机器学习学位纳米学位

猫狗大战毕业项目

黄有维

2018 年 4 月 26 日

I. 问题的定义

项目背景

猫狗大战项目是基于图片二分类的问题， 1 分类为狗，0 分类为猫。主要涉及机器学习的相关领域，包括计算机视觉的图像处理、卷积神经网络等。

本项目就是利用 kaggle 提供的数据[1]，来进行深度学习算法的建模训练和验证。随着机器学习领域的快速发展，现在很多大公司都选择在使用深度学习，比如 facebook 的智能标记、亚马逊的推荐算法等。

图 1 各大公司

问题描述

猫狗大战项目实质就是图像的二分类问题，最后的结果不是猫就是狗。

这里的目的就是要让机器能够对输入的图片做出分类。对于我们人类来说，我们能够非常容易分辨出图像的特征从而分辨出猫还是狗，对于计算机看来，眼中的数据只是 0 和 1。因此，在此要解决的问题就是让计算机能够准确地对图片做出 0 和 1 的判断，1 代表狗，0 代表猫，最后输出狗的概率是多少。

对于该问题，可以利用数据训练分类模型，进行分类判断。最后可以利用测试集对分类模型进行评估，得出猫狗的分类概率。

图 2 人类看到的

图 3 计算机看到的

评价指标

本项目评估指标是预测概率值的$loss\ log$，

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

其中，n为测试集的图片数量，$\hat{y}_i$为预测图片为狗的概率值，y_i为 1 则图片为狗，y_i为 0 则图片为猫，log()是以e为底的自然对数。$loss\ log$值越小越好。

II. 分析

数据的探索

所使用的数据来自 kaggle 的猫狗大战数据集，分为训练集 train 和测试集 test。训练集总共有 25000 张图片，以 type.num.jpg 的格式命名，都做了分类，狗和猫各占一半，12500 张。测试集总共有 12500 张图片，统一从 1~12500 来命名，没有做猫狗分类。

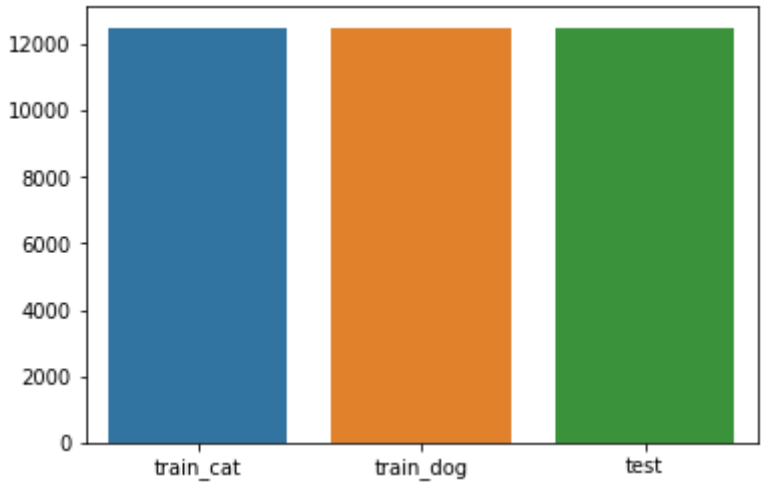
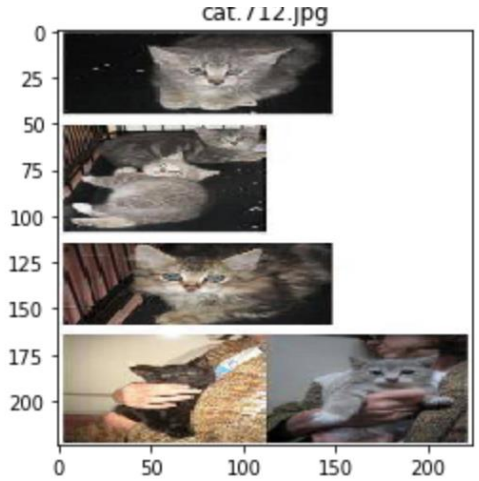


图 4 kaggle 数据集中训练集和测试集数量柱形图

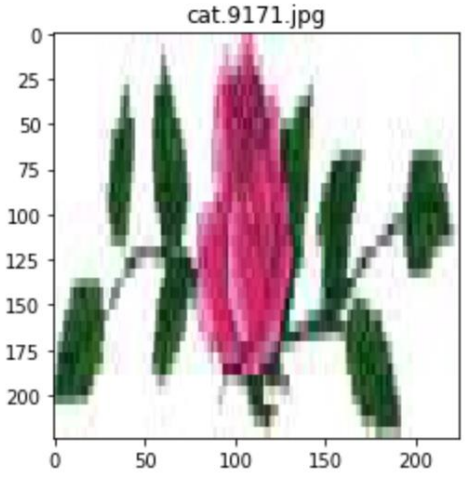
另外，通过利用现有的预训练模型，比如 ResNet50[2]，发现了一些异常值。参照 keras 文档，ResNet50 的 top1 和 top5 分别为 0.759 和 0.929，并且根据自己做实验得知，top 的取值越大，准确率会提高，我这次使用的 top 值为 20。此外，使用单一模型误差还是比较大，如果利用多个模型一起进行多次的分类，准确率应该会更高，从而更能准确挑出异常图片。我这次没有用多个模型强强联合对训练集进行分类，因为使用单个模型得到的异常值率就很低了。使用 ResNet50 模型对训练集 25000 张图片进行分类，最后刚好得到 100 张异常值图片，占比 0.4%，通过人工查看，虽然有分类错的，但是还是把异常值图片分离出来了。因为异常值占比比较小，所以本次项目不做异常值处理。

