机器学习工程师纳米学位毕业项目

猫狗大战

周志锵

2017 年 12 月 1 日

第 1 章 定义

1.1 项目概述

这个项目来自于 Kaggle 上的比赛 Dogs vs. Cat，这个比赛有两个版本，第一版的比赛举行于 2013 年，当时的神经网络以至于整个 AI 行业还没有现在那么火热，但现在这一切都改变了，尤其在深度学习领域和计算机视觉领域，最近几年这两个领域取得了巨大的进步，以至于这个当时很困难的项目现在只能算是个计算机视觉领域的入口项目（2013 年这个项目的第一第二名是有奖品的，现在只有荣誉了）。

2013 年的比赛，第一名的得分是 0.98914，这在当时应该是非常高的分数了，作者运用了 OverFeat Net，这个框架大体类似于 AlexNet[0]。区别在于：

1、训练时输入大小固定，测试时用多尺度输入；

2、没有对比度归一化；

3、采用没有 overlap 的 max pooling；

4、前面两层的 feature map 的个数比 AlexNet 中多。

现在，通过运用更先进的深度学习模型 InceptionV3[5]，ResNet[6]，Xception 等，以及计算能力更加强大的机器，这个分数在 2017 年已经可以很轻易获得了。

项目中将会建立一个卷积神经网络（Convolutional Neural Network, CNN）作为训练模型，通过改进模型并且增强数据，最后调整超参数（Hyper parameters）对网络模型做优化，以期达到预定准确率目标。同时会尝试运用迁移学习技术获取更好的测试集 loss 分数。

项目中选择的数据是 Kaggle 比赛 Dogs vs. Cat Redux 中提供的 25000 张训练集图片和 12500 张测试集图片。

1.2 问题陈述

项目中选择的数据是从真实世界中采集的。一个挑战在于真实世界中的图片中，猫狗的品种毛色形态多种多样，受采集图片时候的光照，以及采集图片的设备质量影响，图片的质量也参差不齐，这些因素都会增加识别的图片的难度。

识别图片中是猫或者是狗是个二分类问题。输入的数据是由图片像素组成的数字数组序列，输出的目标是一个表示二分类的 0 或者 1 浮点数字 a。a >= 0 and a < 0.5 表示这是猫，a < 1.0 and a >= 0.5 表示这是狗。

1.3 评价指标

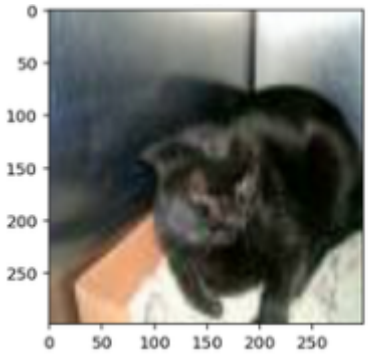
这个项目的评价指标可以是识别猫狗的准确率（Accuracy），即正确识别图片中的动物是猫还是狗的数量除以图片的总数或者 logloss。但是由于 Kaggle 提供的测试数据集并没有包含人工标注的标签（label），而对 12500 张测试集图片做人工标注很费时间，时间很贵的，因此项目将评估指标指定为 logloss，即：

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

第 2 章 分析

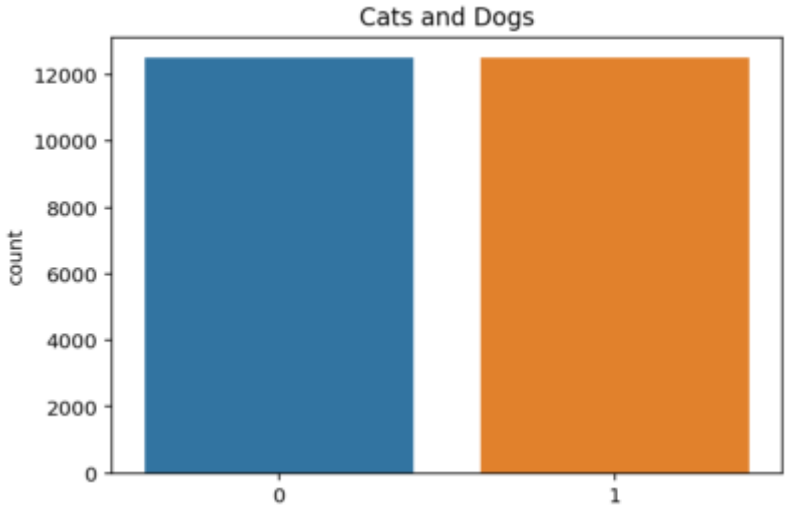
2.1 数据可视化

项目的数据集有两个部分，训练数据集和测试集，训练集中文件名包含了图片中动物分类的人工标注，经过数据预处理后可以作为训练集的标签使用。测试集不包含任何标注，在确保训练过程中模型不接触到这个数据集的前提下可以用于最后测试模型在真实世界中的分类效果。



图：训练集中的一张图片

观察图片可以发现，训练集中不同图片的质量差别相当大，有的图像甚至人工也很难分辨。比如说上图。



图：训练集中猫狗的分布条状图

观察图表可以发现，训练集中二分类的数量一致，这经过充分的随机打乱（shuffle）后，可以达到均衡分布的目的。

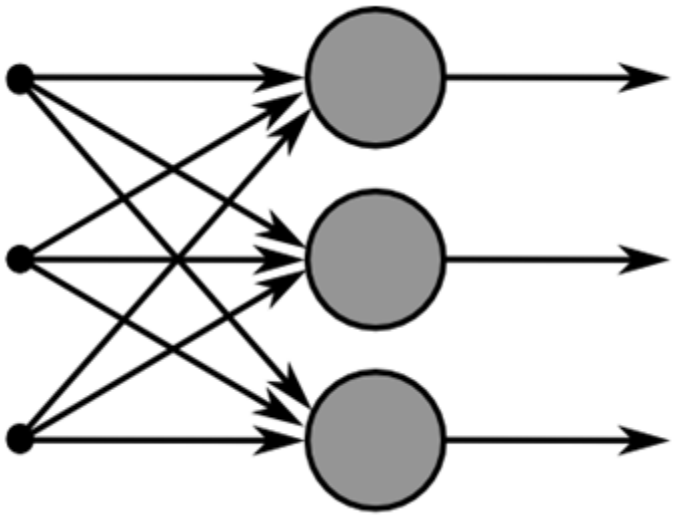
2.2 算法和技术

2.2.1 分类算法

根据前面的分析可知，识别猫狗本质上是个分类问题。用于分类问题的算法很多，逻辑回归、SVM、决策树、神经网络等。

项目中使用的是卷积神经网络算法。

2.2.2 神经网络



图：单层神经元网络

人工神经网络（英文：artificial neural network，缩写 ANN），简称神经网络（英文：neural network，缩写 NN）或类神经网络，是一种模仿生物神经网络(动物的中枢神经系统，特别是大脑)的结构和功能的数学模型或计算模型，用于对函数进行估计或近似。[1]

