

1.定义

ML_Submission_P7_2

增添删改的部分为1.1, 2.3.2, 2.3.4中黑体字。其他部分黑体字为上一次修改内容。

1.1 项目概述

关于计算机视觉的研究始于上世纪60年代，L.R.Roberts发表了被广泛认为是计算机视觉史上第一部关于computer vision的专业论文，它探讨了如何重新构建世界中的各种图案，将其抽象成各种简单地图形，以达到识别出该物体是什么的目的。

而如今关于计算机视觉，已经在各方各面有了发展，深度学习，非监督学习，神经网络，卷积神经网络，各种各样新的技术层出不穷。由新的技术中又催生出了许多新的实际应用，例如自动驾驶（2011年美国内达华州授权自动驾驶汽车），医疗图像识别（人工智能在CHINA杯神经影像识别中击败人类医生），面部识别（智能手机的面部解锁，天网系统）。那么其实在现在这种背景下，计算机对各类事物已经有了非常好的识别能力，人们的生活也因此得到了巨大的改变，那么如今最重要的，就是继续提升精度，提升性能

本项目为kaggle上一个非常有名非常经典的计算机视觉比赛——猫狗大战。这项比赛起始时间为2016年9月，结束为2017年3月，为期6个月。诸多高手过招，为计算机视觉领域提供了许许多多新的调参，模型的实践方法。

这一项目目的在于将12500张图片分为猫和狗，训练集为25000张，数据集中除了训练集中的图片文件名带有cat, dog字样外，没有其他任何的事先处理和分类，所以在对数据集怎么处理才能使我们将其作为数据集是需要考虑的问题。而最终的预测结果，我们将利用已经在IMAGENET上训练过的一些已有的模型进行预测并输出一个值，该值将以0到1之间[闭区间]的数字呈现，0为猫，1为狗，也就是说越接近0，图片中越可能是猫，反之为狗。

本项目将使用pytorch，在windows10企业版中使用rtx2070，cuda进行训练。

1.2 问题陈述

这是一个监督学习二分类的问题，我们需要设计一个模型，使其能够根据图片的标记进行拟合，达到将任意一张新的图片分为猫或狗的目的。

那我们需要解决的问题主要有三点，1.将训练图片分放到不同的文件夹，并正确加载数据，使其能够符合我们模型的输入要求。2.在众多模型当中，找到一个最合适的对该模型进行训练。3.导出测试集的结果，上传到kaggle以验证得分。

对于第一点，我们可以利用照片文件名里的cat, dog把文件分类到不同的文件夹里面，然后把这些图片文件进行一下转换，将其转换成向量的形式，因为本来计算机中图片通常就是RGB三个数值的向量构成的像素点构成的。由于RGB的数值最大为255，那么我们还需要对其进行归一化处理以避免产生过拟合或欠拟合问题。

第二点，我们可能就需要花些功夫了，这是这一项目的关键，也就是训练模型。我们使用的模型都将是带有卷积层的模型，卷积层会将图片的内容一层一层剥离开来，比如第一层识别线条，第二层识别线条组合，第三层识别矩形，圆形，第四层识别眼睛，耳朵等等。这些层叠加在一起就能使计算机得出一个结论。模型的训练就像教育小孩，你需要使其从错误中学习，设定一个loss函数，让它知道自己‘有多错’，‘错在哪’，还要设定一个改正器optimizer，帮助它从错误中以一定的学习速率进行改正；但训练模型又不像教育小孩，因为教育小孩无可重来，但模型的训练却需要模型结构，loss函数，优化器optimizer，等等各方面的组合才能从中得到最优解。所以解决方案就是多加尝试，并从中找出规律。

第三点比较简单，需要我们用训练好的模型，对测试集进行测试，然后将其提交并验证我们的想法即可。那么kaggle有一个排行榜，榜上总有1314名好汉，我们需要至少进入前10%。也就是说我们的预测准确性应该比榜上90%的人都要好。

1.3 评价指标

评价指标只有一个，那就是kaggle中的Logloss，不过这个要提交之后才能看到。

提交之前有两个指标作为参考，一是BCEloss，也就是Binary Cross Entropy Loss，那么这一函数将会告诉模型离它的目标狗——数字1，猫——数字0，差距有多大，这一函数与最终kaggle的评价函数logloss相似。

kaggle最终评分的LogLoss= $-1/n \sum [y_i \log(y_i) + (1-y_i) \log(1-y_i)]$ ，是求n张图片误差平均值。

n: 测试集中的图像数

y_i: 图像是狗的预测概率

y_i: 图像标记，狗为1，猫为0

log(): 自然对数

而与Logloss不同在于BCELoss= $-w_n [y_i \log(y_i) + (1-y_i) \log(1-y_i)]$ ，是求单张图片误差乘以权重w_n，w_n默认为None。 $[y_i \log(y_i) + (1-y_i) \log(1-y_i)]$ 就是两者相似之处。

