

Dog Vs Cat

A Convolutional Neural Network Based Dog/Cat Classifier



([PDF版 Project Summary](#))

项目概览

使用深度学习方法识别一张图片是猫还是狗。

- 输入：一张彩色图片
- 输出：是 🐶 还是 🐱
- 数据集：来自Kaggle [Dogs vs. Cats Redux: Kernels Edition](#) 的已标记的 🐱 / 🐶 图片

问题说明

以彩色图片作为输入，训练适用于区分猫狗图像的分类模型。已有大量研究证明卷积神经网络([Convolutional Neural Network](#))是解决这类问题的非常有效的方法[1, 2, 3]。因此，在本项目中也将采用类似的模型搭建卷积神经网络并进行训练和测试。

模型搭建将采用两种方式：

1. Homebrew Model: 从头搭建一个卷积网络
2. Transfer Learning: 充分利用现存的已经经过实战检验的预训练过的模型，在此基础之上添加相应的结构

获得希望的输出

模型评价指标

本项目中训练获得的模型将以[log loss](#) 作为评价指标。其定义为：

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

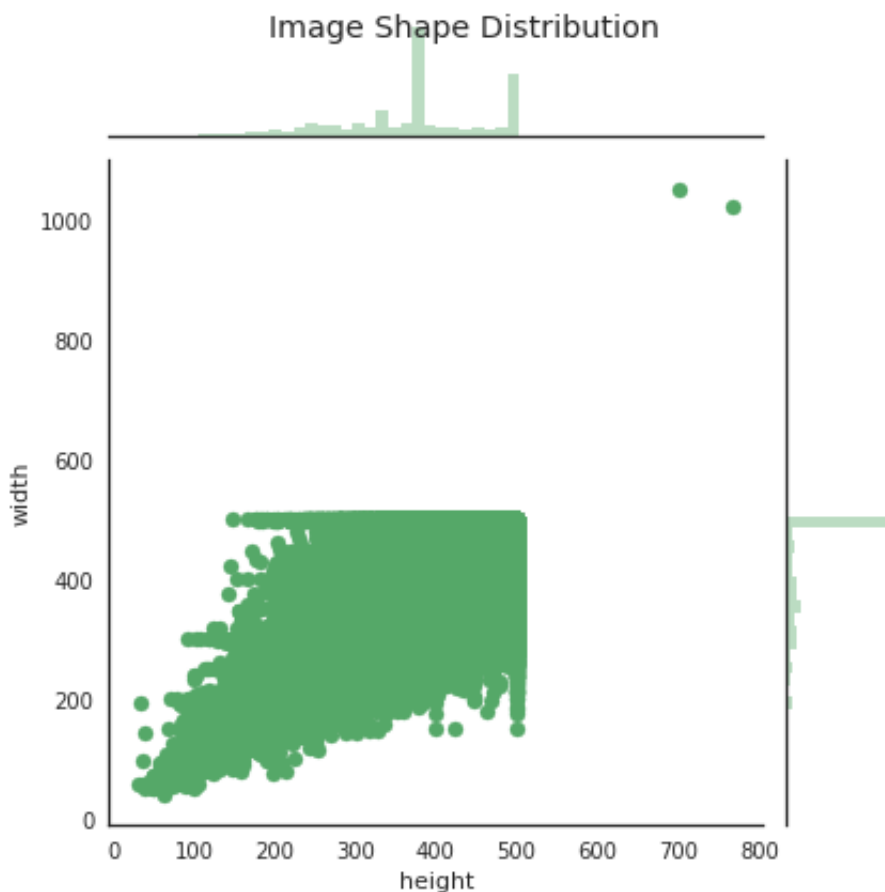
其中 n 为测试集中图片的数量， y_i 为预测图中动物为狗的概率。如果实际图中的动物是狗，则 $y_i=1$ ，否则 $y_i=0$ 。

测试集的来源主要有两个部分：一部分来自从原始数据集保留的部分没有用于训练的图片，按照业界通行的标准计算正确率。另一部分来自从互联网中获取的部分图片，用于对给予感性的理解。同时训练获得的模型将用于预测Kaggle提供的标准测试集，并将预测结果提交Kaggle系统进行评价。

数据研究

在建立模型前，首先对训练数据进行分析以获取数据的特性。

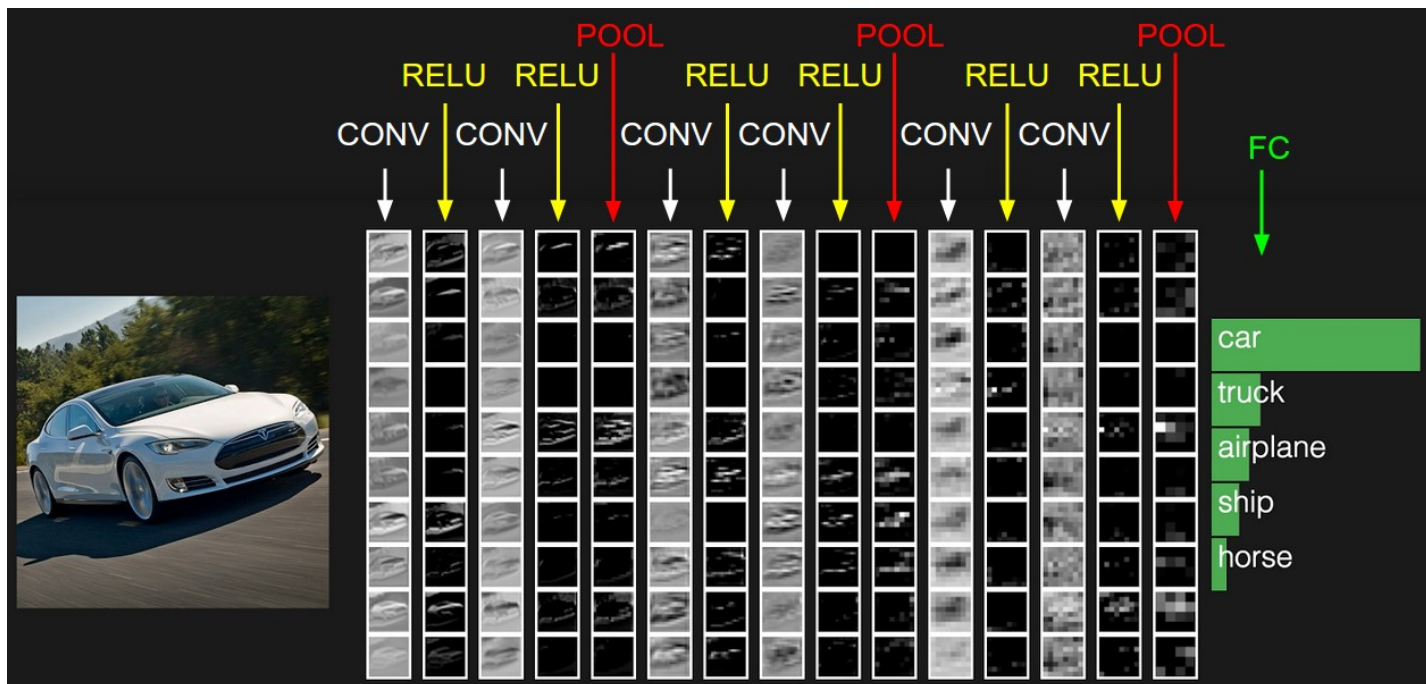
- 原始数据集中共有25000张已标记的图片，其中猫/狗图片各12500张，数目相当，因此不存在训练样本数量不均衡所导致的模型偏差。
- 原始数据集中图片的尺寸分布如图所示。可以看出图片的尺寸并不一致，多数图片的长宽都在600像素以下。本项目中的模型要求输入的图片具有同样的几何尺寸，因此需要对原始训练集的图片尺寸进行归一化处理使得图片具有统一的长宽数值，代码实现可以参考[Jupyter Notebook](#)中的『Load, Analysis, Visualize and Encode Data』代码块。这里我们将会在生成训练和测试数据时将图片尺寸放缩为224x224 pixel.