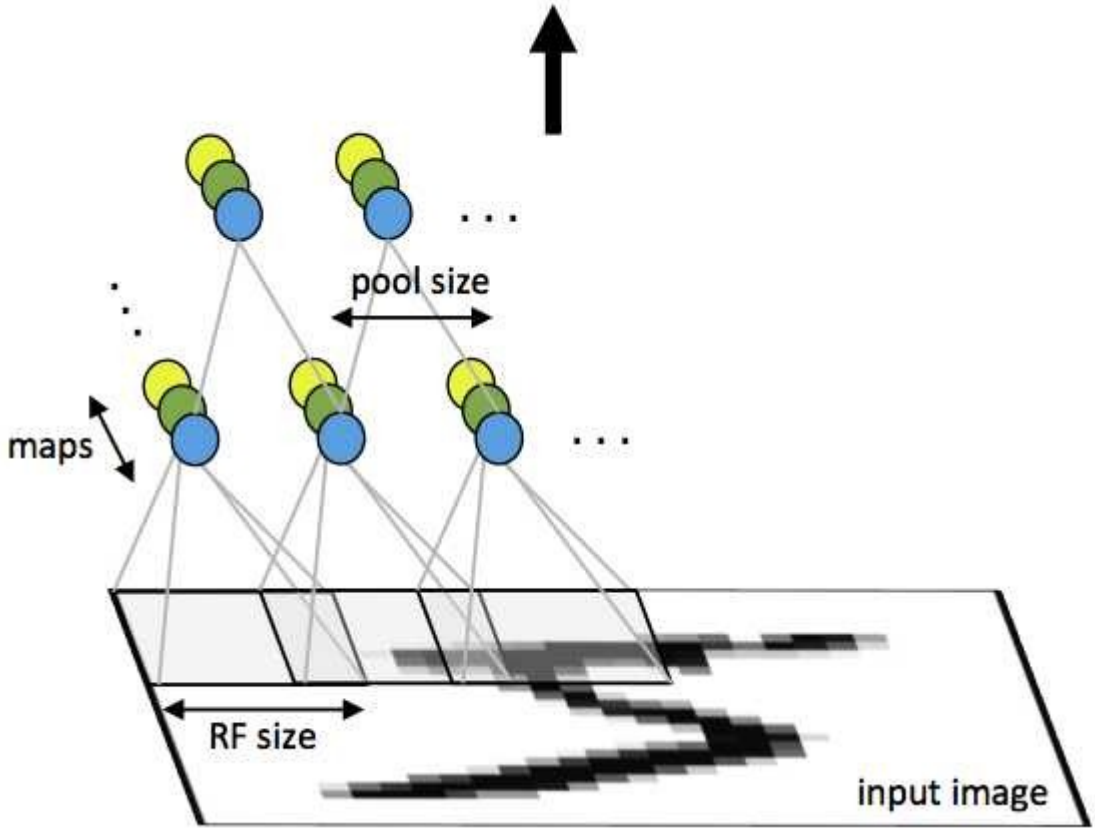
全连接网络（Fully Connected Network）可以通过平整化（Flatten），把图像当作一个数组，并把像素值当作预测图像中数值的特征。通过前向训练和后向反馈得到权值 w 和 b，让网络理解理解图中发生了什么，但这非常的困难。

2.2.3 卷积神经网络

卷积神经网路（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。[2]

上面提到使用全连接网络训练图像非常困难，那么我们可以尝试从图像中提取特征，做法就是使用卷积核。这个卷积核对图像进行操作，相当于对图像进行了低通滤波，例如我们要识别图像中某些特定的特征：转角、直线等，那么定义一个转角或者直线的卷积核，然后对这个图像做卷积操作，如果这个卷积核对图像有很高的输出，那么说明这个图像具有这个卷积核的特征。

在训练卷积神经网络的某一个卷积层时，我们实际上是在训练一系列的滤波器。比如对于一个 32x32x3 的图像，如果我们在 CNN 的第一个卷积层定义训练 12 个滤波器，那这一层的输出便是 32x32x12。按照不同模型，我们可以对这个输出做进一步的处理，这包括激活函数，池化，全连接等。



图：卷积神经网络 [8]

综上所说，所谓卷积神经网络，就是让网络自动的对于一张图片学习出最好的卷积核以及这些卷积核的组合方式，这些卷积核和组合方式构成了这个网络对这张图片的特征判断，然后这个模型就可以用来进行分类任务了。

2.2.4 激活函数

如果不用激活函数，那么每一层的输出都是上层输入的线性函数，那么无论网络有多深，输出总是输入的线性组合，这样与没有隐藏层相当，设计更深层的网络就没有意义了。

引入了非线性函数作为激活函数以后，输出不再是输入的线性组合，可以逼近任意函数。最早的激活函数是 sigmoid，sigmoid 具有以下的特点：

1、 容易求导；

2、 容易出现 Gradient Vanishing

3、 函数的输出不是 Zero-Centered

4、 幂运算相对较耗时

为解决第３个问题，于是有 tanh 函数，但是 Gradient Vanishing 和运算耗时的问题依然存在。

那么如果解决这些问题？答案是使用 Relu 函数作为激活函数。

Relu 函数的公式是：Relu = Max(0, x)，可以看出来这个函数其实就是一个取最大值的函数，这个函数有这些特点：

1、 在正区间解决了 Gradient Vanishing 问题；

2、 计算速度相当快，只需要判断输入是否大于０；

3、 收敛速度远快于 sigmoid 和 tanh；

4、 输出不是 Zero-centered；

5、 某些神经元可能永远都不会被激活，导致相应的参数不会被更新。

解决４的问题理论上可以采用 Leaky Relu 或者 Elu 函数，Elu 理论上同时还能解决５的问题，然而在实际使用中，并没有好的证据证明 Leaky Relu 或者 Elu 函数优于 Relu。

Relu 函数是目前搭建神经网络时最常用的激活函数，项目中会使用这个函数。

2.2.5 优化器

当遇到数据量大的时候，会使用随机梯度下降算法（Stochastic gradient, SGD）。SGD 非常直观，就是随机拿出一些数据做梯度下降，但是这个算法存在一些问题：

1、 收敛速度跟学习率的关系很大，大了容易震荡，小了收敛很慢；

2、 没有考虑稀疏性；

3、 容易陷入局部最优或者是鞍点。

现在有了一些改进的算法，比如 AdaGrad[11]、RMSProp[12]、Adam[13]等。

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
G_t &\leftarrow G_t + \mathbf{g}_t \odot \mathbf{g}_t \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t
\end{aligned}$$

图：AdaGrad 算法

AdaGrad 算法的做法是让学习速率选择的更加容易，通过利用以前的梯度信息判断对应的特征 j 是否经常被更新。因此对稀疏的数据尤其适合。但是随着训练的递增，由于梯度 G_t 会一直累积，对应的特征 θ 会趋向于不更新。