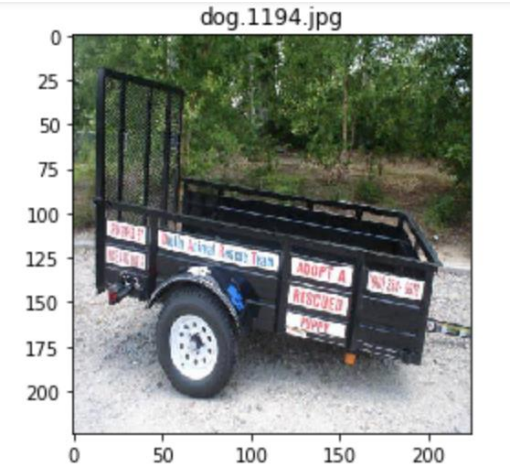
ResNet50 模型分离出来的异常值，举例如下：

1．图片中有多个猫，可能无法辨认

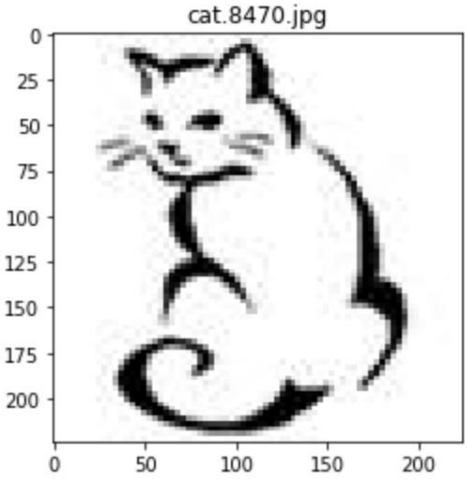


2. 图中没有猫或狗

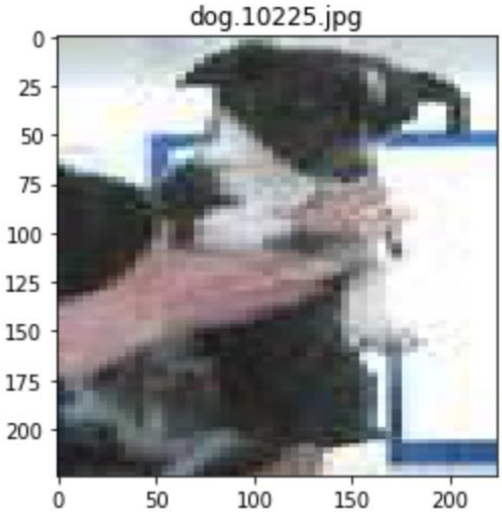




3. 卡通的形象的猫



4. 可能图片比较模糊，辨别不出是狗



这些图片的大小都是不一样的，所以在送入模型之前需要对图像做大小处理，如后面使用的预训练模型为 VGG16[3]，对输入图片(224x224x3)平整化，然后送进模型。另外，为了防止过拟合，需要从数据中选取验证集。并根据验证集的 loss 来调参，验证集的 loss 可以判断这个参数是否合理。我们可以从训练集中选取验证集，但要注意的是我们是把原训练集分成新的训练集和测试集，新的训练集不能与验证集有重合。如在 keras[4]中设置验证集的方法：可以自己建立验证集，然后在模型训练（fit 函数）时，传入 validation_data 中。

探索性可视化

从数据探索知道，训练集都已 cat 和 dog 命名做了分类，而且图片丰富多样，只有个别的图片质量会比较差。但最重要是需要把图片统一大小为所选模型的要求输入大小，如 VGG16 的大小就是 224x244。如图：

图 5 训练集的图片

图 6 测试集的图片

算法和技术

此项目使用的算法主要是深度卷积神经网络。

卷积神经网络（Convolutional Neural Network, CNN）[5]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。它由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。

典型的卷积网络，由卷积层、池化层、全连接层组成。其中卷积层与池化层配合，组成多个卷积组，逐层提取特征，最终通过若干个全连接层完成分类。对于卷积层完成的操作，可以认为是受局部感受野概念的启发，而池化层，主要是为了降低数据维度。综合起来说，CNN 通过卷积来模拟特征区分，并且通过卷积的权值共享及池化，来降低网络参数的数量级，最后通过传统神经网络完成分类等任务。

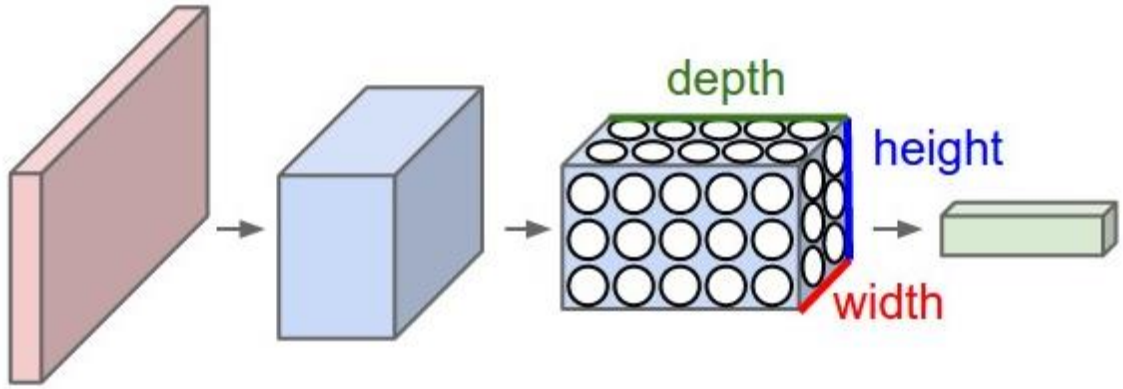


图 7 卷积神经网络 1

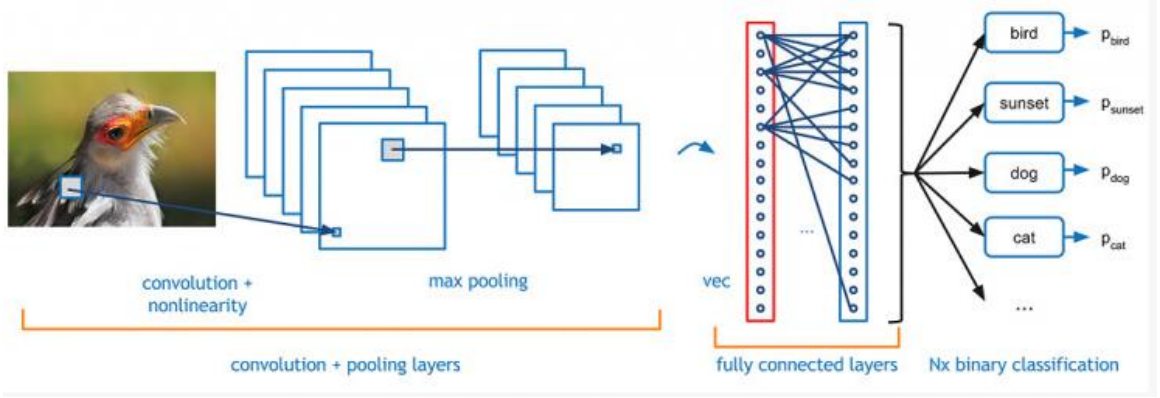


图 8 卷积神经网络 2

卷积神经网络包含以下几个核心部分：

卷积层

卷积神经网路中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法优化得到的。卷积运算的目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。

下面用一个简单的例子来讲述如何计算一个卷积。

假设有一个 5*5 的 image 图像，使用一个 3*3 的 filter 卷积核进行卷积，想得到一个 3*3 的 Feature Map 特征图，并设定 stride 步进为 1，如下所示：