- 人工查看了部分图片，发现训练集已经包括了多种不同背景，光照条件，动物姿态，颜色等的图片。作为一个二分类问题，我认为现有的数据集已足够模型使用，不需要进行进一步的data augmentation。另外也查看了尺寸远大于其他图片的两幅异常图片，发现除了尺寸之外与其他训练图片相比并无其他异常，因此决定保留这两幅图片在数据集中。
- 原始图片是通过文件名中的"dog"或者"cat"来标记🐶/🐱的，为方便之后模型的训练，这里采用了one-hot encoding的方法来将标记转换为2维的向量。

模型简介

本项目中将使用卷积神经网络(Convolutional Neural Network, CNN)建立模型区分图片中猫狗。一个典型的卷积神经网络结构如下图所示：

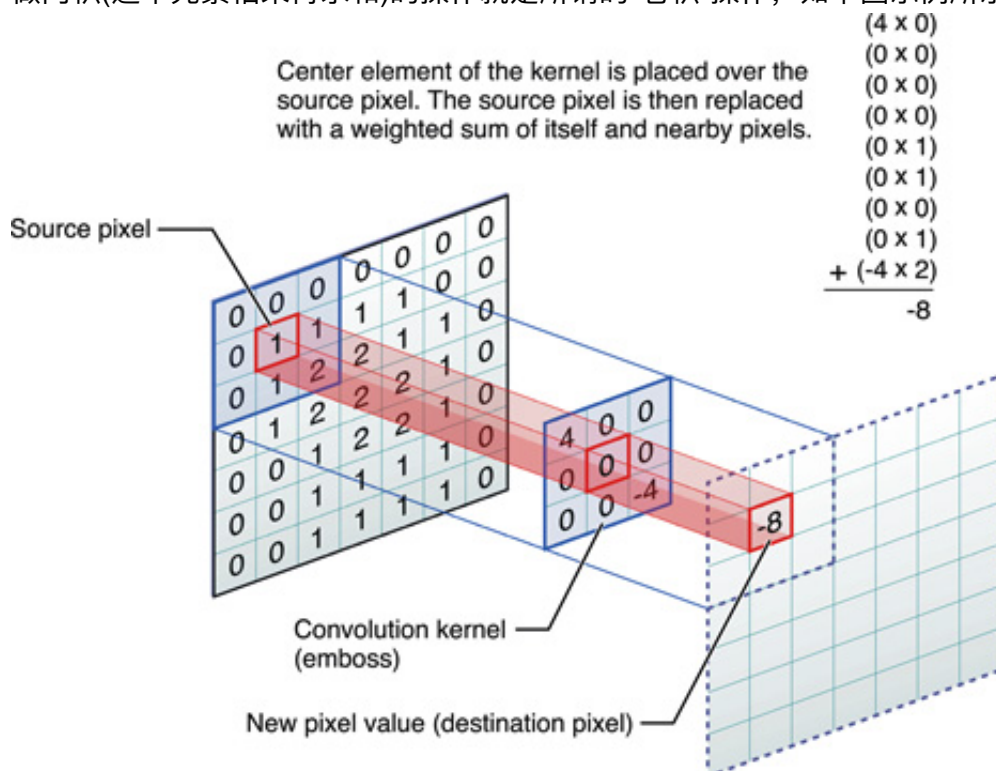


[image source](#)

其中左边的图片是输入，中间CONV, RELU, POOL, FC等代表具有不同功能的layer。模型的输出是关于图片中物体类型的判断(car, truck, airplane....)。 (reference:[1](#),[2](#)):

本项目中的模型将使用基于[TensorFlow](#) backend的深度学习库[Keras](#)来建立。本项目中所采用的模型的整体结构与上图类似, 以下将简略介绍本项目模型的构成要素。

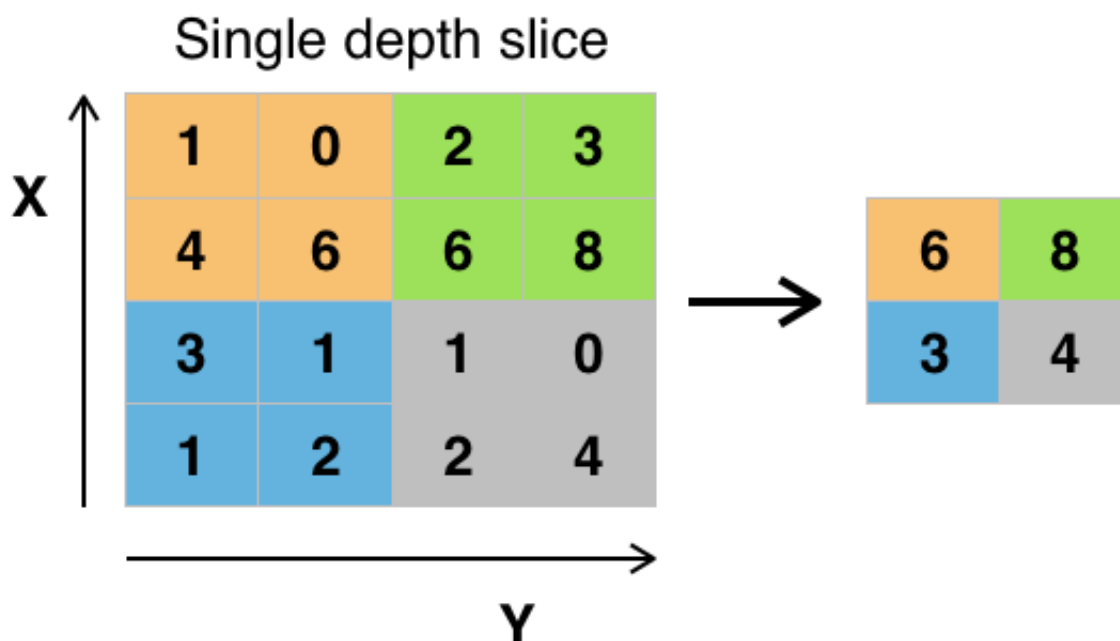
- 卷积：在卷积神经网络这一框架下, 对图像(不同的数据窗口数据)和卷积核(滤波矩阵，一组固定的权重)做内积(逐个元素相乘再求和)的操作就是所谓的"卷积"操作，如下图示例所示。



- 卷积层(Convolutional layer, CONV): 卷积神经网络中每层卷积层由若干卷积单元组成。卷积运算的目的

的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网络能从低级特征中迭代提取更复杂的特征。

- 池化(pooling): 形式上，在获取到我们前面讨论过的卷积特征后，我们要确定池化区域的大小(假定为 $m * n$)，来池化我们的卷积特征。那么，我们把卷积特征划分到数个大小为 $m * n$ 的不相交区域上，然后用这些区域的平均(或最大)特征来获取池化后的卷积特征。这些池化后的特征便可以用来做分类。下图为一个 $2*2$ max pooling的例子。



- 池化层(Pooling Layer, POOL): 池化 (Pooling) 是卷积神经网络中另一个重要的概念，它实际上一种形式的向下采样。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。
- 全连接层(full connection layer, FC):全连接层中的每一个neuron都会与全连接层之前的layer输出连接，在卷积神经网络中起到“分类器”的作用。
- Dropout layer: 在CNN的实际应用中，常常会添加Dropout layer用以防止过拟合。Dropout layer作用方式为在训练过程中每次更新参数时随机断开一定百分比的输入神经元连接。
- 激活函数(Activation): 在神经网络里，一个节点的激活函数定义了给予节点给定的输入或输入的集合的输出。激活函数可以给模型加入非线性因素，以提高简单线性模型的表达能力。常用的激活函数包括 sigmoid, ReLU, softmax等。
- Softmax: [Softmax](#)是sigmoid函数在多分类问题上的推广,具体定义可以参考[这篇文档](#)。
- 线性整流层(Rectified Linear Units layer,RELU): 线性整流层 (Rectified Linear Units layer, ReLU layer) 使用线性整流 (Rectified Linear Units, ReLU) $f(x)=\max(0,x)$ 作为这一层神经的激活函数

(Activation function)。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。

- **Transfer Learning:** 本项目中也会采用迁移学习的方法。迁移学习方法是一种机器学习框架，不同于传统的监督学习、无监督学习和半监督学习，这种方法通过将某一源领域的标注数据样本和目标领域的未标注样本或少量的标注样本中学习到一个紧凑的、有效的表示，然后将学习到的特征表示方法应用到目标领域中。迁移学习没有像传统的机器学习那样要求训练数据与测试数据必需服从相同的分布，因此，迁移学习能有效地在相似领域或任务之间进行信息的共享和迁移。在本项目猫狗分类这一问题中的数据集为包含有猫狗的图片，与[ImageNet](#)数据集存在着很强的相关性，因此我们有理由相信使用基于经过ImageNet数据集训练过的模型可以帮助完成本项目的任务。

模型拟合

本项目中模型的评价采用的评价指标为log loss.可以证明在二分类的情况下，log loss与cross entropy是等价的([reference](#)).因此，keras搭建的模型的loss function将选用[categorical_crossentropy](#)。没有选择binary_crossentropy是希望之后本项目的代码可以更方便的推广到多分类问题上。

相应的，模型的最后输出层的激活函数选用[Softmax](#)。

模型采用的optimizer为[rmsprop](#) optimizer.