那么要解决这个问题，办法是对累计信息增加一个指数衰减 γ，这样最开始的那些梯度就会渐渐的对后面的更新不产生影响了，梯度 G_t 不会再一直累积，也就是说 AdaGrad 的问题就不存在了，这就是 RMSProp 算法。

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t
\end{aligned}$$

图：RMSProp 算法

Adam 算法利用了 AdaGrad 和 RMSProp 在稀疏数据上的优点，然后对初始化偏差的修正表现也更好，速度非常快。现在很多深度学习的问题都优先采用 Adam 算法了，包括视觉识别领域和强化学习领域。

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
\mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\alpha &\leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t} \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t + \epsilon}}
\end{aligned}$$

图：Adam 算法

项目中会使用这几种优化器中表现最好的 RMSProp 和 Adam 算法做为优化器。

2.2.6 技术

项目中将会使用 Google 开源深度学习框架 TensorFlow 以及 Keras，逐步实现一个经过优化的类似 VGG13[9]卷积神经网络，最后尝试使用更先进的网络和预

先训练好的参数对测试集数据做出更好的分类预测。

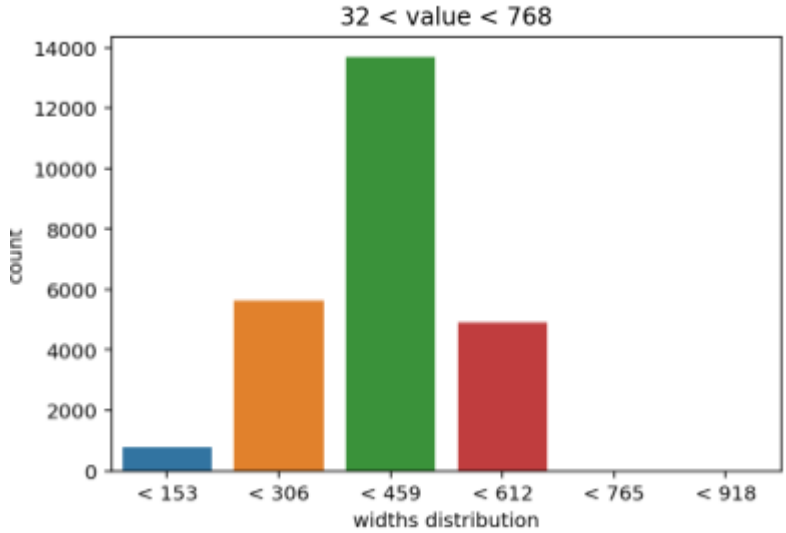
2.3 基准指标

考虑到复杂的神经网需要使用高性能 GPU 计算机，同时需要耗费很长的时间训练，而使用迁移学习需要高性能大显存 GPU，而目前的实验条件难以达到这些条件，因此项目使用一个较为简单的神经网络做训练，设定的目标为 Kaggle Leaderboard 的 Top10%左右，即 0.09 >= logloss >= 0.06。

