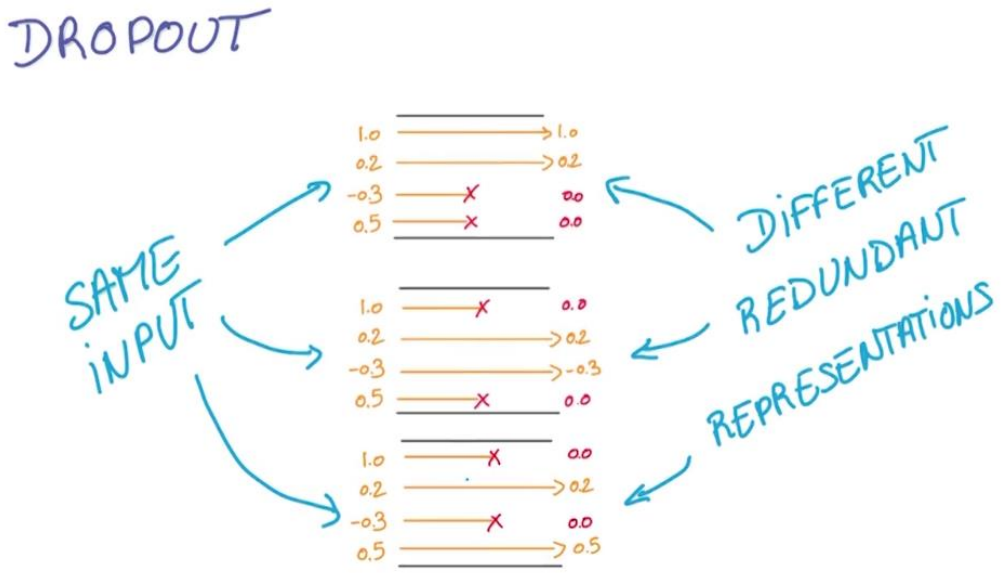
Mean-Pooling: 将 Pooling 窗口中的所有值相加取平均，以平均值作为采样值



然后我们就可以将我们得到的特征输入全连接层，全连接就是经典的神经网络中的神经元（Y=wX+b），然好还有激活函数，，下图的激活函数是 softmax.即分类器，这样我们的分类就完成了。



然后在上面过程中，有一个不得不提到的一个就是 dropout，它是为了神经网络中防止过拟合的一个重要技巧，如下图，你对输入的数据每次都随机的 drop 一些，那样神经网络就不能依赖任何输入的值，因为它们可能下一秒就被忽视掉，这样神经网络就不容易对输入的数据产生过拟合。



基准模型

这次项目我用来衡量解决方案性能基准是提交 kaggle 猫狗大战比赛后由其给出的 logloss 值。

我看了第 100 名为 logloss 值约为 0.05，但我还是不要太自大，就取 0.1 吧，0.1 已经是一个很合适的值了，我训练时看 logloss 为 0.1 就已经是 99%的准确率了，虽说 99%仅是训练时的准确度，但如果测试集（即 kaggle）的 logloss 达到 0.1，那应该也能说明准确率达到 99%,所以就这么决定了。