基准测试 Benchmark

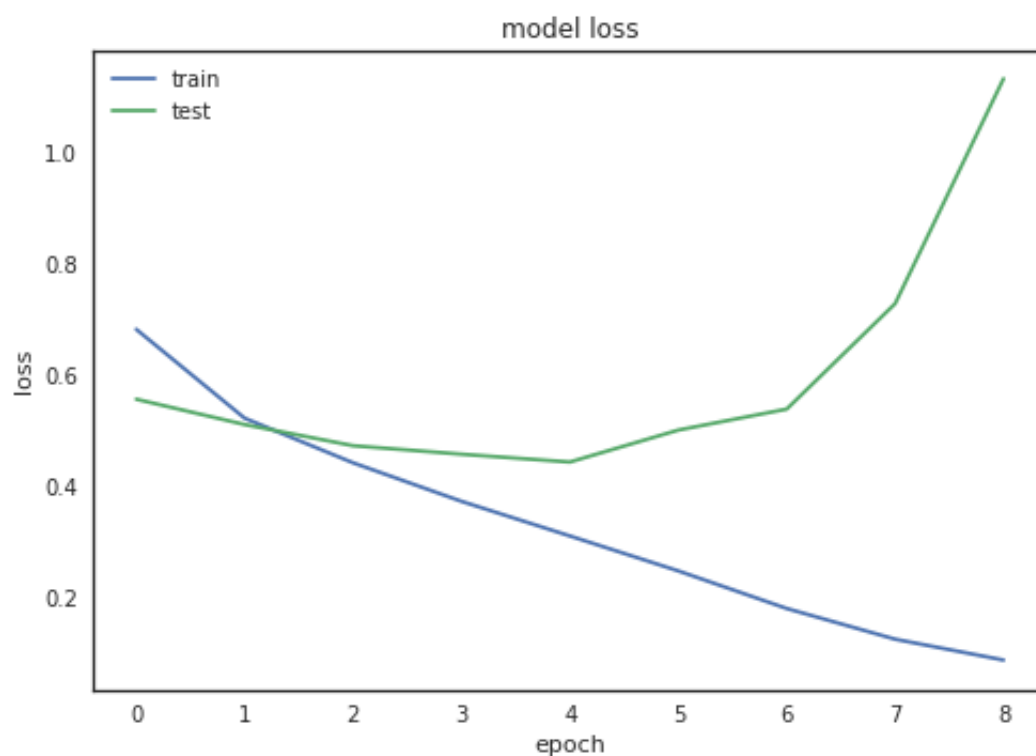
训练获得的模型将用于预测Kaggle提供的标准测试集，并将测试结果提交到[Kaggle](#)进行评估，并采用Kaggle使用的log loss作为评价指标，参考kaggle该项目的leaderboard定义阈值为log loss小于1.0.

模型训练与评估

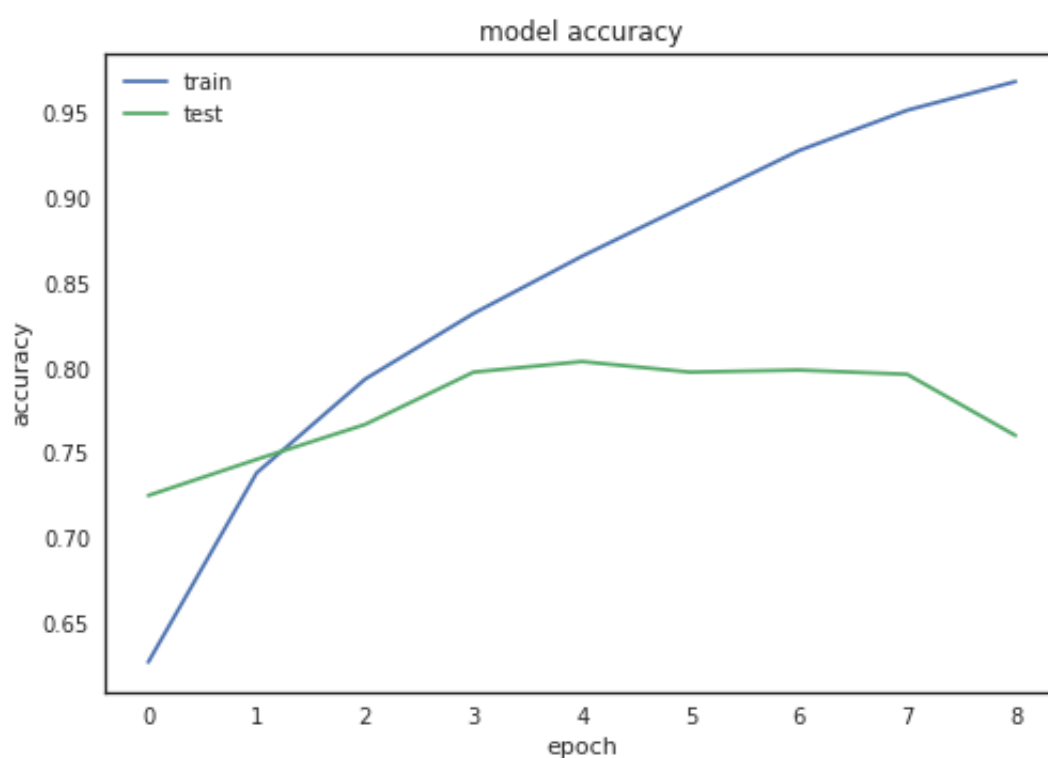
因所需的计算量较大, 本项目的模型训练使用了AWS p2.xlarge instance, 并参考了[这篇文档](#)进行了配置。关于数据预处理,可视化 and 模型搭建的代码可以参阅这个[Jupyter Notebook](#)。原始训练数据按照8:2的比例划分为训练验证集和测试集，之后训练验证集同样按照8:2的比例划分为训练集和验证集。

Homebrew Model 训练与改进

- 首先使用Keras搭建了具有三层Convolutional layer的model作为初始base model，详细的模型结构和参数可以参考[Jupyter Notebook](#)中的『Homebrew Model: 1st run』代码块。在该模型中我们设置dropout rate = 0，即没有dropout。learning rate = 0.001。模型训练设置了50个epoch并以validation loss为指标设置了early stop，以节省计算时间并在一定程度上防止overfitting。模型在8个epoch后训练结束，loss和accuracy曲线分别如下所示：
 - loss (final 0.96)



- accuracy (final 0.76)