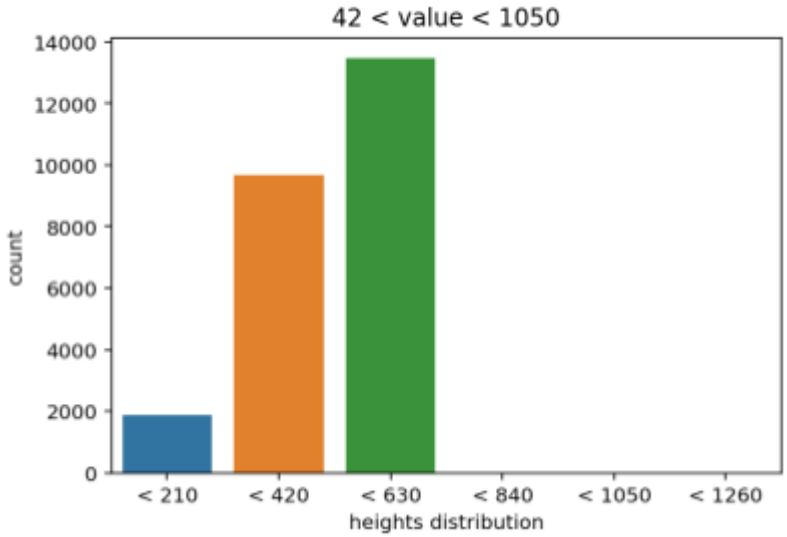
第 3 章 具体方法

3.1 图片预处理

3.1.1 剪裁和缩放



图：训练集图片的宽度分布，大部分图片集中在 306~459 的区间



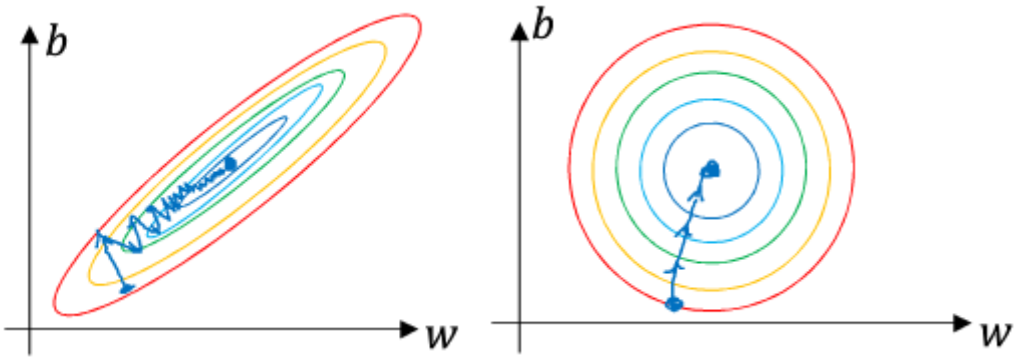
图：训练集图片的高度分布，大部分图片集中在 420~630 的区间

训练集图片宽高分布很不均匀，为了确保训练数据的一致性，对图片做剪裁

和缩放，限定图片的宽高在 299。

3.1.2 归一化

项目中令 data = data / 255.0，限制数据在 0.0 到 1.0 之间



图：归一化让模型更容易被学习

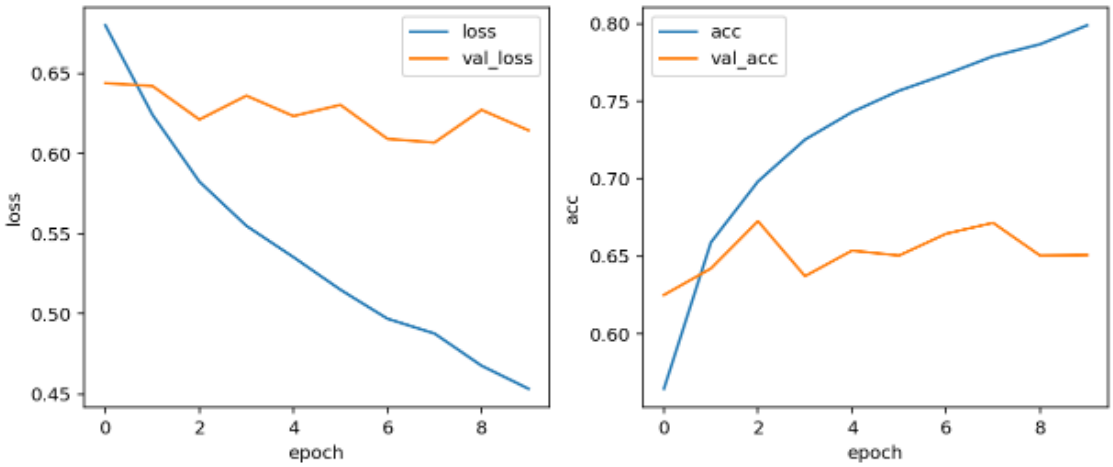
3.2 实现

使用简单的 15 层卷积神经网络

Layer (type)	Output Shape	Param #
conv2d_121 (Conv2D)	(None, 299, 299, 32)	6176
max_pooling2d_19 (MaxPooling	(None, 149, 149, 32)	0
dropout_12 (Dropout)	(None, 149, 149, 32)	0
conv2d_122 (Conv2D)	(None, 149, 149, 16)	8208
max_pooling2d_20 (MaxPooling	(None, 74, 74, 16)	0
dropout_13 (Dropout)	(None, 74, 74, 16)	0
flatten_5 (Flatten)	(None, 87616)	0
dense_14 (Dense)	(None, 512)	44859904
dropout_14 (Dropout)	(None, 512)	0
dense_15 (Dense)	(None, 256)	131328
dropout_15 (Dropout)	(None, 256)	0
dense_16 (Dense)	(None, 64)	16448
dropout_16 (Dropout)	(None, 64)	0
dense_17 (Dense)	(None, 1)	65
activation_99 (Activation)	(None, 1)	0

Total params: 45,022,129
Trainable params: 45,022,129
Non-trainable params: 0

3.3 改进



图：简单卷积神经网络的表现曲线

没有进行改进之前，神经网络表现出比较明显的过拟合特征，如上图。

这个简单的模型获得了 65.07%准确，对于简单的 CNN 网络来说，这个数字似乎已经不算低了。纯粹猜测的准确大概 10%左右。以下通过对网络进行改进增加训练准确率。

3.3.1 增加神经网络层数

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					

增加网络层数，使用更复杂的神经网络：类似 VGG13。

Layer (type)	Output Shape	Param #
conv2d_123 (Conv2D)	(None, 299, 299, 32)	896
conv2d_124 (Conv2D)	(None, 299, 299, 32)	9248
max_pooling2d_21 (MaxPooling	(None, 149, 149, 32)	0
conv2d_125 (Conv2D)	(None, 149, 149, 64)	18496
conv2d_126 (Conv2D)	(None, 149, 149, 64)	36928
max_pooling2d_22 (MaxPooling	(None, 74, 74, 64)	0
conv2d_127 (Conv2D)	(None, 74, 74, 128)	73856
conv2d_128 (Conv2D)	(None, 74, 74, 128)	147584
max_pooling2d_23 (MaxPooling	(None, 37, 37, 128)	0
conv2d_129 (Conv2D)	(None, 37, 37, 256)	295168
conv2d_130 (Conv2D)	(None, 37, 37, 256)	590080
max_pooling2d_24 (MaxPooling	(None, 18, 18, 256)	0
flatten_6 (Flatten)	(None, 82944)	0
dense_18 (Dense)	(None, 256)	21233920
dropout_17 (Dropout)	(None, 256)	0
dense_19 (Dense)	(None, 256)	65792
dropout_18 (Dropout)	(None, 256)	0
dense_20 (Dense)	(None, 1)	257
activation_100 (Activation)	(None, 1)	0

Total params: 22,472,225
Trainable params: 22,472,225
Non-trainable params: 0

3.3.2 正则化

在训练神经网络的时候，经常会遇到过拟合的问题，过拟合来源于高方差（high variance）。要解决这个问题，一个方法是让模型训练更多的数据，这个方

法受限太多不现实。另外一个方法是采用正则化（regularization）。常用的正则化方法有：

L2 regularization (weight decay)

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2,$$

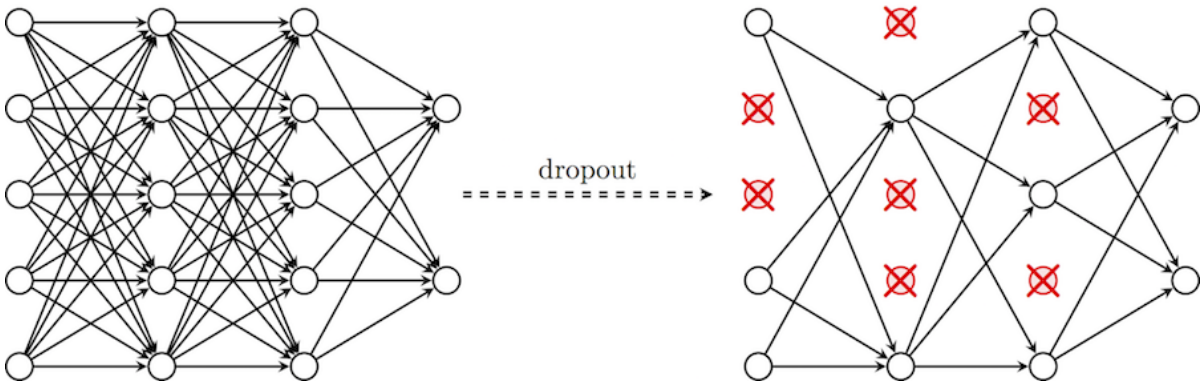
L1 regularization

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|.$$

项目中主要应该了 Dropout、数据增强以及早期停止这些正则化方法。

3.3.3 Dropout

通过 Dropout[10]，模型会遍历所有的层（layer），然后以一定的概率删除这个层的某些节点。



图：Dropout 的实现过程[3]

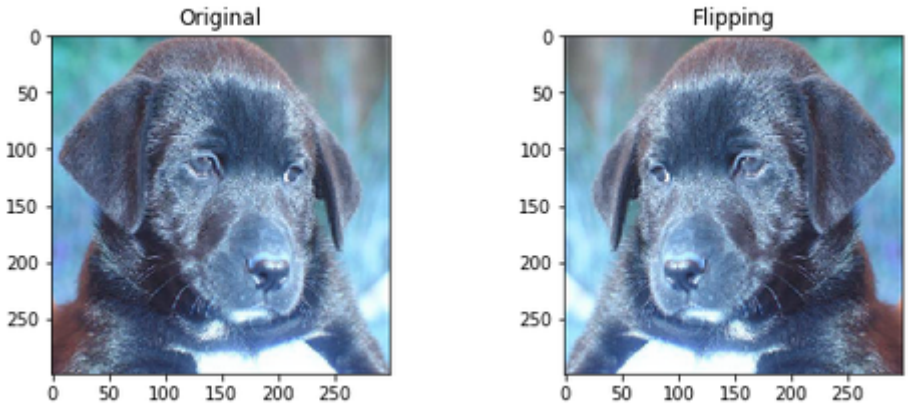
Dropout 的实现可以降低神经网络中节点对前面层任意节点特征的依赖，从而达到减轻过拟合的目的[4]。