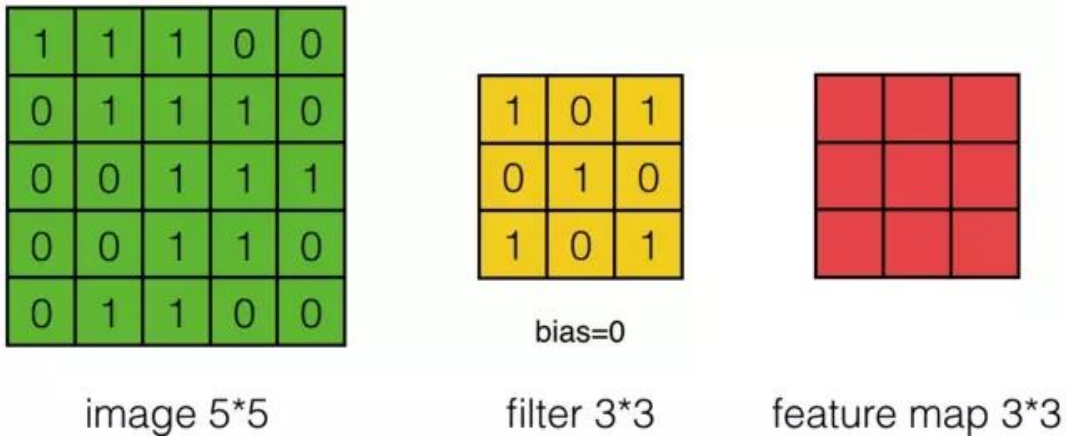


图 9 卷积核卷积

如图，输入数据为

[[1,1,1,0,0],[0,1,1,1,0],[0,0,1,1,1],[0,0,1,1,0],[0,1,1,0,0]],

卷积核的数据为[[1,0,1],[0,1,0],[1,0,1]],通过卷积核对输入图像进行卷积，则得到的特征图元素。如特征图的左上角元素为

1*1+1*0+1*1+0*0+1*1+0*0+0*1+0*0+1*1=4,其它元素依次计算，最后特征图的数据为[[4,3,4],[2,4,3],[2,3,4]]。

卷积层是卷积核在上一级输入层上通过逐一滑动窗口计算而得，卷积核中的每一个参数都相当于传统神经网络中的权值参数，与对应的局部像素相连接，将卷积核的各个参数与对应的局部像素值相乘之和，（通常还要再加上一个偏置参数），得到卷积层上的结果。

实际上，卷积神经网络在训练过程中会自己学习这些过滤器的值（尽管在训练过程之前我们仍需要指定诸如过滤器数目、大小，网络框架等参数）。我们拥有的过滤器数目越多，提取的图像特征就越多，我们的网络在识别新图像时效果就会越好。

而卷积特征的大小由我们在执行卷积步骤之前需要决定的三个参数控制：

深度

深度对应于我们用于卷积运算的过滤器数量。

步幅

步幅是我们在输入矩阵上移动一次过滤器矩阵的像素数量。当步幅为 1 时，我们一次将过滤器移动 1 个像素。当步幅为 2 时，过滤器每次移动 2 个像素。步幅越大，生成的特征映射越小。

零填充

有时，将输入矩阵边界用零来填充会很方便，这样我们可以将过滤器应用于输入图像矩阵的边界元素。零填充一个很好的特性是它允许我们控制特征映射的大小。添加零填充也称为宽卷积，而不使用零填充是为窄卷积。

池化层

池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种形式的降采样。我们之所以使用卷积后的特征，是因为图像具有“静态型”的属性，也就意味着在一个图像区域的特征极有可能在另一个区域同样适用。所以，当我们描述一个大的图像的时候就可以对不同位置的特征进行聚合统计（例如：可以计算图像一个区域上的某个特定特征的平均值或最大值）这种统计方式不仅可以降低维度，还不容易过拟合。这种聚合统计的操作就称之为池化。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。除了最大池化之外，池化层也可以使用其它池化函数，例如“平均池化”甚至“L2-范数池化”等。过去，平均池化的使用曾经较为广泛，但是最近由于最大池化在实践中的表现更好，平均池化已经不太常用。

池化层总的来说主要有两个作用。

1. 保留主要特征的同时减少参数（降低维度）和计算量，防止过拟合。
   如果我们直接拿从卷积层提取得到的特征去训练分类器，这样的计算成本是非常高的。比如一个 50x50 像素的图像，假定我们已经学习到了 200 个定义在 8x8 输入上的特征，每一个特征和和图像卷积都会得到一个(50-8+1)*(50-8+1)=1849 维的卷积特征，所以每个样例就会得到 200x1849=369800 维的卷积特征向量。
2. 不变性，这种不变性包括平移、旋转和缩放。这使网络鲁棒性增强，有一定抗扰动的作用
   假设要识别 16x16 图片中的数字“1”。图中的“1”可能偏左也可能偏右，如图 a 偏左，图 b 偏右，它们相差一个水平单位，经过最大池化后它们都变成相同的 8x8 特征矩阵，捕获到了主要特征，又降到了 8x8，这具有平移不变性。