二是accuracy，准确率，这个指标意思是判断正确的百分比，我们的激活函数会是sigmoid，所以我们的输出结果为[0,1]区间的数字，大于0.5，我们就将其判断为狗，小于0.5，我们就将其判断为猫。

那么作为最终的评价指标，kaggle中的logloss，我们需要在kaggle的排行榜上达到前10%，也就是logloss得分起码要低于0.061才可以。

2.分析

2.1 数据的探索

数据集为：猫与狗的图片，分为训练集和测试集。

与问题的关系及使用：问题就是从图片中分辨是猫还是狗，所以数据也是猫和狗的图片，训练集用于训练，测试集用于测试。

数据来源：<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data>

(<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data>)，图片具有标记1或者0，1就是狗，0就是猫。

使用是否合理：猫狗的图片一部分用于训练，一部分用于测试，能够使模型得到学习，也可以测试模型好坏，使用它们是合适的。

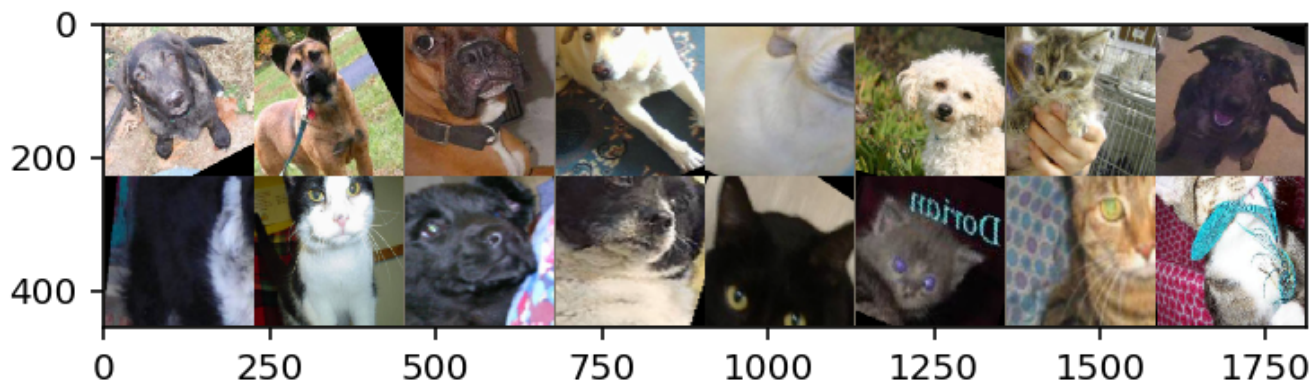
训练集图片数量为25000，测试集图片数量为12500。

训练集当中所有的文件名都以'猫狗.数字.jpg'的形式构成，猫狗即可以看作是kaggle预先对训练集做出的标记，猫为cat，狗为dog，我们可以利用这个特性将其分开，由于这是kaggle给我们的训练集，所以我们不考虑其中的离群值或者坏掉的没法用的图片，默认全部图片都是高宽大于等于224像素的标记正确的图片，因为大多数迁移学习的模型都需要图片高宽至少为224。

测试集中所有文件名都以'数字.jpg'的形式构成，意味着没有标记，猫和狗混在一起，图片的性质同上，可以供大多数迁移学习的模型使用。还有一个需要注意的点是，我所使用的pytorch中的imagefolder的功能，并不会将训练集中的文件按照文件名放入datasets中，而是以['1', '10', '100', '1000', '10000', '10001', '10002', '10003', '10004', '10005', '10006', '10007', '10008', '10009', '1001', '10010', '10011', '10012', '10013', '10014']这样的排序，所以将预测结果和具体文件名对应起来，是一个需要处理的问题。