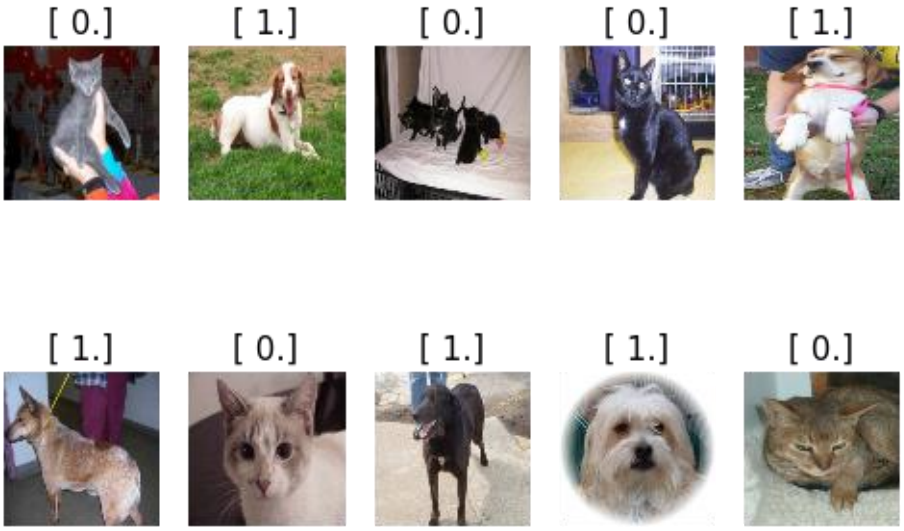
III. 方法

数据预处理

上面我已近提到了数据的一些特征，现在我就要根据这些特征来进行数据预处理，首先是图片的尺寸大小问题，我运用 openCV [5] 将所有图片进行 resize，这样它们才能输入同一个模型，我可不想每张图都弄一个模型，另外，因为我用

的 VGG16，所以我把他们都换成 224x224 的尺寸。还一件事就是 cv2.imread 将图片转为了 BGR 通道，所以我又将他们转回了 RGB 通道。

然后前面已经说过训练数据的文件名，我根据文件名将图像进行划分，每张图片都给它加上标签，猫就是 0，狗就是 1，而测试数据是没有标签的，我就暂时把他们放在一边，现在图像和标签都弄好了，还要将图片和标签进行洗牌 shuffle，这样处理好的数据才能将他们放入模型训练。基于此次处理后的数据，我将它们放到模型中训练，