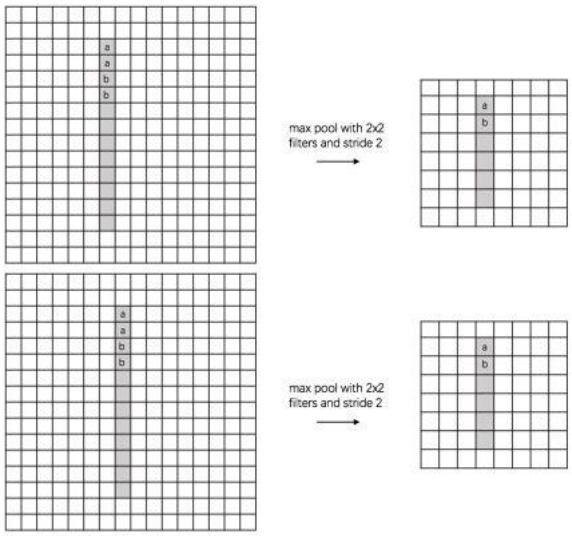


图 10 平移不变性

假设要识别图片中的中文“一”，如图 a 相对 x 轴倾斜，图 b 平行 x 轴，它们相当于做了旋转，经过多次最大池化后具有相同特征。

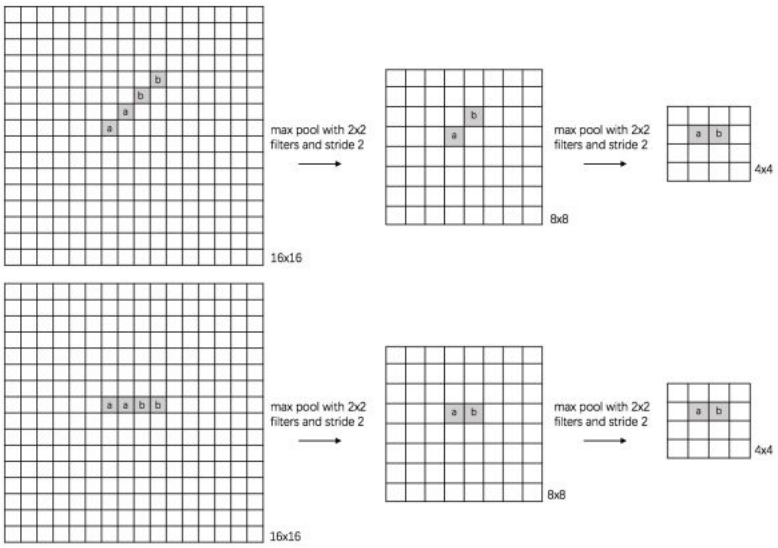


图 11 旋转不变性

假设要识别图片中的“0”，如图 b 相对图 a 缩小了，经过多次最大池化后也具有相同的特征。

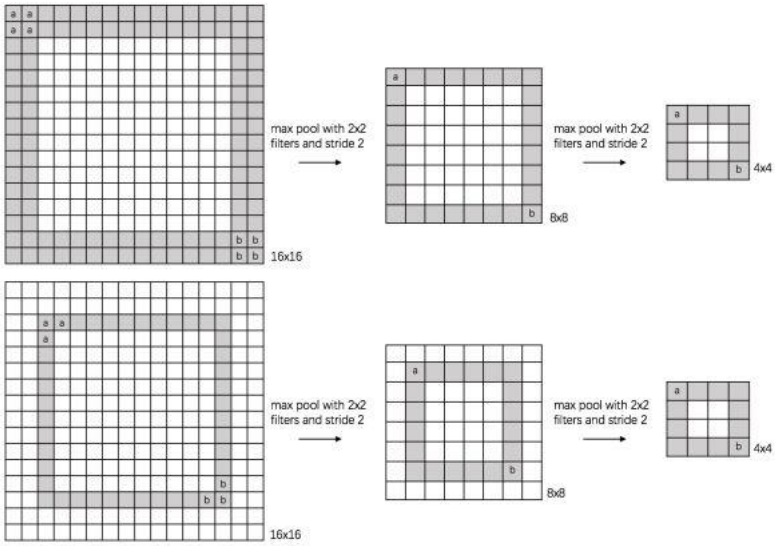


图 12 缩放不变性

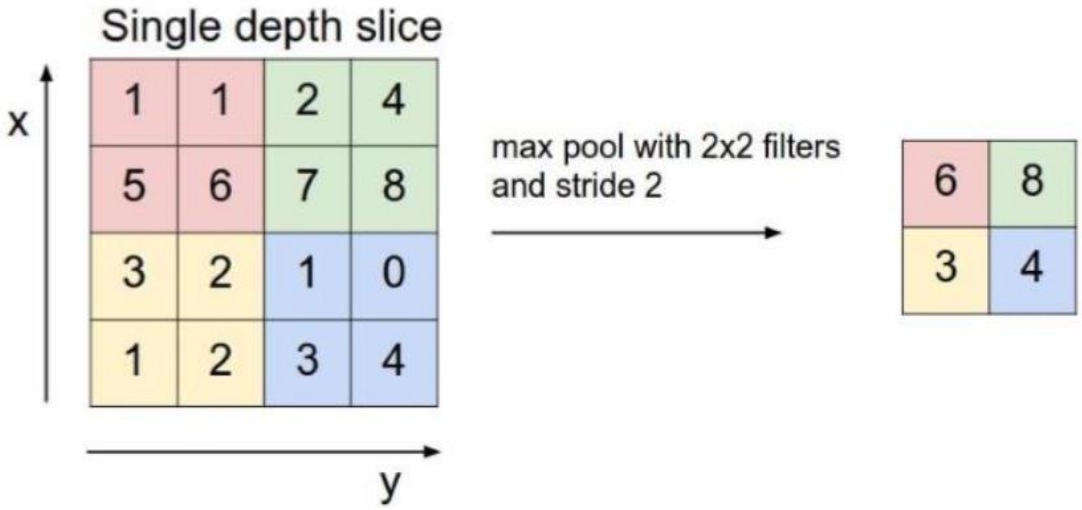


图 13 最大池化

激活函数

激活函数是用来加入非线性因素的，因为线性模型的表达力不够。在神经网络中，对于图像，我们主要采用了卷积的方式来处理，也就是对每个像素点赋予一个权值，这个操作显然就是线性的。但是对于我们样本来说，不一定是线性可分的，为了解决这个问题，我们可以进行线性变化，或者我们引入非线性因素，解决线性模型所不能解决的问题。

本项目使用迁移学习技术，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 0 学习。

本项目选择了 VGG16 作为预训练模型，因为该模型已预先在 ImageNet 数据集上进行训练，而 ImageNet 数据集已经包含了 1000 个类中的几个”猫”类和”狗”类，所以这个模型已经学到了与我们的分类问题相关的特征，作为本项目的猫狗分类效果应该会不错。

对于 VGG16 模型，首先去掉全连接层的网络，得到数据集的 bottleneck feature，然后自己设计几层全连接层，对其进行训练，最后再微调得到最终模型。VGG16 的基本架构如下：