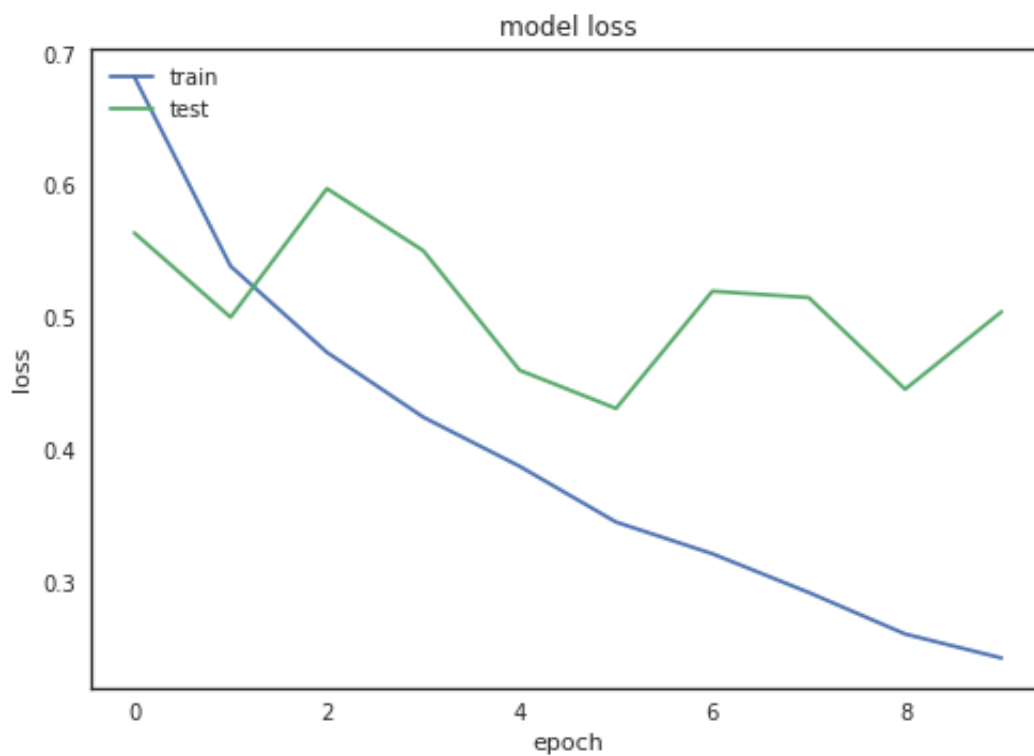
从

曲线中可以看出model在第6个epoch后出现了比较明显的过拟合的趋势。

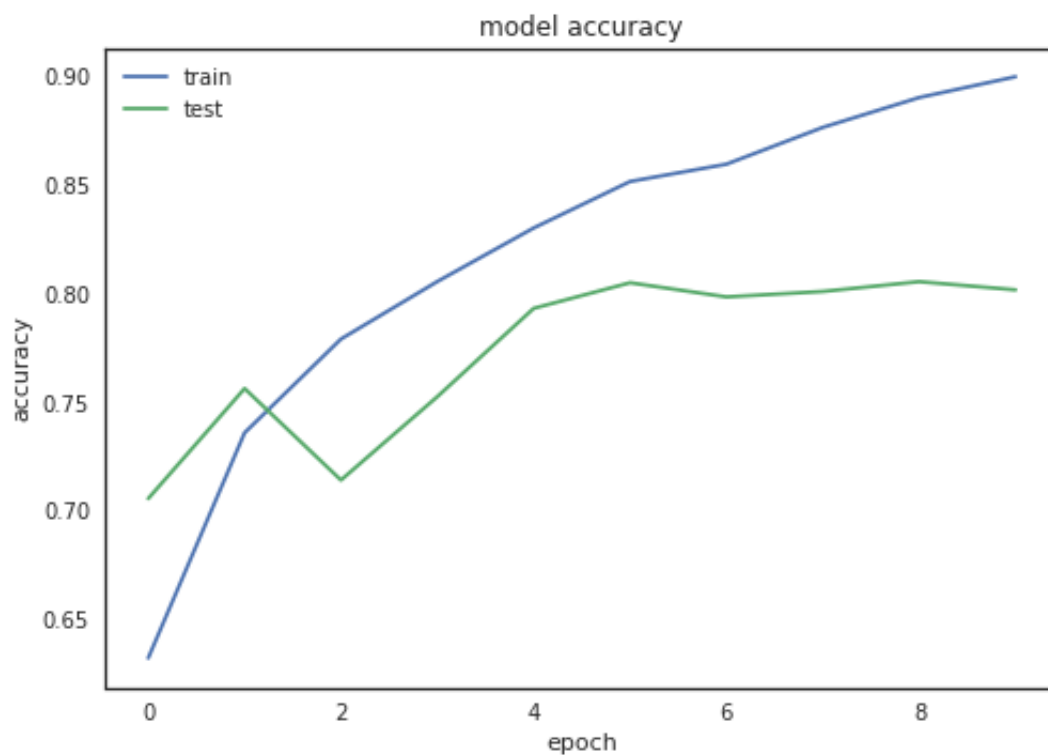
- 基于上面的base model中出现了overfitting的趋势，我们尝试设置dropout rate = 0.5来对模型进行改进。具体代码可以参考[Jupyter Notebook](#)中的『Homebrew Model: Tuning 1』代码块。以下是改进过的

model的loss与accuracy曲线：

- loss (final 0.47)



- accuracy (final 0.80)

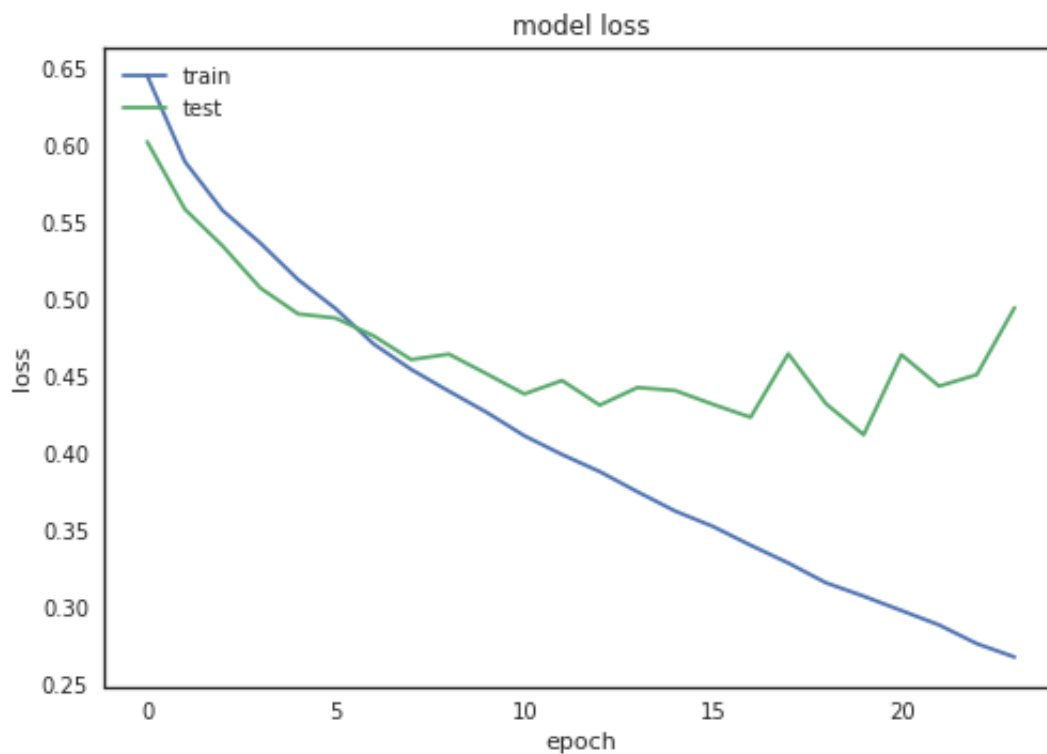


曲线中可以看出overfitting的情况有所改善，同时loss和accuracy也有提升。

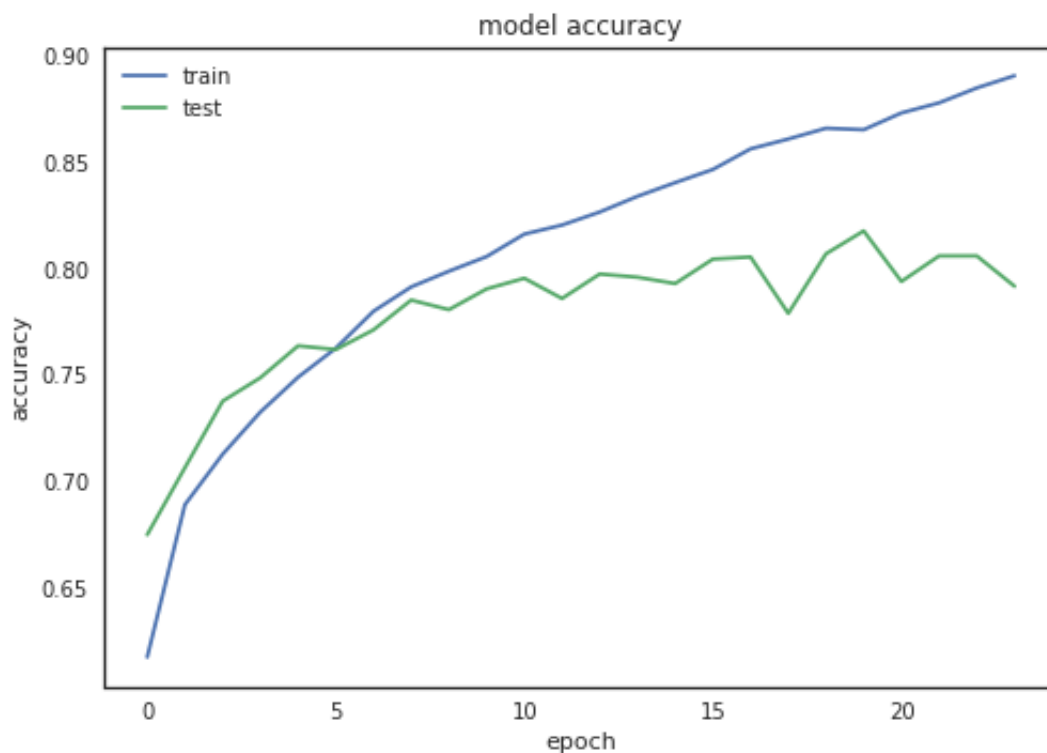
从

- 接下来基于上面的tunning 1的model我们尝试通过调整learning rate的方法改善model的性能。设置 learning rate = 0.0001 (前面model的learning rate = 0.001),其他设置不变。具体代码可以参考[Jupyter Notebook](#)中的『Homebrew Model: Tuning 2』代码块。由于learning rate的减小，model使用了更多 epoch收敛。以下是改进过的model的loss与accuray曲线：

- loss (final 0.47)



- accuracy (final 0.80)