上面是训练集，这是随机抽样的图片，有猫有狗有标签，通过我自己的观察，嗯，数据没错。

至于训练数据，我除了标签部分，也对其进行了如上的处理，

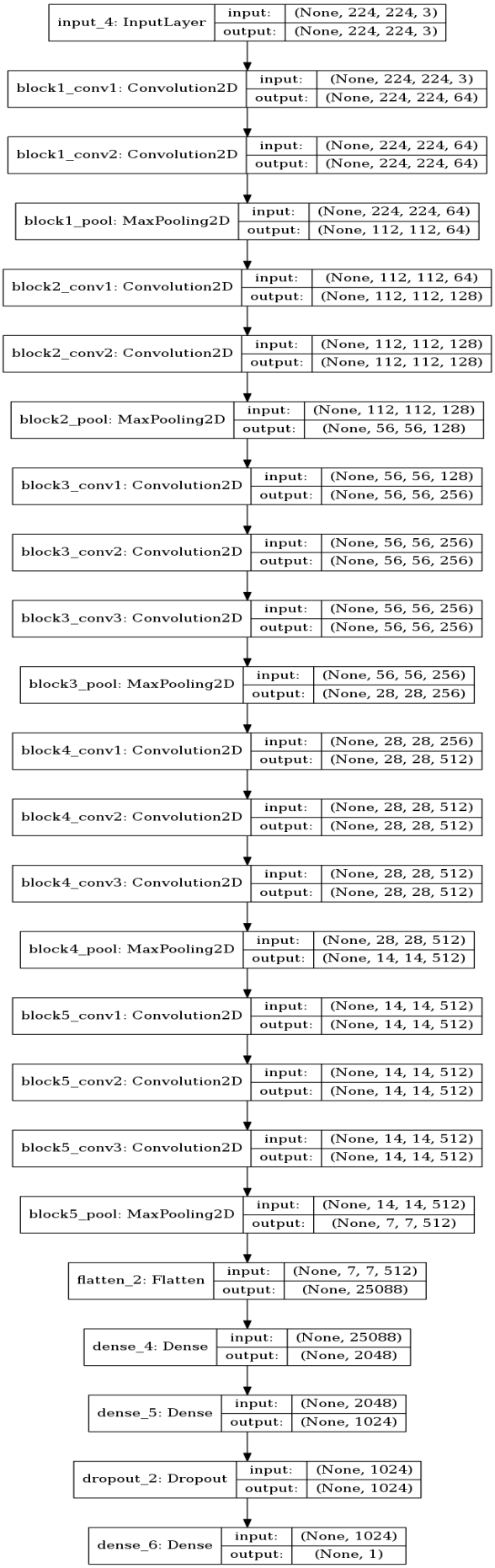
（其实我之前给训练数据都加上了标签 2，代表无标签，之后觉得没必要删了）

另外，我还可以将数据进行提升，详见下面的完善。

执行过程

现在，终于可以把数据放进模型进行训练了，本次项目使用的是 keras[6]

我先简单描述一下我的模型吧，我的模型是基于 VGG16 的模型，VGG16 输入的图像尺寸为 224x224，然后训练集有 25000 张图像，所以我的 input_shape 为(None,224,224,3)，经过第一层卷积后 shape 变为(None,224,224,64)，再经过第 4 层的 maxpool 后 shape 变成(None,112,112,64)，再经过第 5 层的卷积 shape 变成(112,112,128)，再经过第 7 层的 maxpool 后 shape 变成(None,56,56,128)，再经过第 8 层的卷积 shape 变成(56,56,256)，再经过第 11 层的 maxpool 后 shape 变成(None,28,28,256)，再经过第 12 层的卷积 shape 变成(28,28,512)，再经过第 15 层的 maxpool 后 shape 变成(None,14,14,512)，再经过 3 层的卷积，再经过第 19 层的 maxpool 后 shape 变成（None,7,7,512），经过 flatten 参数数量达到 25088，再经过我的 3 层分别深度为 2048，1024，1 的全连接层，输出我们的最终结果，结果是 0 到 1 间的一个数，越靠近 1 模表示模型越觉得图片是狗，越接近 0 则表示越像猫。


input_4: InputLayer | input: (None, 224, 224, 3) | output: (None, 224, 224, 3)
block1_conv1: Convolution2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 64)
block1_conv2: Convolution2D | input: (None, 224, 224, 64) | output: (None, 224, 224, 64)
block1_pool: MaxPooling2D | input: (None, 224, 224, 64) | output: (None, 112, 112, 64)
block2_conv1: Convolution2D | input: (None, 112, 112, 64) | output: (None, 112, 112, 128)
block2_conv2: Convolution2D | input: (None, 112, 112, 128) | output: (None, 112, 112, 128)
block2_pool: MaxPooling2D | input: (None, 112, 112, 128) | output: (None, 56, 56, 128)
block3_conv1: Convolution2D | input: (None, 56, 56, 128) | output: (None, 56, 56, 256)
block3_conv2: Convolution2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
block3_conv3: Convolution2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
block3_pool: MaxPooling2D | input: (None, 56, 56, 256) | output: (None, 28, 28, 256)
block4_conv1: Convolution2D | input: (None, 28, 28, 256) | output: (None, 28, 28, 512)
block4_conv2: Convolution2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
block4_conv3: Convolution2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
block4_pool: MaxPooling2D | input: (None, 28, 28, 512) | output: (None, 14, 14, 512)
block5_conv1: Convolution2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_conv2: Convolution2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_conv3: Convolution2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_pool: MaxPooling2D | input: (None, 14, 14, 512) | output: (None, 7, 7, 512)
flatten_2: Flatten | input: (None, 7, 7, 512) | output: (None, 25088)
dense_4: Dense | input: (None, 25088) | output: (None, 2048)
dense_5: Dense | input: (None, 2048) | output: (None, 1024)
dropout_2: Dropout | input: (None, 1024) | output: (None, 1024)
dense_6: Dense | input: (None, 1024) | output: (None, 1)