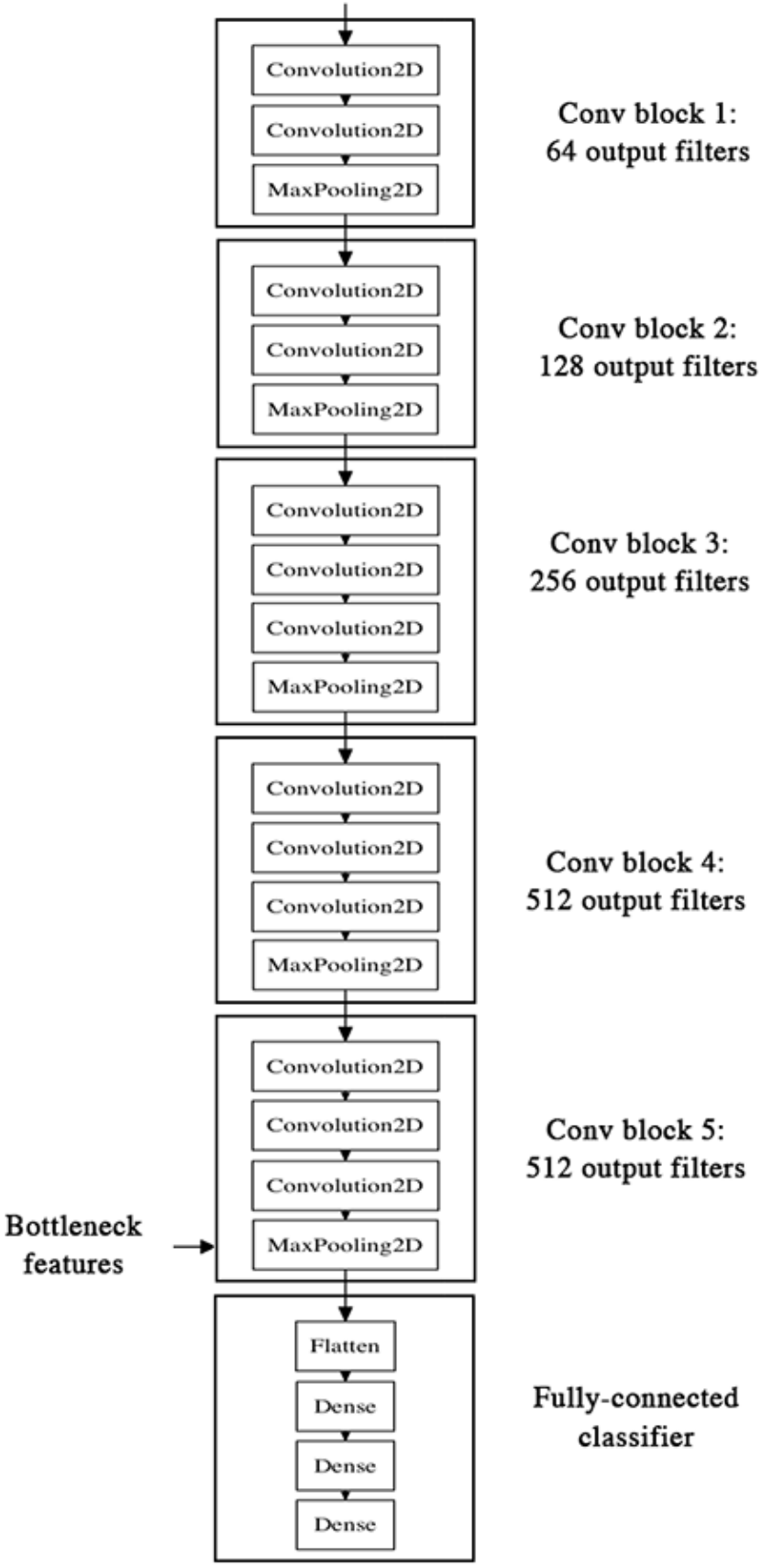


图 14 VGG16 模型结构图

如图，VGG16 共 5 个卷积区，每一个卷积区最后是一个最大池化层，总共有 13 层的卷积层。

当我们在使用预训练模型时，往往把自身的全连接层去掉，换成适合我们项目的全连接层，这里，就直接加入了 dropout 层然后分类。

基准模型

本项目是对猫狗的二分类，得出的结果值上传到 kaggle 官网进行评估，得出分数。最后预测值的基准的最低要求是 kaggle Public Leaderboard 前 10%，也就是 logloss 要低于 0.06127。

III. 方法

数据预处理

先把训练集数据的猫狗作好分类标签并存到数组中，如，train_data 为图片数据，train_labels 为图片对应的分类，猫为 0，狗为 1。然后把训练集 train 利用 train_test_split 拆分为了训练集和验证集，验证集占 20%大小。

由于数据集的图片大小不一，所以，必须要根据所选模型来把图片重新平整大小。如本项目选用的 VGG16 模型图片输入大小为 224x224，使用了模型自带的 preprocess_input 函数做输入数据的预处理，这个函数的作用是对样本执行逐样本均值消减的归一化，即在每个维度上减去样本的均值。

常用的数据预处理包括数据归一化。

数据归一化的常用方法有：

1. 简单缩放

   简单缩放方法是通过对数据各个维度上的值进行重新调节，使得数据整体上分布在[0,1]或[-1,1]区间。

2. 逐样本均值消减

   逐样本均值消减，也称为移除直流分量，具体操作是把每个样本都减去所有样本的统计平均值，这种归一化方法在图像领域常见。

3. 特征标准化

   特征标准化指的是（独立地）使得数据的每一个维度具有零均值和单位方差，具体操作是首先计算每一个维度上数据的均值（使用全体数据计算），之后在每一个维度上都减去该均值。下一步便是在数据的每一维度上除以该维度上数据的标准差。

做好以上预处理后，准备把数据送入模型进行训练。

执行过程

1.基本迁移训练

VGG16 模型是训练好的模型，可以直接使用，首先是去掉全连接层，加上自己的全连接层，由于该分类任务不是很复杂，简单的二分类，直接加入了 Dropout 和一层全连接层。

为了防止容易过拟合，对 VGG16 模型卷积层的输出也做了 GlobalAveragePooling2D 全局平均池化，同时又对所有卷积层进行冻结。Dropout 系数使用 0.25，优化器选用 adadelta[6]。选用 adadelta 优化器，是因为它是自适应学习率调整的，不需要设置一个默认的学习率。

其实 adadelta 算法是 adagrad[7]算法的改进版，它主要解决了 adagrad 算法单调递减学习率的问题。通过约束历史梯度累加来替代累加所有历史梯度平方。这里通过在历史梯度上添加衰减因子，并通过迭代的方式来对当前的梯度进行计算，最终距离较远的梯度对当前的影响较小，而距离当前时刻较近的梯度对当前梯度的计算影响较大。

adagrad 算法的计算公式为：

$$\Delta x_t = -\frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2 + \epsilon}} g_t$$

其中，η为原始学习率，gt 为 x 在 t 时刻的梯度，ϵ 是一个比较小的数，用来保证分母非 0。

学习率是单调递减的，我们可以只使用 adagrad 的分母中的累计项离当前时间点比较近的项，如下式：

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

$$\Delta x_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

这里ρ是衰减系数，通过这个衰减系数，我们令每一个时刻的 g_t 随之时间按照ρ指数衰减，这样就相当于我们仅使用离当前时刻比较近的 g_t 信息，从而使得还很长时间之后，参数仍然可以得到更新。其中 E 代表求期望。

在此处 adadelta 其实还是依赖于全局学习率的，但是作者做了一定处理，经过近似牛顿迭代法之后：

$$\Delta x_t = -\frac{\sqrt{E[\Delta x^2]_{t-1}}}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

可以看到，如此一来 adagrad 中分子部分需要人工设置的初始学习率也消失了。

以上便是 adagrad 的最终更新参数公式。

最终得到的模型如下图：

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
lambda_1 (Lambda)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d_1 (	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513