3.3.4 数据增强

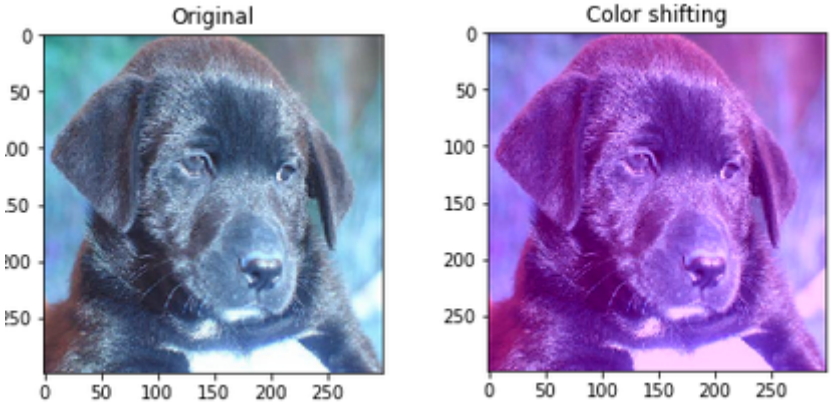
获取新的训练数据的代价总是很高的，通过对图片做数据增强，可以达到给

训练集增加新的训练数据的目的。

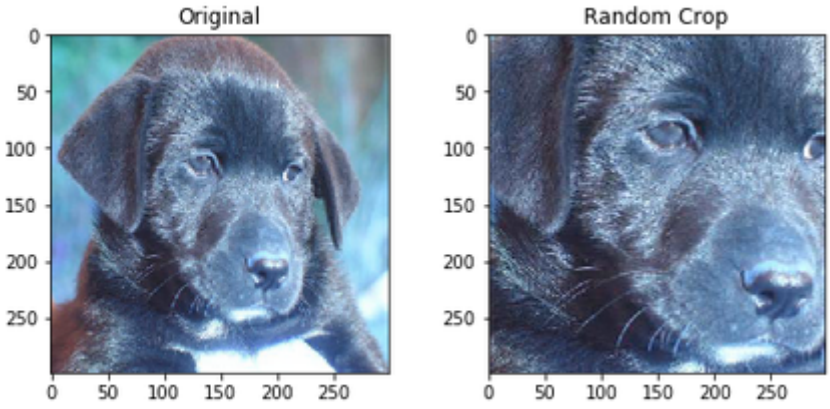
常用的数据增强方法有，翻转，颜色改变和随机裁剪。



图：水平翻转



图：颜色改变



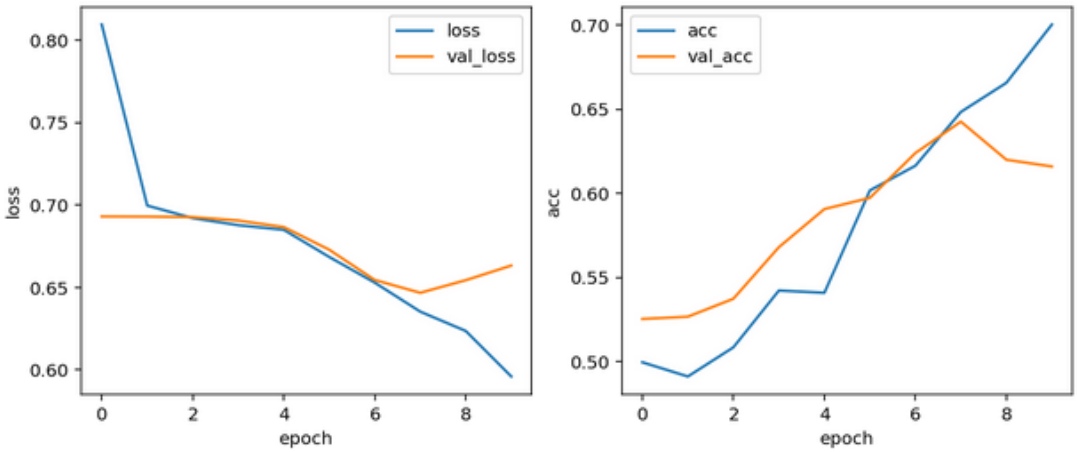
图：随机裁剪

项目中对所有的图片做 180 度水平翻转，这相当于增加了一倍的训练数据。

然后对所有图片再做一次颜色改变（R+20, G-19, B+20），这相当于又增加

了两倍的训练数据。

3.3.5 早期停止 (early stopping)



图： 训练集 loss 和验证集 loss 的曲线图

观察上图可发现，验证集 loss 在训练的第 6 代时是最低的，而第 7 代开始验证集 loss 从减少转变成增加，那么早期停止的做法就是在训练到第 6 代左右，停止训练，此时模型的 loss 是最好的。

3.3.6 超参数调优

使用早期停止这个正则化方法可以减少对其他超参数调优的时间。受限于机器的计算速度，项目中只对 batch_size 作调优，epochs 限制在 10 以内，学习率（learning rate）设定为 1e-4。

然而这样的初始化设置导致 epochs 全部运行完成后模型还没有收敛，在更深层的网络模型下还出现的较大的震荡。解决方案是增加训练的代数和降低学习率。

对于 batch_size 受限于 GPU 显存的大小的限制，项目中设置为 32。

3.3.7 迁移学习

在计算机视觉领域，已经存在了诸如 ImageNet 或者 MSCOCO 这样的超大规

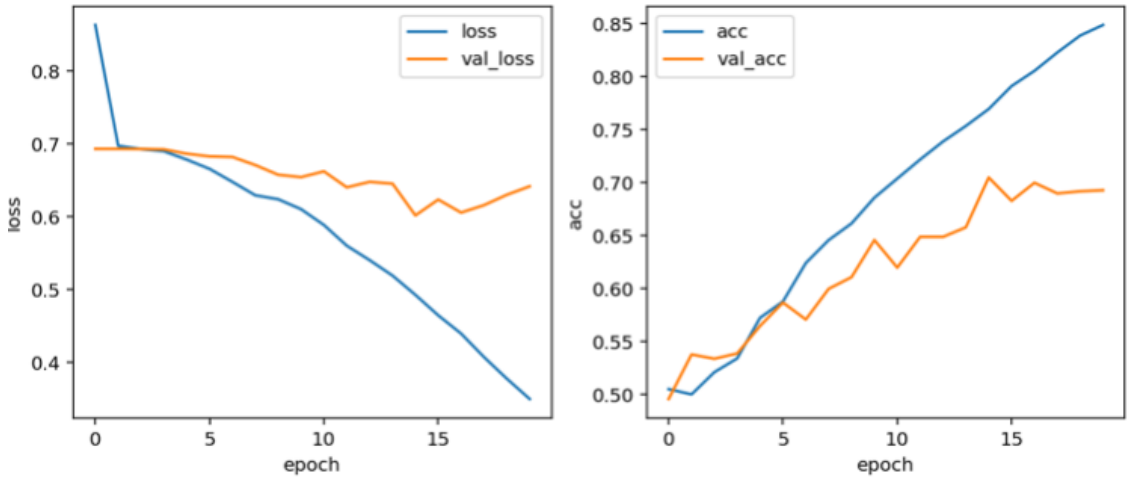
模数据集的预先训练好的模型参数（pretrained weights），直接使用这些参数作为项目的初始化参数，在这个基础上训练模型，这就相当于模型直接在初始化阶段就拥有了这些超大规模数据集预先训练好的模型参数的知识（knowledge），大大的节省了训练时间同时规避了数据集规模不足的问题。