另外，我最先使用的优化器是 adatelta，后来试了 RMS，又调整成 sgd，Learning rate 为 0.0001，还设置了 0.9 的 momentum，用来加快收敛。我的 loss 用的 binary_crossentropy，这个其实就是 logloss，也就是 kaggle 评估提交者的指标，所以我选的它。

由于 VGG16 是与训练好的模型，所以我可以直接利用，中间我就有一个失误，就是忘了冻结参数，导致要上千秒才能训练一代，改正后训练时间大大缩短，只要 400 多秒（在 p2.xlarge 上训练，25000 张图）。

训练过程中又出现了 loss，accuracy 都不变的情况，输出的预测值也只有一个固定的值，我试过减少卷积、换优化器、改 batch_size,都没成功，在查阅了大量资料后，我终于找到了问题之所在，就是模型的学习率太高，这时我想起看 udacity 的 DL 教程时老师说的‘一旦出现问题，永远记得降低你的 learning rate‘，当时没太在意，现在终于认识到它的重要性了。

所以在经过上面这些调整后，模型终于训练出啦，让后保存它，这是我的一个教训，永远记得保存你的模型。对了，数据也要记得保存，这样下次再做的时候可以直接拿出来用了。

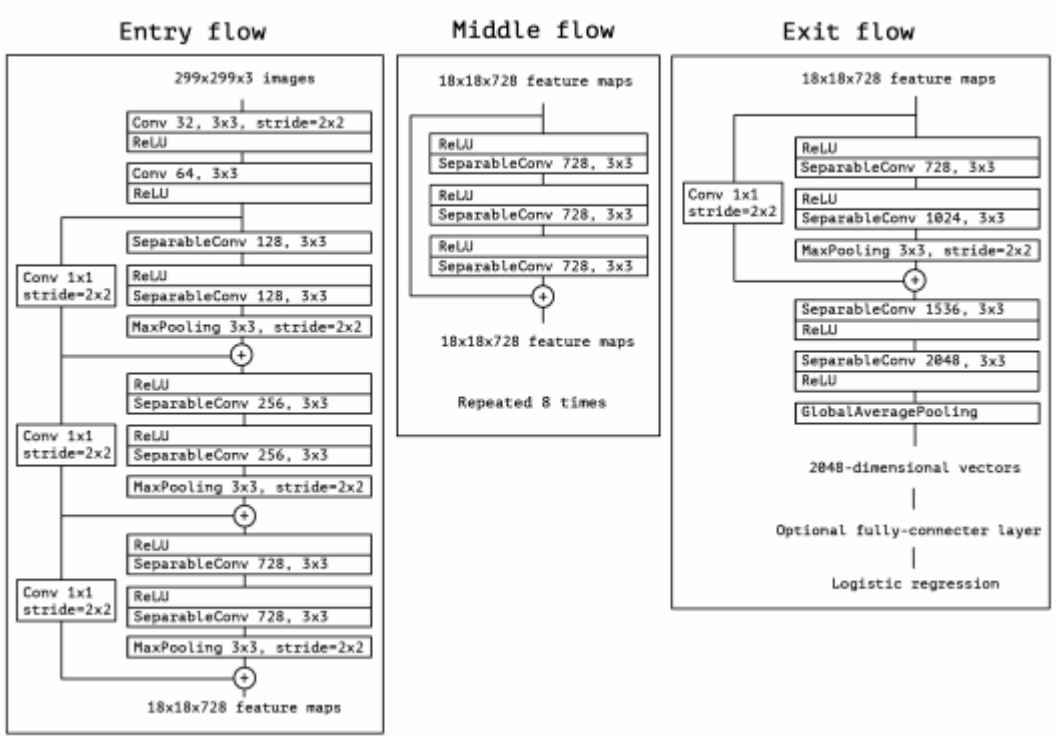
都弄好后就可以拿去评估了，我先对模型的预测进行了可视化，看起来没问题后，就生成了 csv 文件并提交到 kaggle 上评估。