Total params: 14,715,201
Trainable params: 513
Non-trainable params: 14,714,688

图 15 模型输入输出参数结构

使用 fit 得到的训练参数如下：

图 16 VGG16 的训练和验证 accuracy 曲线

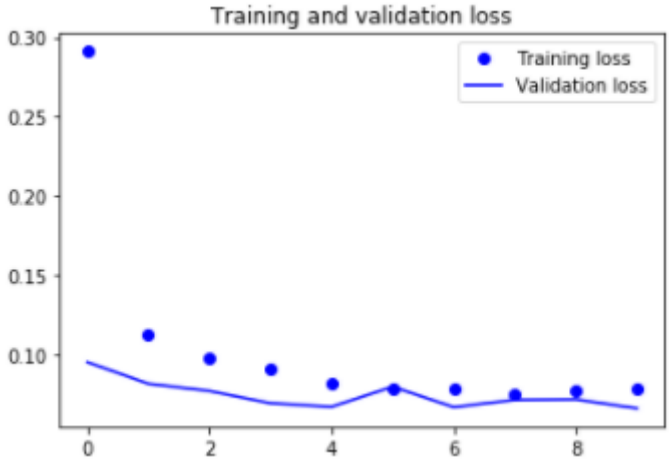


图 17 VGG16 的训练和验证 loss 曲线

把结果存储为 csv 后上传到 kaggle 官网进行评估，最终得分为 0.07674。

2. 进行 finetune

为了提高分数，对 VGG16 模型的 block5 卷积层解冻加入训练，此时需要使用很低的学习率，这是为了保证更新的幅度保持在较低的程度，要不然会破坏原有的训练权重，这里使用优化函数为 SGD(lr=1e-4, momentum=0.9)。

SGD 随机梯度下降，是随机选取一个点做梯度下降，而不是遍历所有样本后进行参数迭代。因为梯度下降法的代价函数计算需要遍历所有样本，而且是每次迭代都要遍历，直至达到局部最优解，在样本量庞大时就显得收敛速度比较慢了，计算量非常庞大。随机梯度下降仅以当前样本点进行最小值求解，通常无法达到真正局部最优解，但可以比较接近。

对于训练数据集，我们首先将其分成 n 个 batch，每个 batch 包含 m 个样本。我们每次更新都利用一个 batch 的数据，而非整个训练集。即：

$$x_{t+1} = x_t + \Delta x_t$$

$$\Delta x_t = -\eta g_t$$

其中，η为学习率，g_t 为 x 在 t 时刻的梯度。

```
input_1 False
lambda_1 False
block1_conv1 False
block1_conv2 False
block1_pool False
block2_conv1 False
block2_conv2 False
block2_pool False
block3_conv1 False
block3_conv2 False
block3_conv3 False
block3_pool False
block4_conv1 False
block4_conv2 False
block4_conv3 False
block4_pool False
block5_conv1 True
block5_conv2 True
block5_conv3 True
block5_pool True
global_average_pooling2d_1 True
dropout_1 True
dense_1 True
```

图 18 解冻 block5 层

本次的训练图如下：

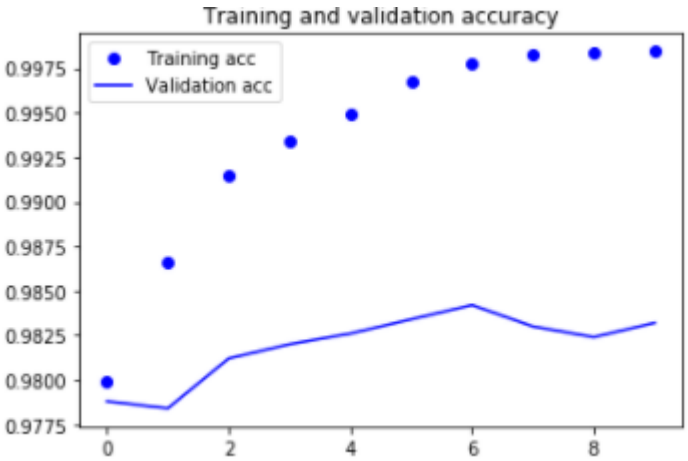


图 19 finetune 后 acc

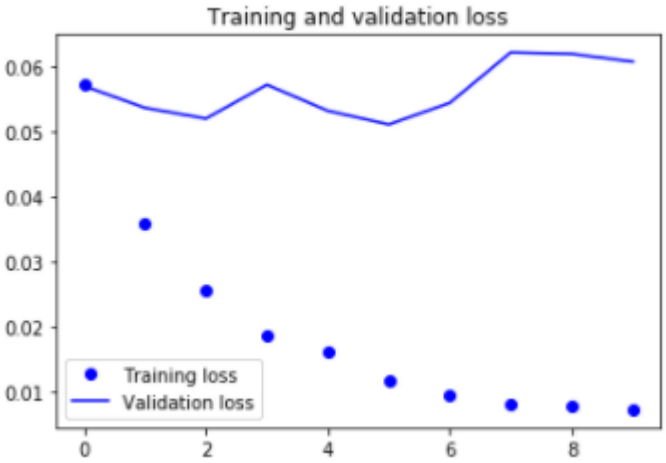


图 20 finetune 后 loss

从图上看出，虽然 val acc 有上升，但 val loss 却是震荡、上升趋势的，这应该有过拟合嫌疑了。同时，为了评估这个模型，我加载了测试集，而这些测试集是不需要做特别处理的，然后把测试集数据喂到该模型中进行预测。这里也运用技巧[8]，把预测值做了区间剪切限制在[0.005,0.995]，因为这可以避免在我们得到错误结果时得到严重的惩罚，即可以减少惩罚。最后把预测结果传到 kaggle 官网评估，结果为 0.05844。

另外为了得到更好评分，我尝试加大了 batch_size 参数为 30 后，val loss 相对平滑一些，但是上升趋势也比较明显了。另外，把 dropout 系数提高为 0.5，学习率改为更小 0.00001,似乎效果也不太明显，可能哪里出错了。但是，我没有接着继续调试这个单模型，一是因为进行训练时间比较长，二是具体也不知道继续调整哪些参数了，所以，我直接对该模型进行继续完善。

完善

可能使用其他的模型比如 InceptionResNetV2[9]，会比 VGG16 分数要高。但是这里考虑参考 keras 官方例子和网上大神文章的思路，进行两个模型融合。先把特征向量以 h5 文件格式导出到本地，然后后续再导入特征向量进行训练，而不是直接加入自己的全连接层进行训练，这样会大大节省资源和时间，不用重复做过多的计算。

本次项目最终使用图片生成器 ImageDataGenerator 来提取图片，根据生成器的定义，需要新建一个目录，目录包含各种分类，默认会按照字母排序。这里，建立了训练集文件夹 mytest，该文件夹下建立 cat 和 dog 文件夹，然后从 kaggle 的 train 数据集中分别把 cat 和 dog 图片拷贝到对应的 cat 和 dog 文件夹中。建立 mytest 文件夹，该文件夹下建立 test 文件夹，然后从 kaggle 的 test 数据中的数据拷贝到该 test 文件夹下。因为客观原因，本次目标是最低要求，所以没有做图片的特殊处理，ImageDataGenerator 使用默认值即可。