然后我们将数据集中的狗和猫的图片分好类后，可以随机展示一些图片，以便检查有没有分类错误的，遮挡等等的问题。

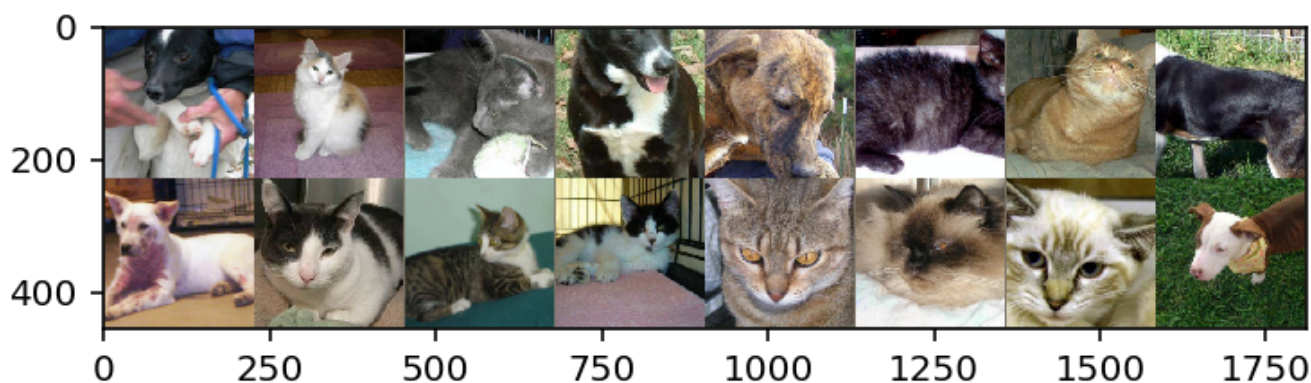
标记为：['dog', 'dog', 'dog', 'dog', 'dog', 'dog', 'cat', 'dog', 'dog', 'cat', 'dog', 'dog', 'cat', 'cat', 'cat', 'cat']
可以看到没有太大问题，只是有些图片仅展示了身体的一部分。对于这个问题，可以先尝试训练，之后再看看这些只显示身体一部分，或者有遮挡的图片对结果是否有较大影响，如果这些看不太清出的图片对结果有较大影



再看看测试集中的图片。

标记为[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

但真实的文件名为['1', '10', '100', '1000', '10000', '10001', '10002', '10003', '10004', '10005', '10006', '10007', '10008', '10009', '1001', '10010', '10011', '10012', '10013', '10014']



接着随机打印一些图片的高宽，看图片是否适用于迁移学习中的预训练模型。

(336, 447, 3) (346, 499, 3) (189, 249, 3) (408, 499, 3) (342, 500, 3) 可以看到大多数图片宽高都大于224，不存在图片过小的问题。

2.3 算法和技术

2.3.1 关于层的介绍

总的来说，在计算机视觉领域我们现在主要用到的算法技术是卷积神经网络，这一网络够结合卷积层，池化层，relu层，全连接层，dropout层等等，对图像进行分类。

卷积层是用来提取特征的，比如图片中小到线条，大到狗狗的脸，人的脸，都能够一层一层提取出来。

这一层的计算方式是，给出一个一定大小的窗口，称为卷积窗，比如 3×3 大小，用该窗口以一定步长遍历整个输入矩阵，并且与矩阵做点积返回一个新的数值，所有新的数值组成一个新的矩阵，也就是卷积层。

relu的计算方式是将0以下的数值全都回归0，0以上的数值不变，起到减小数值加快计算的作用。

池化层则是为了减少卷积层深度加深带来的维度上的变化。

池化层的计算方式和卷积层类似，不过不是利用一个窗口计算点积，而是使用一个窗口以一定步长遍历整个输入矩阵，并返回每次窗口内的平均值（或者最大值，可以根据需要设置为最大值还是均值或者其他类型的均值），返回的所有平均值（或者最大值），组成一个新的矩阵或者向量，也就是池化层。

BN(Batch Normalization)层的作用是使值规范化，改变值呈正态分布，使激活函数能更好的利用其进行激活。

该层的计算方式是在每个batch上将前一层的激活值重新规范化，使得其输出数据的均值接近0，其标准差接近1。

dropout层起到随机关闭某些节点，起到避免过拟合的作用，计算方式是随机的将某些节点的权重设置为0。

而全连接层则负责最终的分类，它将决定如何使用之前所有层传递来的信息，到底使用哪一些，并以此得到一个结果，这个结果可能是1个，也可能是多个，我们将可以按需要进行调整，并且可以在后面加入一个激活函数。

全连接层的计算方式是 $w \cdot x + b$ ，也就是以权重乘以输入，再加上偏置，返回一个新的值，由全部返回的值组成一个新的向量，也就是全连接层。

那么对于网络，我们有现成的已经训练过的模型，也就意味着不用对大多数层级作考虑，如vgg，resnet，densenet等等有一大批模型可用。所以我们将使用成熟的训练过的模型，然后将最后的全连接层进行替换即可。不过需要注意一点就是需要冻结最后的全连接层或者分类器之前的所有参数，否则可能会导致不必要的对于所有参数进行的更新，增加了时间成本。