完善

这是我运用上面模型得到的初步结果，logloss 为 0.17295，对于这个结果我并不太满意，所以还需要继续完善。

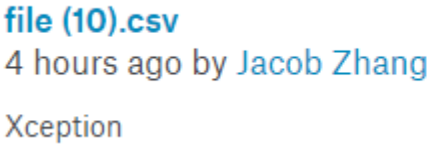
依据导师的建议我使用了 Xception [7]，在 ImageNet 上,该模型取得了验证集 top1 0.790 和 top5 0.945 的正确率,

Xpception 是一个非常大的网络，详见下图。



由图就可以看出它的内部有多复杂，我们只需知道它进行了多次卷积，池化，还有许多循环和其他一些什么，但其实它的每一个小的单元都是简单的，无非是卷积、池化，和其他一些卷积神经网络中常见的东西，这些可以见上面的说明。

根据 Xception 的得到了下面的结果



终于达到了当初定下的基准了。

上面是没有进行数据提升的结果，然后我进行了数据提升，

首先我将训练集分为 train_data 和 valid_data，将 train_data 进行数据提升，valid_data 不动。

首先更改图片亮度，首先将图片转化成 HSV 通道，然后在第三个维度随机乘上一个 0.5 到 1 间的数，最后在将图片返回 RGB 通道,然后我还将图片进行了上下翻转和左右翻转，效果如下。




上面只是举的一个例子，对于所有训练数据，我随机将 25%的数据进行了亮度调整，25%进行上下翻转，25%进行左右翻转。

对 valid 数据就不进行处理。当然，测试集是不会去动它的

进行了数据提升后，我们用上面的 Xception 模型对新的数据进行拟合，得到结果。

我说一下 jupyter notebook 中训练记录丢失的问题，那是由于网页接口端口造成的，不过模型其实我都已经训练完成并保存了，也保存了 tensorboard 的 log,

详见下面的分析。

IV. 结果

模型的评价与验证

先来说一下最终模型，我的最终模型以 Xception 为基本模型，包含了 除去其顶层的全连接的所有层，再加上 flatten，2 层全连接（包括输出层），其中包含激活函数，dropout.

再来我们的评价指标是 logloss，所以我们使用 binary_crossentropy（即对数损失 logloss）为目标函数。优化器为原来的 SGD, Learning rate 为 0.0001，还设置了 0.9 的 momentum.

最终模型达到了 0.099 的 logloss，说明模型还是达到了要求。

最后模型对数据和环境的敏感性，如果你的数据较少，而 dropout 又不是很高的话，模型是有可能出现过拟合的，这时你可以，将我的数据提升改为不是直接将生成的图片转化为原图片，而是生成新的图片，这样就可以弥补数据的不足，还可以通过提高 dropout 或添加 L2 正则项来防止过拟合。如果数据量过多，模型的训练速度可能会较慢，这是可以通过减少 batch_size 和 epoch 来减少训练时间，虽然会慢，但模型往往会达到更高的准确率。