这里利用 VGG16 和 InceptionResNetV2 模型对数据做了特征向量的提取，使用了 aws 的 p3.2xlarge 云服务器进行计算，VGG16 仅花了将近 400 多秒，特征量文件约 74MB，InceptionResNetV2 仅花将近 1100 多秒，特征量约 219MB。
首先需要把数据从 h5 文件中提取出来，分为训练集和测试集，和图片生成器自动生成的训练集分类标签。我们需要把这些特征向量合成一条特征向量，然后对训练数据进行 shuffle 打乱，测试集保持不变。此时，已把输入数据准备好。最后加入自己的全连接层，优化器还是使用 SGD(lr=1e-4, momentum=0.9)，最后开始训练。
训练曲线如下：

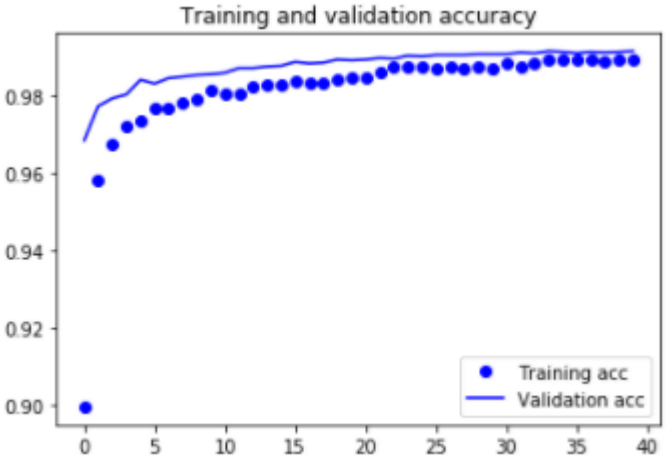


图 21 融合模型的 acc

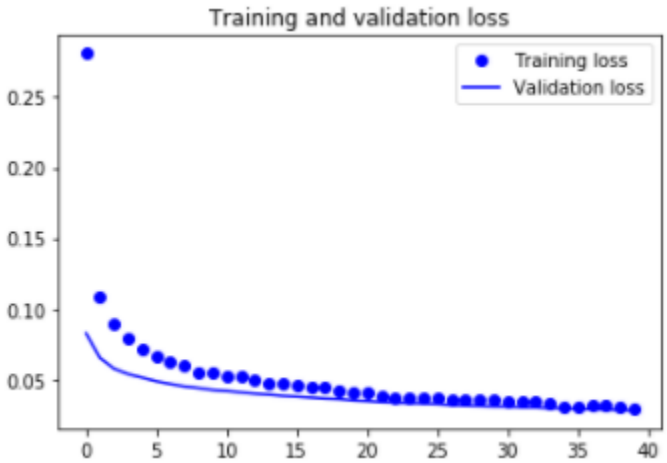


图 22 融合模型的 loss

Val loss 和 val acc 趋于平稳，感觉已经趋于稳定了，可调空间不太大。
最后得到的预测结果，上传到 kaggle 进行评估，分数为：0. 04783。

IV. 结果

模型的评价与验证

就单模型而言，VGG16 模型的深度只有 23，得到的分数也接近基准值。InceptionResNetV2 模型的深度多达 572，是非常复杂的网络，单模型的分数为 0.05515，说明网络的深度，可以提高模型表现。

对于多特征融合模型，综合了各个模型的优异表现，最后得到更低的 logloss。最终的模型，就是加入了自己的全连接层，即一个 Dropout 层和一个 Dense 层，激活函数使用 sigmoid，优化器使用了 SGD，最终 logloss 值达到了要求。
另外，如果还做图像增强，模型拟合应该应该更好。

合理性分析

利用最终模型达到了目标值。从测试集中随机抽取 10 张图片进行预测，预测结果不错。所以我认为，这个模型是合理的。预测结果如下：

图 23 预测结果可视化

V. 项目结论

结果可视化

这次项目，我记录了对 VGG16、InceptionResNetV2 和融合后的模型 tensorboard 日志，可以很直观的看到 acc 和 loss 的变化曲线，以下是混合模型的训练曲线：