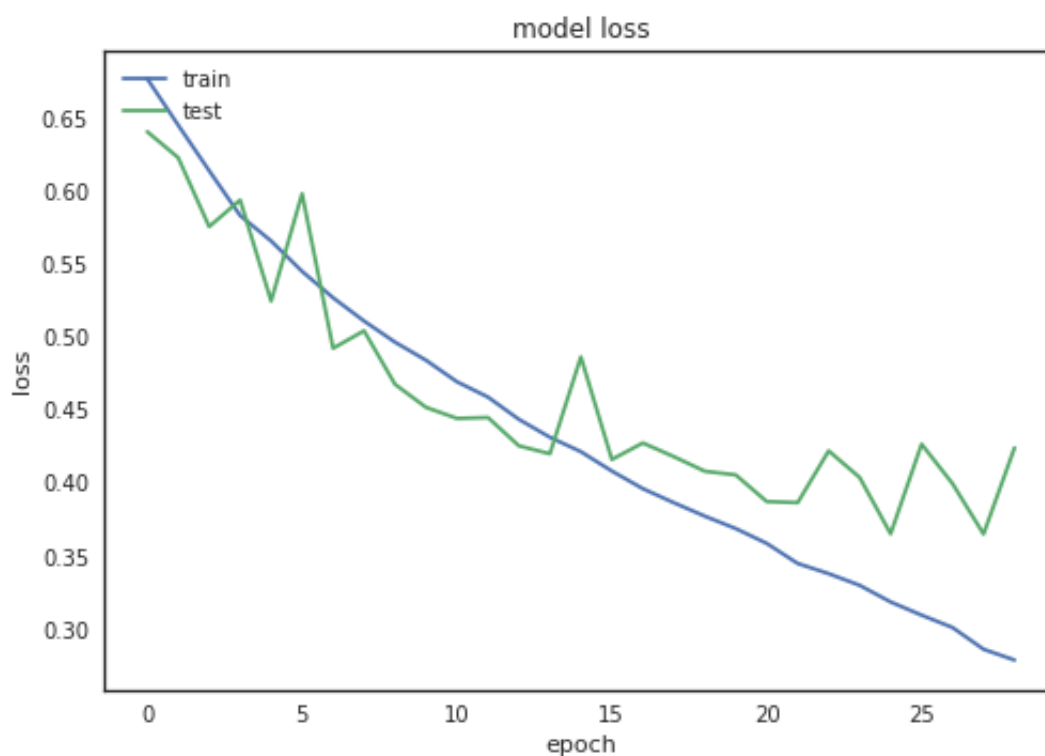


从

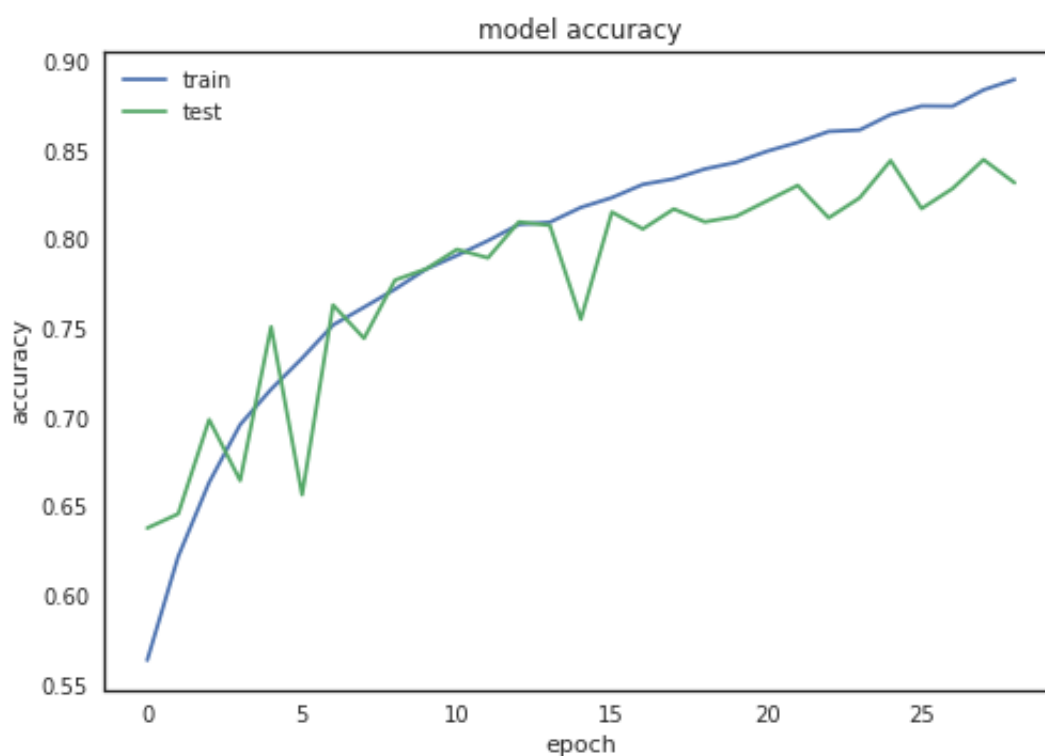
结果可以看出，虽然经历了更多的epoch，model的loss和accuracy几乎没有提升。

- 考虑到上面的tunning结果，决定从model本身的结构下手做一些尝试，在base model的基础之上多添加一层Convolutional Layer。具体实现可以参考[Jupyter Notebook](#)中的『Homebrew Model: Go Deeper』代码块。以下是这一model的loss与accuracy曲线：

- loss (final 0.41)



- accuracy (final 0.83)