就算模型并不是用来对猫狗进行分类，而是用到其他用途，模型也能够胜任，只是要根据实际问题更改其输出层。

合理性分析

我们先简单预测一下测试集的数据，我随机选择了 20 张测试集的图片进行预测，结果如下，可以看到这 20 张都预测成功了。

然后我们可以提交到 kaggle 上去了，得到的结果如下，这是数据提升后 Xception 的结果。

file (11).csv
just now by Jacob Zhang

add submission details

这个结果较前面的 0.099 又有所上升，到达了 0.092，终于，我的 logloss 达到了 0.1 以下.。

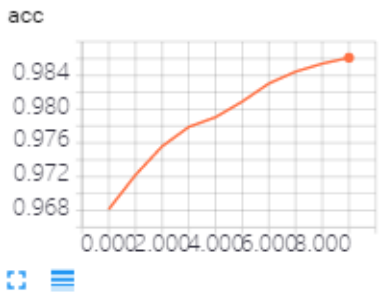
V. 项目结论

结果可视化

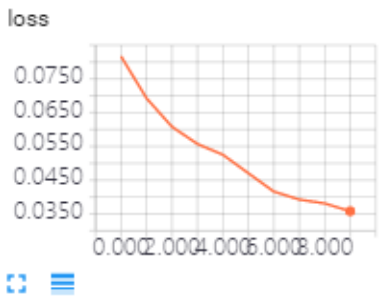
在这里我就稍微对训练过程的 loss 和 accuracy 值进行下分析吧，由下图可见模型的 train loss, valid loss 都是在不断降低，train accuracy, valid accuracy 都是不断上升，只是 valid accuracy 上升过程有点曲折，波动有点大，我猜可能施训练过程中过拟合了，不过模型又自己慢慢调整过来了，最终不管是训练数据还是验证数据都达到了一个好的结果。