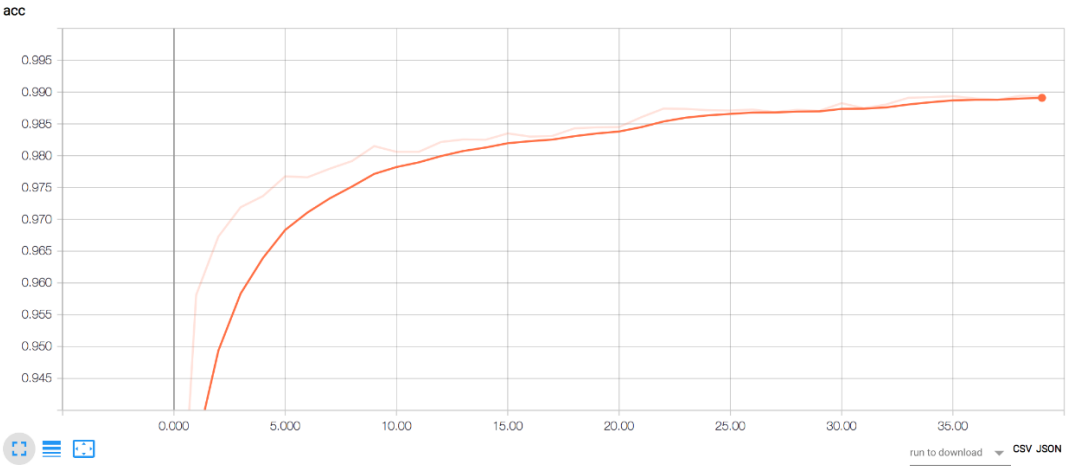


图 24 混合模型的 acc 训练曲线

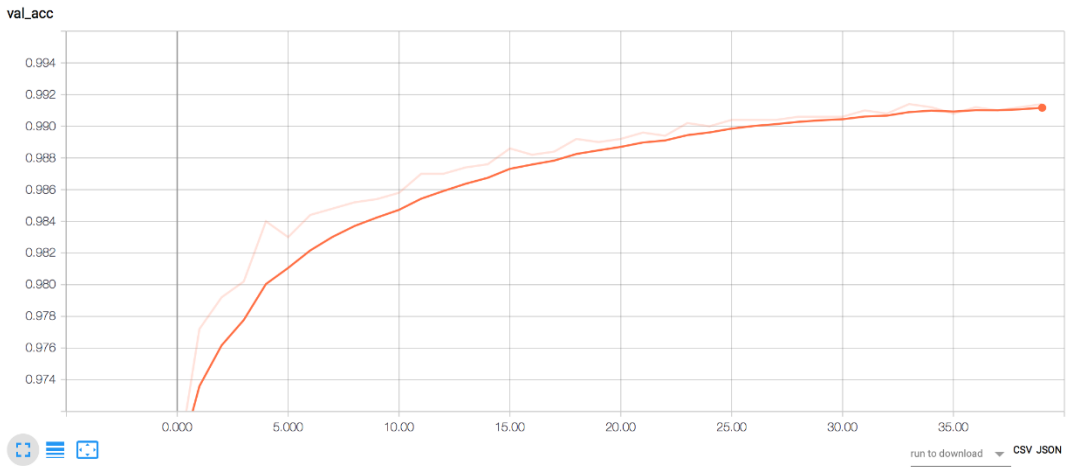


图 25 混合模型的 val acc 曲线

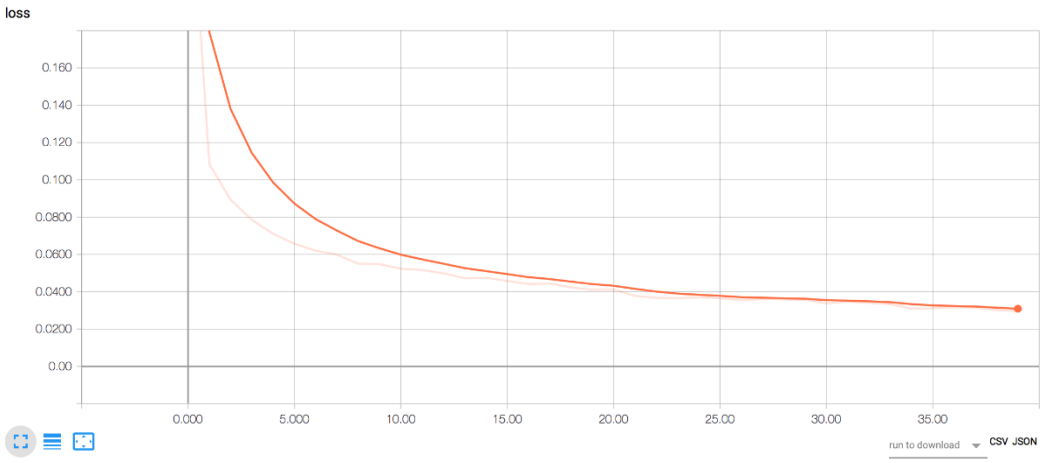


图 26 混合模型的 loss 曲线

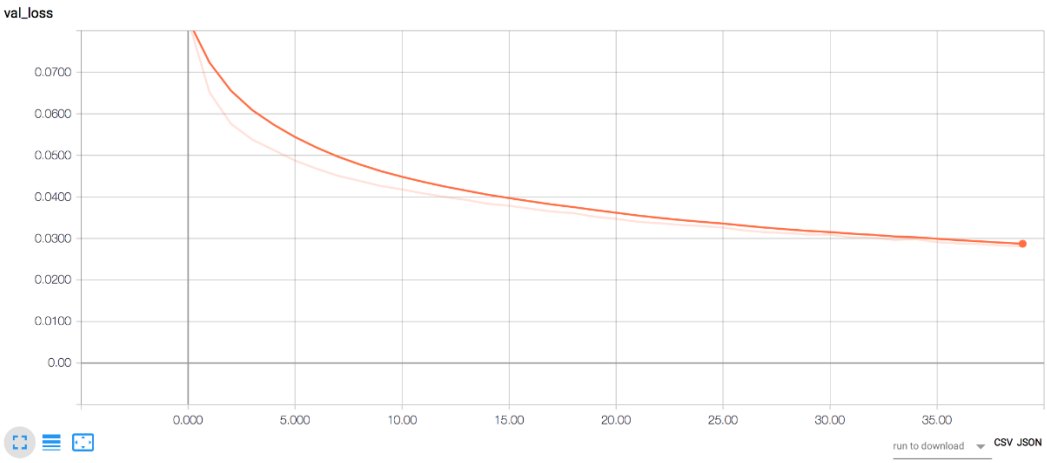


图 27 混合模型的 val loss 曲线

上图，横轴表示 epochs 的数量，纵轴为损失值或准确率。Loss 是训练集损失值，acc 是训练集准确率，vall acc 是验证集准确值，val loss 是验证集损失值。通过验证集的损失值和准确值可以评估该模型是否是过拟合，还是最佳模型。

本次调试的 epochs 迭代数为 40，损失率降低到了 0.03 左右，但是实际在 kaggle 验证的 loss 为 0. 04783，说明有一点点过拟合了，应该还有调参的空间。
模型已经达到期望值了，所以，如果需要更高的准确率，需要对模型做出进一步的改进。

对项目的思考

这个项目主要尝试了使用迁移学习来完成，最开始是使用单模型 VGG16，并进行了 finetune，得到结果与期望值接近。后来改为特征融合模型来训练，效果更好，最终达到了基准值。其实本次的项目，只是很基本的训练，没有过多调参，可能基准值也只是个及格值，一般选用好的预训练模型，再微调一下基本可以达到。因为我目标是达到基准值，所以，没有做过多的微调就达到了基准值。

在项目中，最有趣应该可以看到 acc 和 val loss 按照预想的路线在走，比如随着 val loss 降低，acc 在提升。而恰恰比较困难的，也是对模型的微调使 acc 和 log loss 按照正确路线走，避免过拟合。

模型已经达到了期望，而且对进行一些简单调整就能在大多的通用场景 模型已经达到了期望，而且对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。
通过猫狗大战项目，可以更深一步了解深度卷积神经网络，为进阶学习打下良好基础。

需要作出的改进

可以再融合其它比较新的预训练模型进来训练，还有就是做数据提升，比如图像的缩放、翻转、平移等处理。还有比较重要的一部就是，要清理数据的异常值，这样更能提高模型的准确率。另外，还可以制作一个 app，使用本算法和技术，实现图像分类，是新技术进入到人们生活中，这是最好的实践。

参考文献

[1] Kaggle, https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, Submitted on 10 Dec 2015

[3] Karen, Simonyan, Andrew, Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition，2015

[4] Keras Doc, https://keras.io/

[5] Convolutional neural network, https://en.wikipedia.org/wiki/Convolutional_neural_network

[6] Matthew D. Zeiler,ADADELTA: An Adaptive Learning Rate Method,2012

[7] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159.

[8] Kaggle, https://www.kaggle.com/sbrugman/tricks-for-the-kaggle-leaderboard

[9] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi ,Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,2016

[10] Francois Chollet,Building powerful image classification models using very little data,2016

[11] 手把手教你如何在 Kaggle 猫狗大战冲到 Top2%, https://zhuanlan.zhihu.com/p/25978105