项目中使用了 ResNet50 训练好的 ImageNet 参数作为初始化参数。

第 4 章 结果

4.1 模型评价与验证

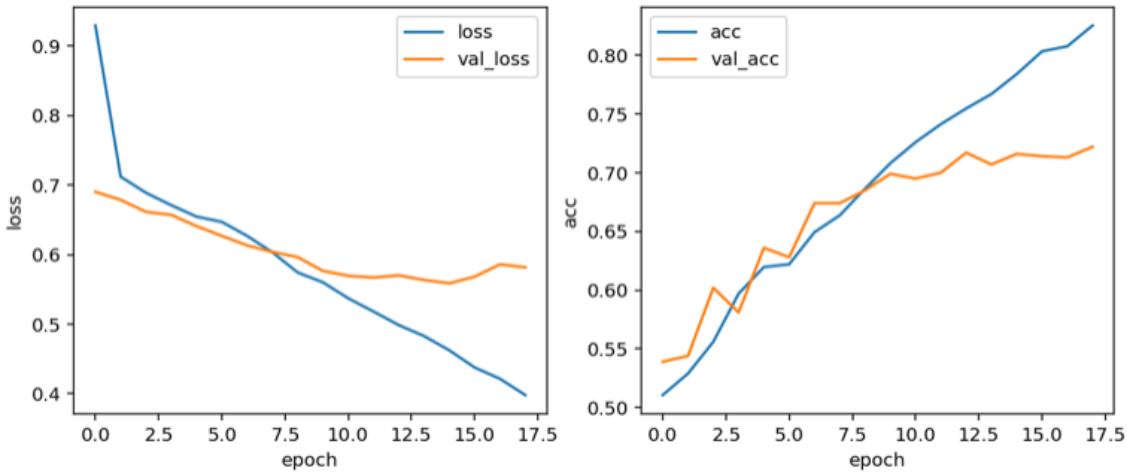
4.1.1 简单卷积网络



图：验证集 loss 0.6417 验证集 accuracy 0.6930

观察曲线图发现，训练集与验证集偏离较大，这时需要更好的训练模型。

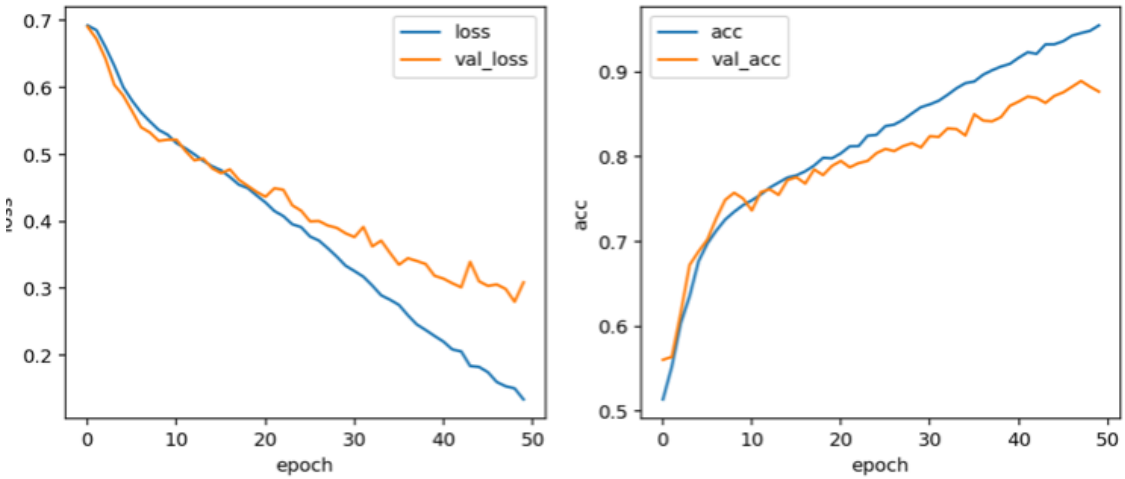
4.1.2 更深的卷积神经网络



图：验证集 loss 0.5817 验证集 accuracy 0.7220

观察曲线发现，这个模型的曲线偏离相对之前有改善，更深的网络是有效的，那么进一步可以尝试增加更多的数据对这个网络做训练。

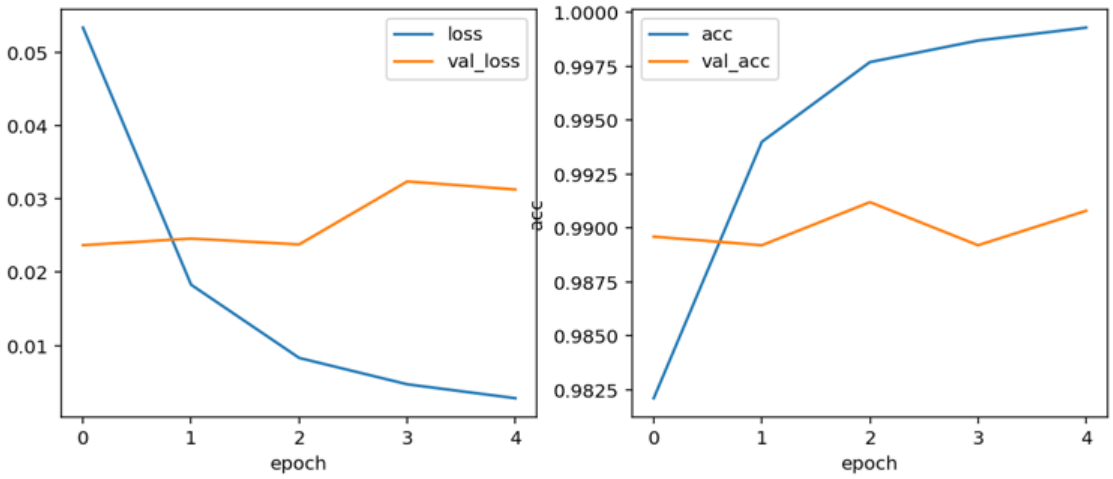
4.1.3 更深的卷积神经网络（数据增强）