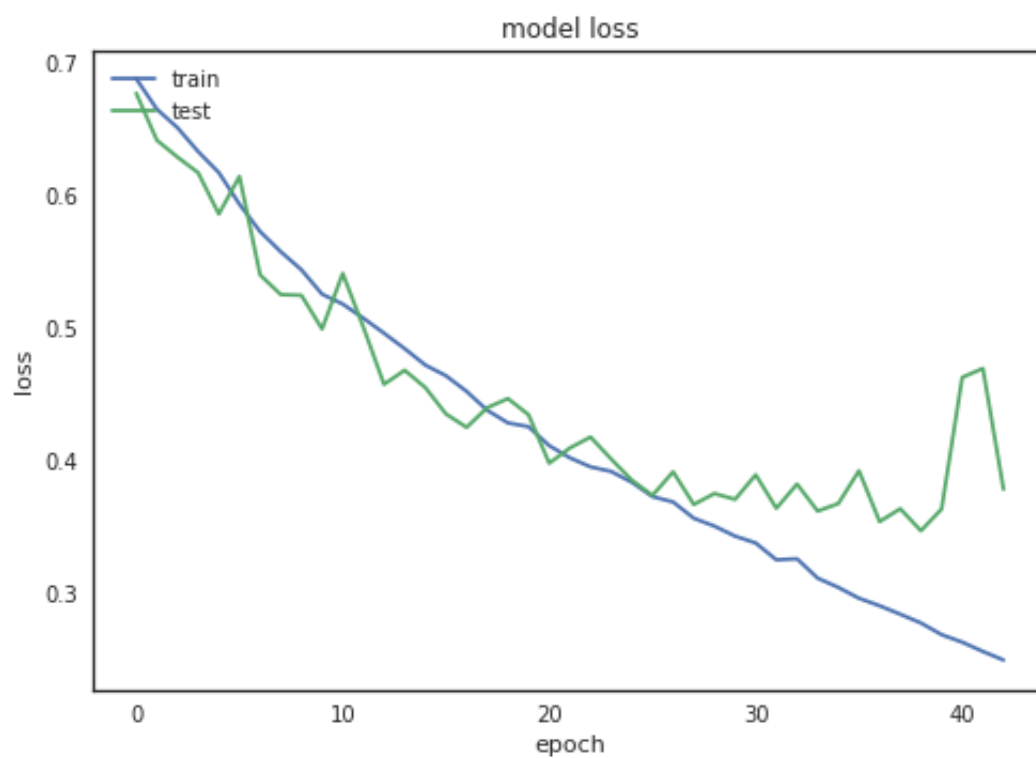


可以发现经过29个epoch的training后, model的loss和accuracy表现都有了一定程度的提升。

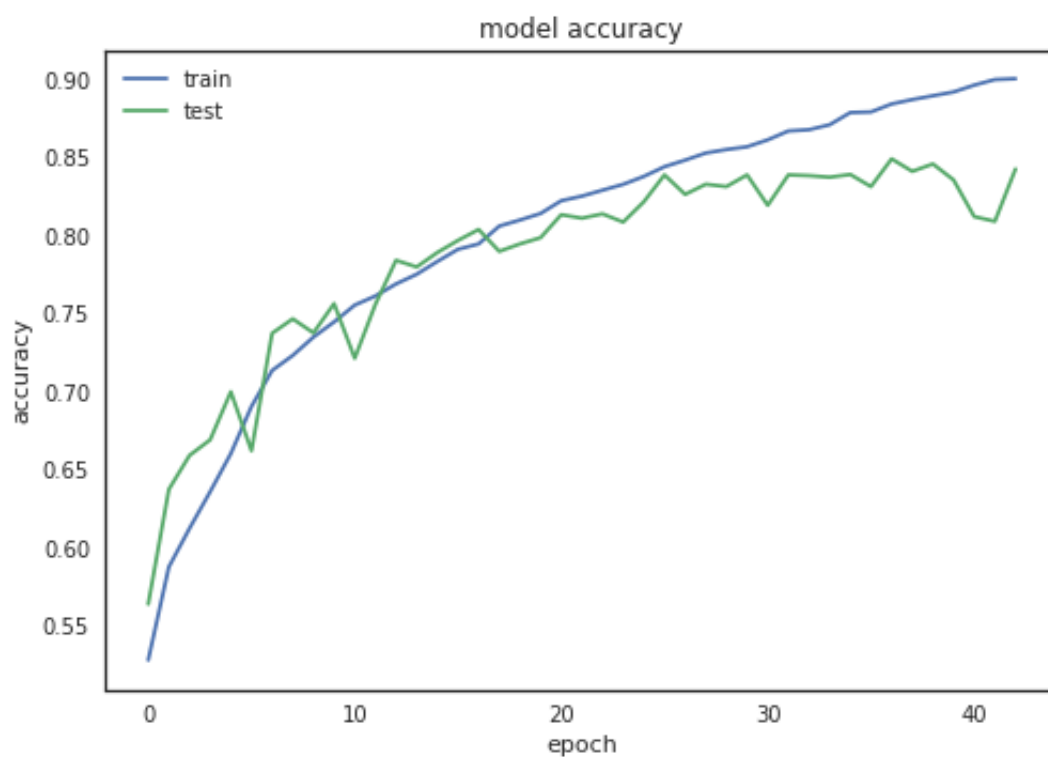
- 根据上面的training结果, 可以发现更deep的网络结构可能具有更好的表现, 因此决定go one step further, 继续增加一层Convolutional Layer, 具体实现可以参考[Jupyter Notebook](#)中的『Homebrew

Model: Go Even Deeper』代码块。以下是这一model的loss与accuracy曲线：

- loss (final 0.35)



- accuracy (final 0.85)



我们发现由于网络结构更深，训练所需要的epoch变得更多,经过43个epoch后model的log loss=0.35和

accuracy = 0.85, 都具有了更进一步的提升。

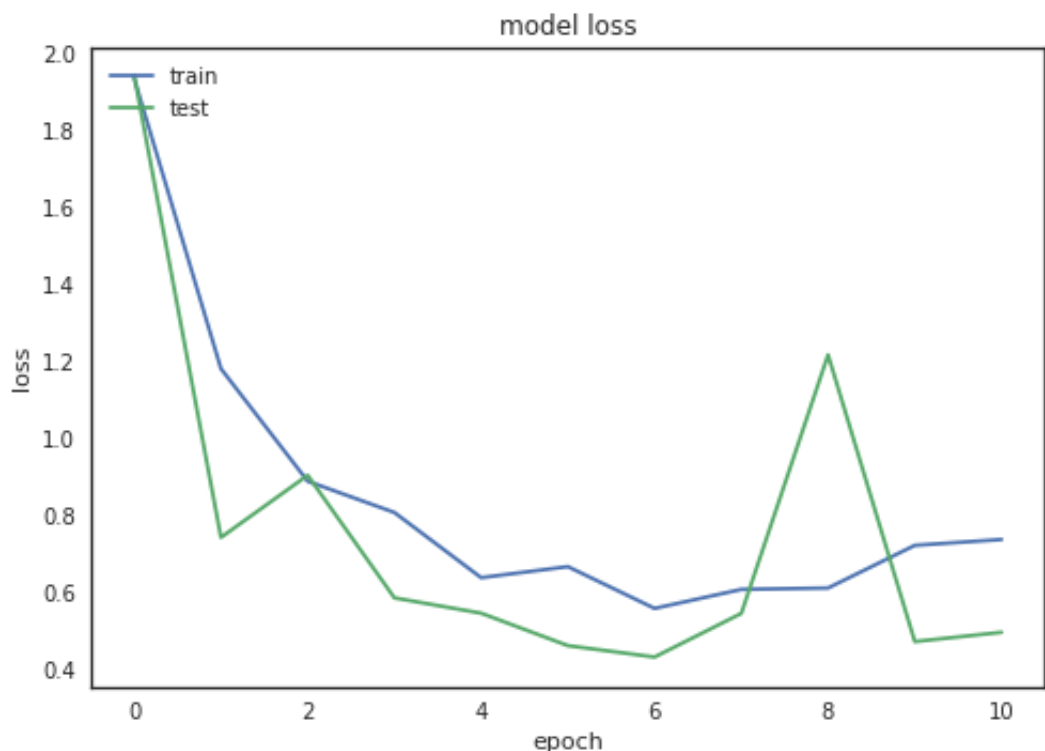
综合以上的改进过程, 可以看出对于从头搭建的homebrew model来说, 加入Dropout Layer可以有效的改善overfitting的情况, 减小learning rate对model的性能影响不大, 增加网络的深度可以在一定程度上提升模型的性能。最终我们获得了**log loss = 0.35, accuray=0.85**的模型。

Transfer Learning Model 的训练

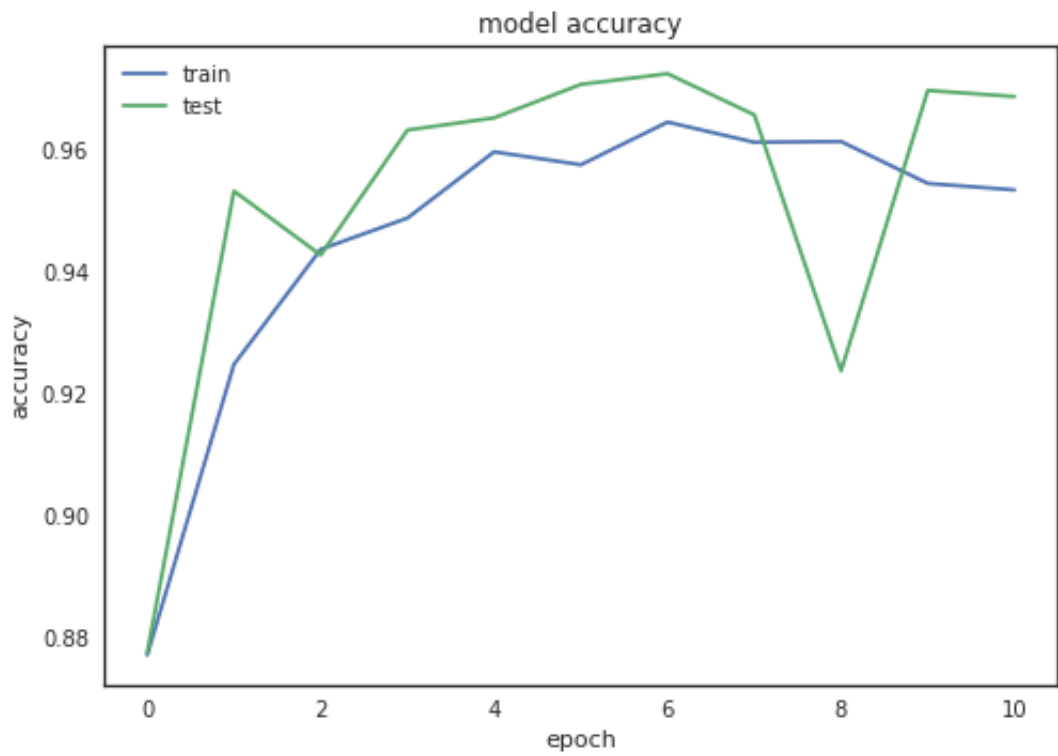
- 除了从零开始建立model, 我们也可以使用[Transfer Learning](#)的方法将已训练好的model参数迁移到新的model中去, 帮助新模型训练,加快模型的训练并优化模型。
- 在本项目中, 我们使用了已在上训练过的[VGG16](#), [VGG19](#)以及[ResNet](#)并保留了相应参数值的model作为Transfer Learning的起点, 作为新model的前端, 保持参数值不可变,用以提取图像的特征。之后加入可训练的若干FC层和Dropout层, 进行训练。详细的模型结构和参数可以参考[Jupyter Notebook](#)中的『Transfer Learning: Model Creation』代码块及其运行结果。因为所采用的这几个带有预训练权重的模型都包含了非常多的Convolutional Layer, 按照上面homebrew model的研究我们预测Transfer Learning将可以取得较好的结果。以下是各个模型的loss和accuracy曲线:

- VGG16

- loss (final 0.62)

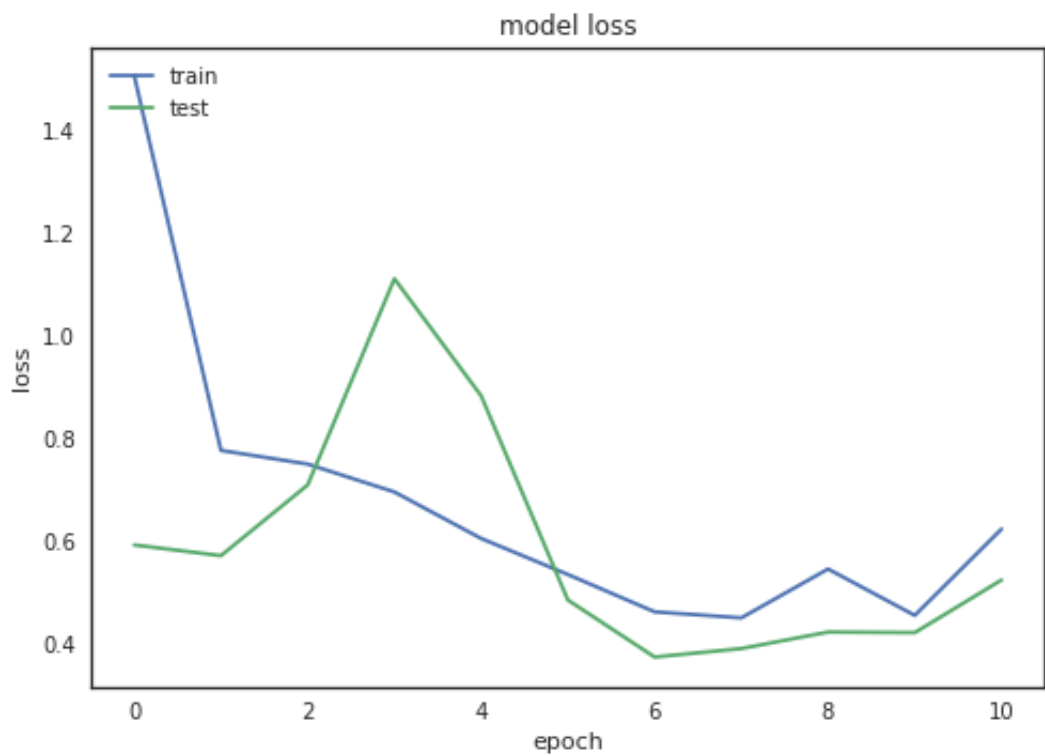


- accuracy (final 0.96)

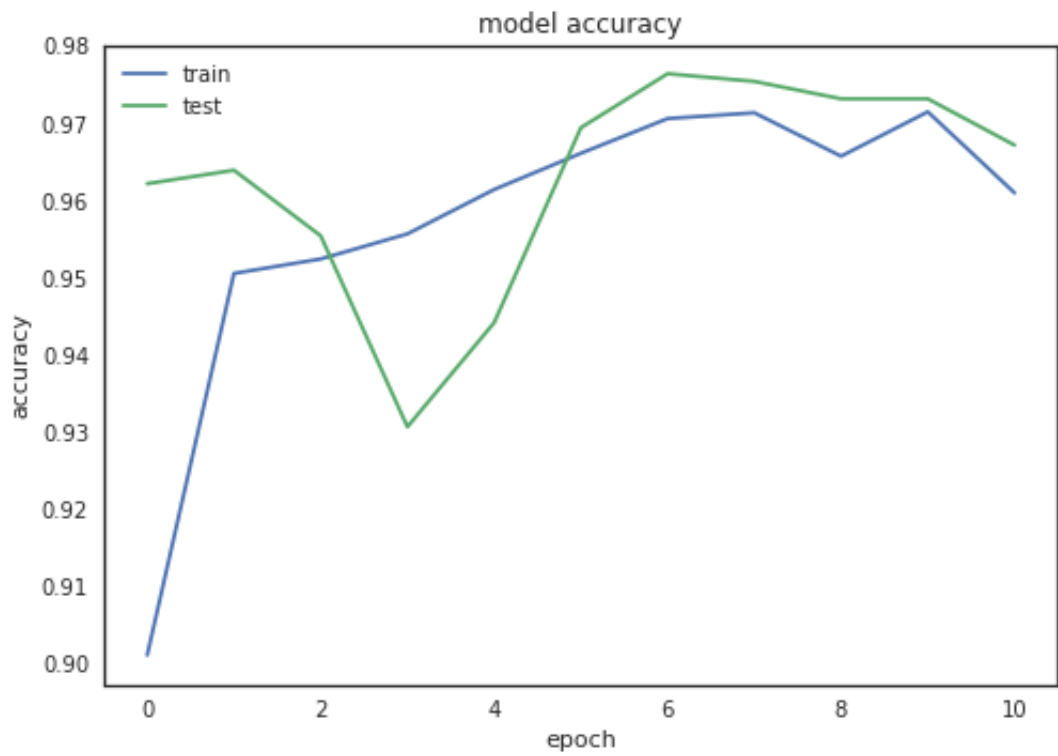


- VGG19

- loss (final 0.47)

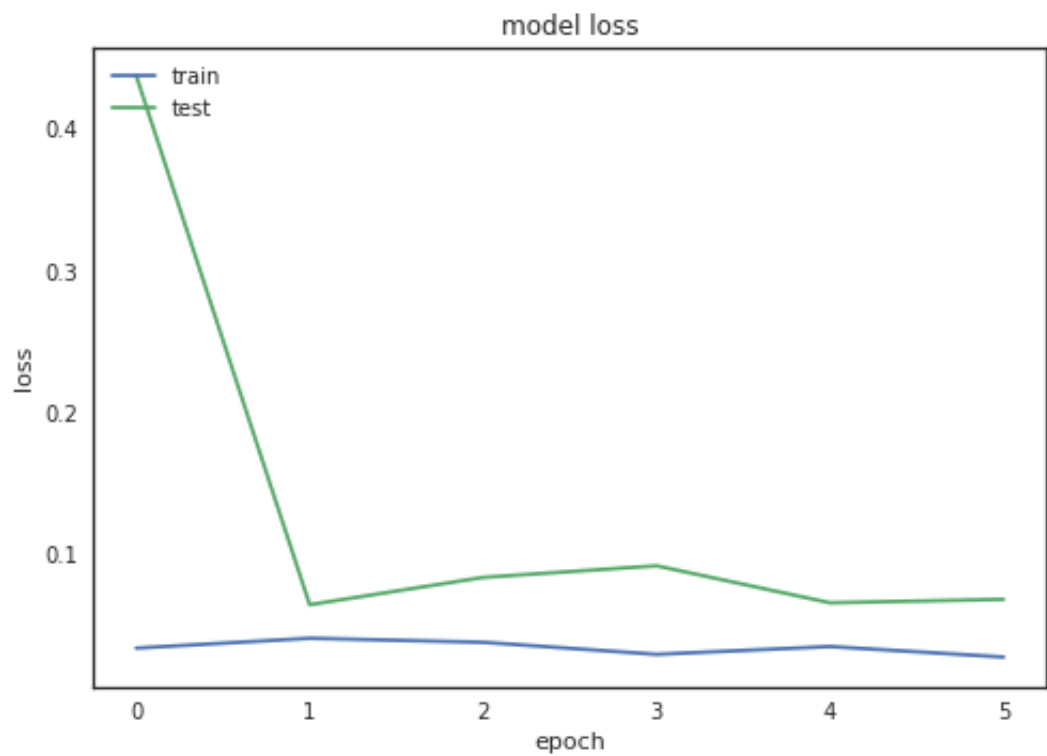


- accuracy (final 0.97)