VGG16 Xception

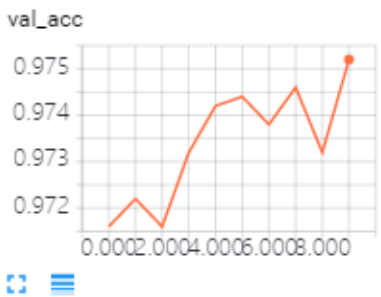
acc



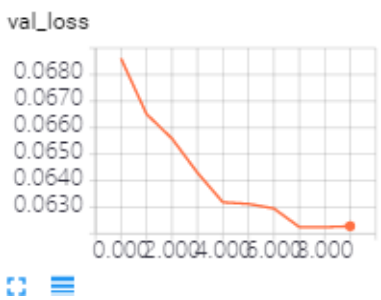
loss



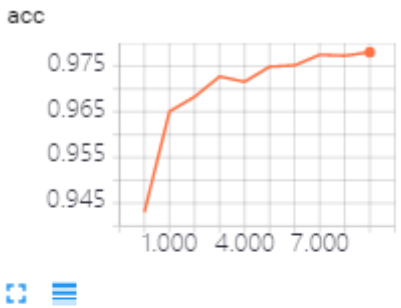
val_acc



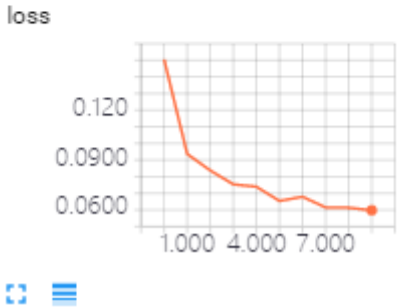
val_loss



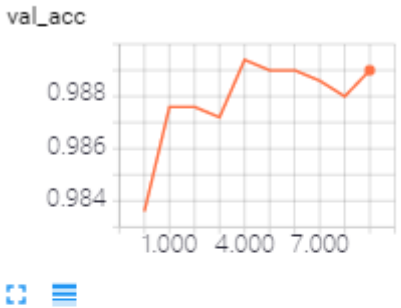
acc



loss



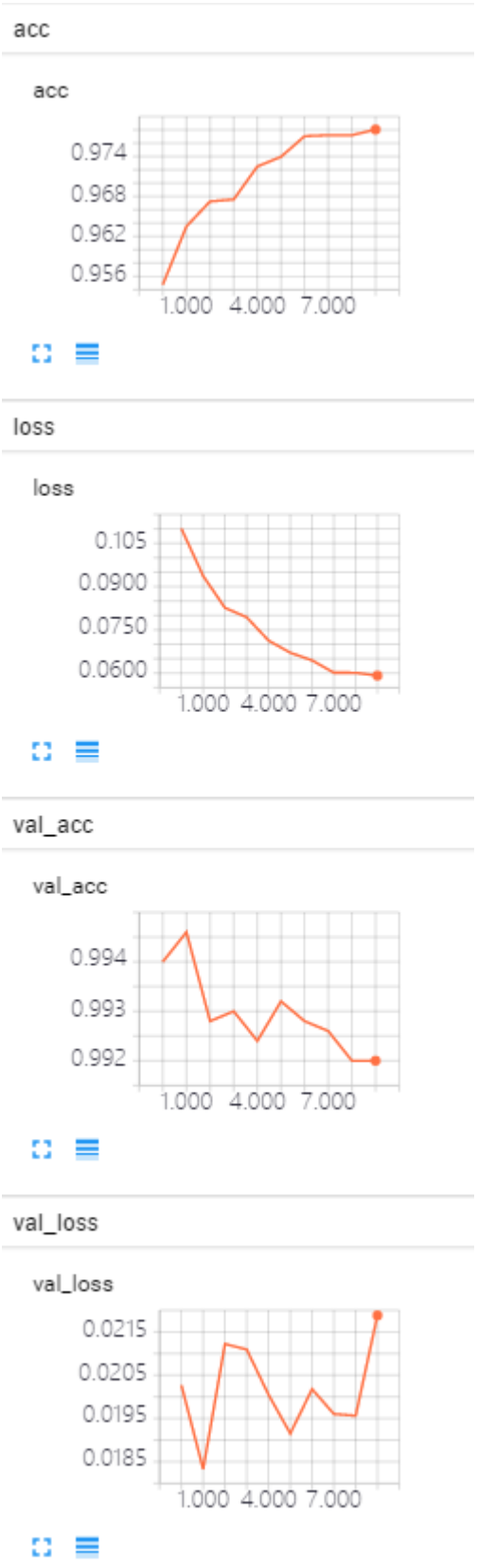
val_acc



val_loss

（截图有点失真）

数据提升后的 Xception



在下面我们来细看一下，上面两者的 loss 吧，VGG16 的 loss 明显较小，但

是在提交 kaggle 测试 logloss 却更高，可见它在训练中过拟合了，而 Xception 却达到了一个较好的结果。

在最后一张图，我们的 valid_data 的 accuracy 竟然在下降而 loss 竟然在上升，可是它的结果却是三个中最好的，正是奇怪，我猜可能模型 3 训练时的 valid_data 在第二个模型训练时已经有一部分拟合过了，所以在模型 3 对新的数据（即经过数据提升的数据）进行训练，训练集的 loss 下降，accuracy 上升，而在 valid_data 达到了过拟合，然后在测试集（kaggle）又是新的数据，所以它的 loss 才会最低。

对项目的思考

到此，这个项目就告结束了。在项目中，最有趣的就是在它训练的时候盯着 loss 看了，看着 loss 不断下降，accuracy 不断上升真的好有趣。

至于项目困难的地方就是，完善模型施对其进行微调。俗话说达到 90%的准确率很容易，达到 99%的准确率很难，到达 99.99%的可能性更是难上加难，所以在我试图将准确性提高到一个更高的程度时就深感其困难了，不过我最终还是达到了一个较为令人满意的结果。

模型已经达到了期望，而且对模型进行一些简单调整就能在大多的通用场景解决此类型的问题。

需要作出的改进

当然，模型还是可以进行完善的，可以用其他的预训练模型或可以自己设计一个更好的。

也可以添加 L1、L2 正则项或约束项，然后数据提升方面也可以做得更多，如缩放、，暂时想到这些。

如若以我的最终模型作为基准模型，我会尝试自己设计一个更好的模型，设计更好的解决方案，这样我相信肯定能达到更好的结果。

资料及来源

【0】 udacity

【1】 kaggle

【2】 CNN

【3】 Transfer learning

【4】 VGG16

【5】 openCV

【6】 keras

【7】 Xception