图：验证集 loss 0.3087 验证集 accuracy 0.8767

观察曲线发现，这个模型的数据拟合比之前做的更好了，增强数据是有作用的，但是训练时间就拉长了很多了，在资源有限制的情况下尝试迁移学习。

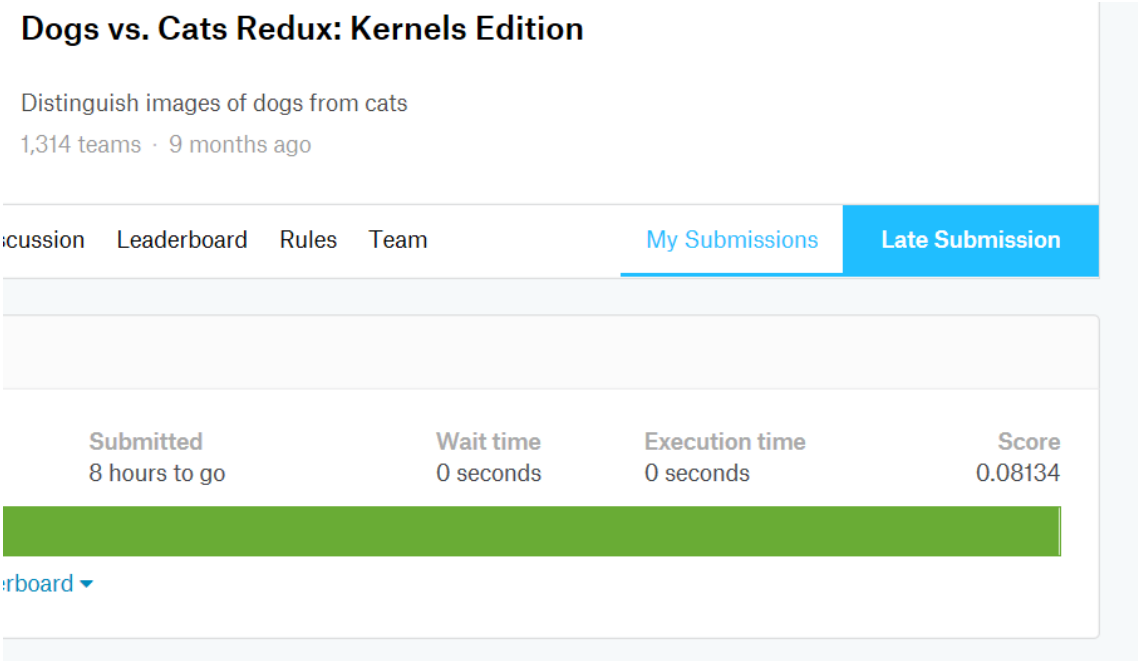
4.1.4 迁移学习+Fine-tune



图：验证集 loss0.0313 验证集 accuracy 0.9908

实验设备显卡为 GTX1070，25000 张图转为 ResNet50 特征集后，8G 显存就被全部占满了。

4.2 结果分析

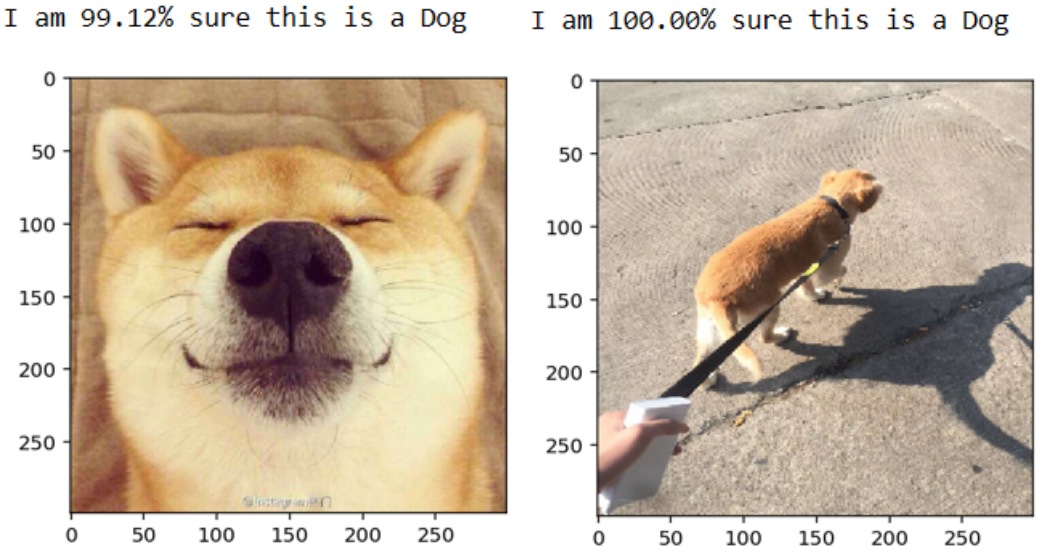


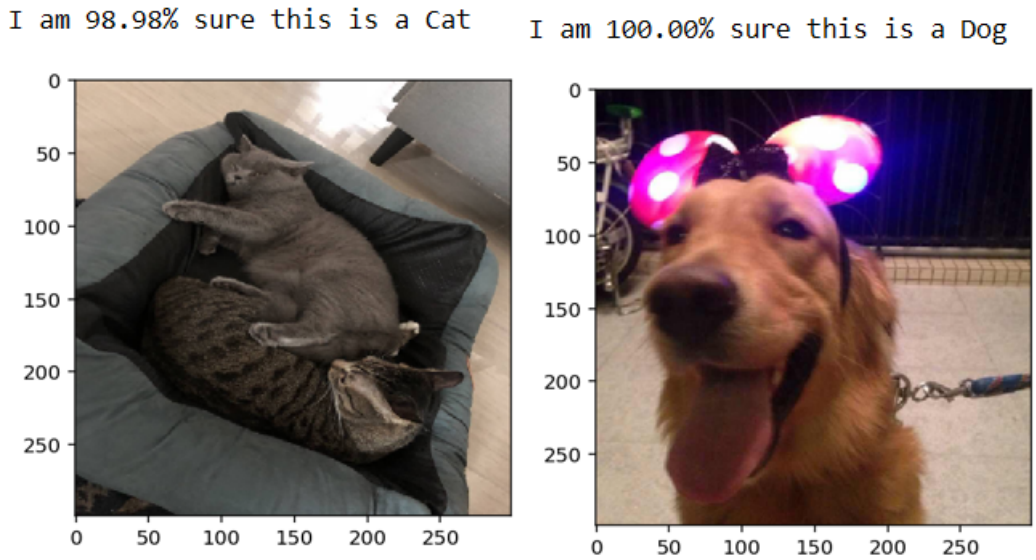
对于训练集的数据，使用迁移学习后，准确率达到了 99.08%，logloss 达到 0.08，已经达到预期目标。