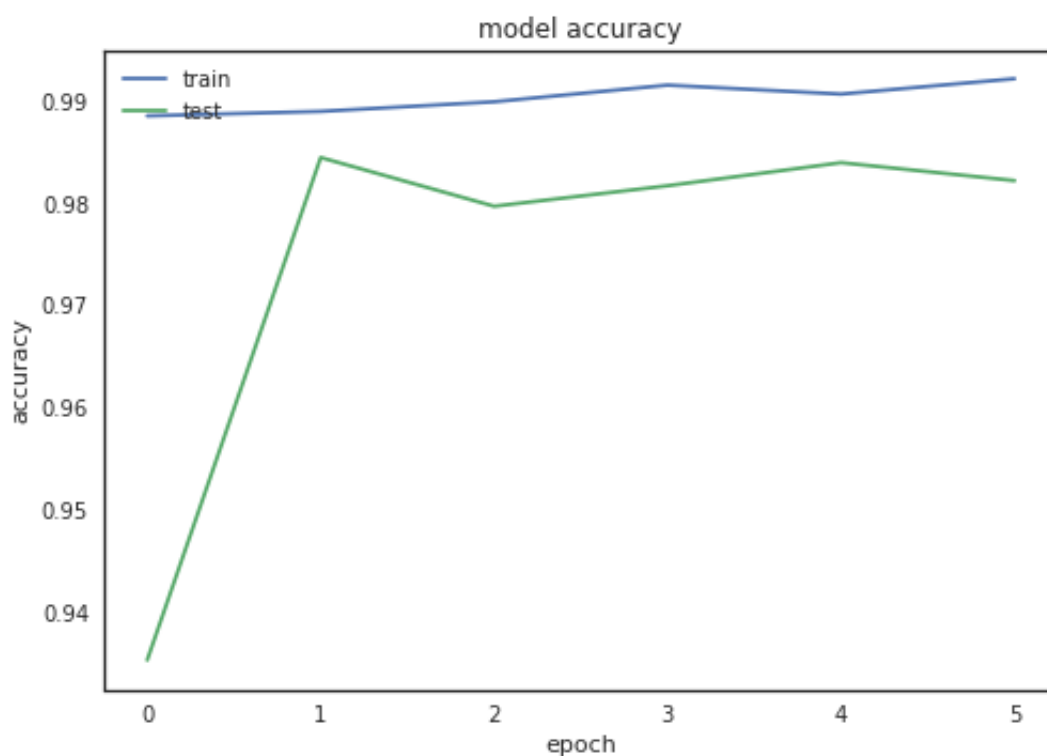


- ResNet

- loss (final 0.06)



- accuracy (final 0.98)



可以得到三个model都获得了非常高(>95%)的accuracy, 同时log loss也较小, 尤其是ResNet的log loss远小于其他model.

模型基准测试结果

我们使用最终获得的四个model(homebrew, VGG16, VGG19 and ResNet)对Kaggle提供的测试数据集进行了预测, 并将预测结果提交到了Kaggle进行验证。具体代码可以参考[Jupyter Notebook](#)。以下是从Kaggle获得的测试结果：

Submission and Description	Private Score	Public Score	Use for Final Score
VGG19_prediction.csv 2 hours ago by Wenhao VGG19		1.07107	<input type="checkbox"/>
VGG16_prediction.csv 2 hours ago by Wenhao VGG16		1.21746	<input type="checkbox"/>
ResNet50_prediction.csv 2 hours ago by Wenhao ResNet50		0.21334	<input type="checkbox"/>
homebrew_even_deeper_prediction.csv 2 hours ago by Wenhao home brew model		0.37260	<input type="checkbox"/>

可以看出整体的log loss都比本地的测试结果大一些。同时ResNet的log loss最小，Homebrew Model其次，均小于我们之前定义的阈值1.0。与Kaggle官方提供的排名对比，ResNet model的排名大概在800/1314的位置，homebrew model的排名大概在900/1314左右，与排名前列的log loss小于0.05的model相比还存在巨大的优化空间。

同时，为对模型的实际预测能力有一个直观的认识，我们也通过搜索引擎从互联网上获取了一些图片用于测试，以下是部分测试结果。测试代码可以参阅这个[Jupyter Notebook](#)

cat1.jpg

ResNet50 prediction: cat raw output: [1. 0.]

VGG16 prediction: cat raw output: [1. 0.]

VGG19 prediction: cat raw output: [1. 0.]

homebrew_even_deeper prediction: dog raw output: [0.4891772 0.51082283]



cat2.jpg

ResNet50 prediction: cat raw output: [1.00000000e+00 1.18650086e-16]

VGG16 prediction: cat raw output: [1. 0.]

VGG19 prediction: cat raw output: [1. 0.]

homebrew_even_deeper prediction: cat raw output: [0.84836632 0.15163369]



cat3.jpg

ResNet50 prediction: cat raw output: [1.00000000e+00 6.07020614e-11]
VGG16 prediction: cat raw output: [1. 0.]
VGG19 prediction: cat raw output: [1. 0.]
homebrew_even_deeper prediction: dog raw output: [0.40846953 0.5915305]



cat4.jpg

ResNet50 prediction: cat raw output: [1.00000000e+00 1.26669972e-17]
VGG16 prediction: cat raw output: [1. 0.]
VGG19 prediction: cat raw output: [1. 0.]
homebrew_even_deeper prediction: dog raw output: [0.11739068 0.88260925]



cat5.jpg

ResNet50 prediction: cat raw output: [1.00000000e+00 3.47572065e-19]

VGG16 prediction: cat raw output: [1. 0.]

VGG19 prediction: cat raw output: [1. 0.]

homebrew_even_deeper prediction: dog raw output: [0.00665362 0.99334645]



dog1.jpg

ResNet50 prediction: dog raw output: [4.80636930e-31 1.00000000e+00]

VGG16 prediction: dog raw output: [0. 1.]

VGG19 prediction: dog raw output: [0. 1.]

homebrew_even_deeper prediction: dog raw output: [4.93089610e-04 9.99506950e-01]



dog2.jpg

ResNet50 prediction: dog raw output: [0. 1.]

VGG16 prediction: dog raw output: [0. 1.]

VGG19 prediction: dog raw output: [0. 1.]

homebrew_even_deeper prediction: dog raw output: [1.55710477e-05 9.99984384e-01]



dog3.jpg

ResNet50 prediction: dog raw output: [1.55489235e-15 1.00000000e+00]

VGG16 prediction: dog raw output: [0. 1.]

VGG19 prediction: dog raw output: [0. 1.]

homebrew_even_deeper prediction: dog raw output: [0.00111197 0.99888808]



dog4.jpg

ResNet50 prediction: dog raw output: [1.37699844e-05 9.99986172e-01]
VGG16 prediction: cat raw output: [1. 0.]
VGG19 prediction: dog raw output: [0. 1.]
homebrew_even_deeper prediction: dog raw output: [0.08364888 0.91635114]



dog5.jpg

ResNet50 prediction: dog raw output: [7.30816217e-28 1.00000000e+00]
VGG16 prediction: dog raw output: [0. 1.]
VGG19 prediction: dog raw output: [0. 1.]
homebrew_even_deeper prediction: dog raw output: [0.00122691 0.99877304]



```
dog6.jpg
ResNet50 prediction: dog raw output: [ 0.01392018 0.98607987]
VGG16 prediction: dog raw output: [ 0. 1.]
VGG19 prediction: dog raw output: [ 0. 1.]
homebrew_even_deeper prediction: dog raw output: [ 0.00615013 0.99384993]
```



相较而言，采用带有预训练权重的model的表现较为出色，对绝大多数的测试图片都给出了正确的预测结果 (VGG16在dog4图片给出了错误的预测)。而homebrew model在cat4和cat5图片上给出了错误的结果。因为homebrew model的结构相对简单，这一结果也并没有出乎意料。

总结

在本项目中，我们使用了深度学习的方法来处理猫狗图片分类的问题，采用两种不同的路径搭建,训练并验证了CNN模型,并将预测数据上传到Kaggle进行了测试。

Homebrew Model

- 方案总结：首先使用Keras搭建了具有三层Convolutional layer的model作为初始base model, 之后基于这个model做了相应的tunning,尝试调整drop out rate, learning rate, 以及添加更多的卷积层。经过试验可以看出对于从头搭建的homebrew model来说，加入Dropout Layer可以有效的改善overfitting的情况，减小learning rate对model的性能影响不大，增加网络的深度可以在一定程度上提升模型的性能。我们发现增加Convolutional Layer的层数会改善model的性能，但同时会增加model的计算时间。
- 比较困难的点：目前还不清楚的是对特定大小的数据集来说，对Convolutional Layer的层数是否有一定的限制，即随着Convolutional Layer的层数，model的性能的提升是否存在一个可以预知的上限。同时也需要研究在layer层数更多之后，early stop和dropout还能否有效的避免overfitting.
- 后续可能的改进: 还可以尝试数据增强的方法来扩充训练集中样本的数量，观察是否可以提高模型的泛化性能。同时更多的数据可能可以支持更深的网络。

Transfer Learning model

- 方案总结：对于Transfer Learning model, 我们发现利用带有预训练权重的模型可以提高model的训练效率，并取得较高的accuracy.
- 比较困难的点：是在本项目中VGG16与VGG19 model最终的log loss甚至要高于结构简单的多的homebrew model，这是我们所没有预料到的，具体原因值得继续深入研究。*后续可能的改进：同时可以考虑"解冻"model的部分layer权重并参与训练，观察这类fine-tuning是否可以改善model的表现。



* title image source: <https://www.pinterest.com/pin/365636063472803484/>