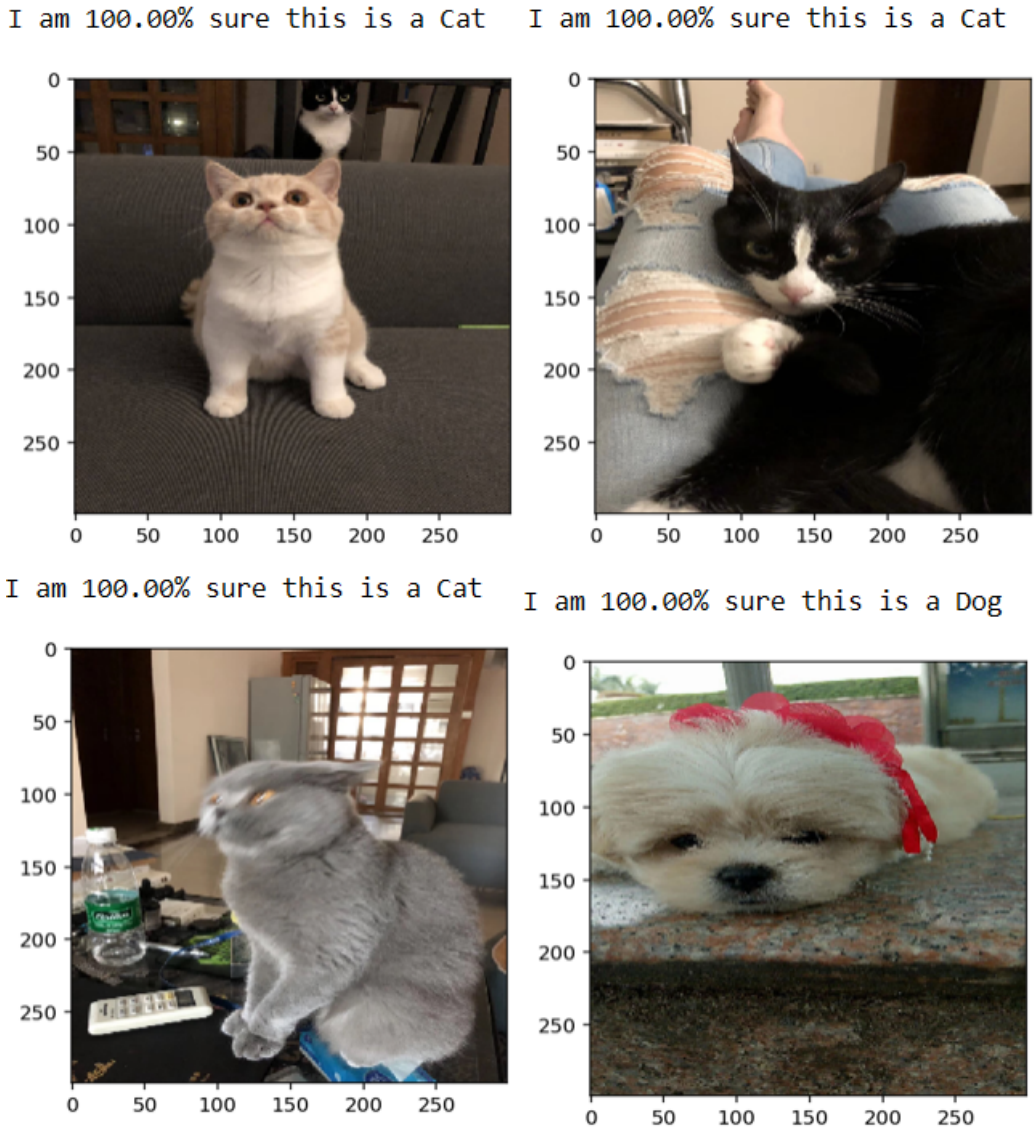
第 5 章 结论

5.1 应用于真实世界的猫狗

在训练完成之后，将模型应该于平常街上宠物店能看见到的猫狗，下图所示图片均是来自 wechat 好友头像以及朋友圈炫耀。







模型已经能准确的分辨真实世界的猫狗，已经具备实用价值。

5.2 总结

项目达成预定目标，即预测准确率达到 90%以上，logloss 达到 0.08。

在训练数据的过程中，遇到了各种问题，比如：

1、对于简单的模型，val_loss 共 loss 偏离很大，这表明模型过拟合了；

2、最开始限定所有的训练代数上限都是 10，然而观察曲线图发现基本上还没收敛训练就结束了，解决办法是增加训练的代数；

3、增加训练代数后，观察更复杂的模型发现 val_loss 震荡很大，这表明学习率的设置太大了，于是减小 RMSProp 优化器的学习率；

4、增加训练代数并且调整学习率后发现，loss 曲线从由某一代开始会由下降转为上升，表明此时开始过拟合了，解决办法就是使用 EarlyStopping，中断训练保留最优化参数；

5、综合各种调整和优化发现 Adam 优化器的表现是最好的，不过为了试着观察学习率对数据拟合的影响项目中也用了 RMSProp；

6、对于个人开发者来看，机器真是个大问题，显存不够用，内存不够用需要虚拟内存帮忙，25000 数据量外加 3 倍数据增强的简单模型跑一次就耗时一个晚上，深度学习一定是个很烧钱的工作；

7、在 Mentor 的建议下，尝试使用迁移学习，训练的速度的确得到了提升，训练指标也得到很大提升，不过由于迁移学习很吃显存，项目中只使用了 25000 个原始训练数据集，在迁移学习的基础上应该还可以加上数据增强等优化。

5.3 后续改进

受限于手头上实验设备的计算能力，项目中训练次数均被限制在 10 次以内，以节省训练时间，后续可以将训练模型放到更先进的计算设备中训练。

项目中用的是个较为简单的类 VGG13 模型，后续可以尝试使用更先进的模型：Inception V3、Xception、ResNet 等。

项目中采用 RMSProp 和 Adam 优化器，后续可以尝试使用 Nadam 和其他的优化器。

除了了 Kaggle 中提供的训练集及测试集数据，后续可以尝试加入更多的训练数据，比如这个：http://www.robots.ox.ac.uk/~vgg/data/pets/，The Oxford-IIIT Pet Dataset。

参考文献

[0] ImageNet Classification with Deep Convolutional Neural Networks. NIPS2012.

[1] https://en.wikipedia.org/wiki/Artificial_neural_network

[2] https://en.wikipedia.org/wiki/Convolutional_neural_network

[3] convolutional-neural-networks-with-keras

[4] https://www.coursera.org/learn/convolutional-neural-networks

[5] Rethinking the Inception Architecture for Computer Vision

[6] Deep Residual Learning for Image Recognition

[7] Xception: Deep Learning with Depthwise Separable Convolutions

[8] https://www.zhihu.com/question/39022858

[9] Very Deep Convolutional Networks for Large-Scale Image Recognition

[10] Dropout: A Simple Way to Prevent Neural Networks from Overfitting

[11] Adaptive Subgradient Methods for Online Learning and Stochastic Optimization

[12] rmsprop: Divide the gradient by a running average of its recent magnitude

[13] Adam - A Method for Stochastic Optimization