2.3.2 关于模型结构

VGG使用的是较为简单的结构，（卷积层 + BN层 + RELU层）* n + 池化层 + 全连接层这样的形式，并且深度相对于Resnet和Densenet来说都比较浅，优点在于使用 3×3 的小窗口，可以有效避免在加深深度时造成参数过多的情况。

Resnet的优势在于它下面的层和上面的层是连接起来的，这种残差学习的方式解决了增加网络深度后的梯度消失的问题，因为一旦深度变深，下方的梯度会很小，如果梯度小，优化器就不能很好的使其进行优化，而且深度过深时的优化难度将会增大，所以这种类似短路连接的形式，使得模型能够在增加深度的同时得到很好的优化。

Densenet与ResNet在构建方式上很相似，不过Densenet的连接方式是将每一层都与前面的所有层连接起来，实现一种密集的连接。

那么总的来说，三个模型要求的最低输入图片大小都为 224×224 ，输入类型相同，三个模型的基本结构都是（卷积层 + BN层 + RELU层）* n + 池化层 + 全连接层。VGG是最基本的模型，而Resnet和Densenet与之不同，这两个模型中层与层之间会有连接。

ResNet是其中一些层之间的短路连接。

而在DenseNet中，每个层都会与前面所有层连在一起，是将各个层的特征都连接起来，这时我们可以重复使用其中的特征，达到提升效率的目的。可以看到，同一时期Densenet的模型通常比Resnet层数要少。

2.3.3 关于全连接层的选择

这里对于最后的全连接层有两个选择，一是输出两个值，这时候我们选用logsoftmax作为激活函数，softmax的作用是将所有输出结果，转化成和为1的数值，通俗易懂的讲就是分类为其中某一个的全部概率和为1，然后再对其求自然对数log，增大两者的差距，但不改变大小关系。那我们可以使用两个输出中较大那一个的索引，与我们的目标标记进行对比，但此时只能输出0和1。optimizer使用对用的NLLLoss。

另一个选择是只输出一个值，这时我们使用sigmoid作为激活函数，sigmoid函数的作用是将所有输出值，全部转换为0到1的值，因为我们的结果要求就是0为猫，1为狗，那么这时可以直接使用该值，并且分类错误之后的logloss惩罚更少。而这里的loss函数我们使用与之对应的binary cross entropy loss (BCELoss)，这个函数将能够计算我们得到的值离目标值‘差距’‘错误’到底有多大，，我们输出值为0到1，而目标值为0或1，那么这里选用BCELoss正合适。

2.3.4 关于优化器的选择

我所用到的优化器是Adadelta，这是一种自适应学习速率方法，优势在于设置好初始学习速率，之后就能自动调整学习速率，比较省心，学习速率可以根据loss和accuracy的情况判断是否需要手动调节。**它是计算简化版的Adagrad。**

2.3.5 关于dropout，正则化，迁移学习，模型融合及其他

dropout: dropout起到随机关闭某些节点，起到避免过拟合的作用，计算方式是随机的将某些节点的权重设置为0。

正则化: 起到优化模型，避免过拟合的作用。计算方式是 $error = loss + \sum(\text{参数}) * \lambda$ (这里是L1 Regularization, L2的话是参数的平方, λ 是可以设置的值, 值越大, 越限制模型复杂程度), 公式的含义也就是将模型的复杂程度也纳入考量, 使模型在复杂度和精确性上得到平衡。

迁移学习: 迁移学习的作用和好处在于, 对于该模型已经训练过的类似的图片, 无需再次训练其卷积层, 只需要将全连接层改变即可进行新的训练, 能够极大地提升训练效率。

原理: 这些现有架构是从无数个架构和大量的调试超参数的成果, 它们已经在庞大的ImageNet数据库上进行了长期训练, 这一架构已经学会区分ImageNet中存在的1000种不同类别的图片, 其中包括动物, 日常可见的物品等。我们知道卷积神经网络的第一层卷积神经一般是检测边缘和色块, 第二层则是圆圈, 条纹, 长方形等等, 第三层则可能会有眼睛, 鼻子或者鸟, 汽车。那么对于我们所要区分的对象: “猫”, “狗”, 我们可以利用这些已经训练好的过滤出来的特征对图片进行识别, 而不用再花费几个月的时间去重新调整超参数和结构, 然后再花几周的时间进行训练, 因为边缘色块, 圆圈条纹, 眼睛鼻子已经存在于这一构架当中, 所以预训练模型直接替我们省去了这一步骤。

而实现的方法只需要将先前的权重冻结, 然后将最后的几个层级替换即可使用。

模型融合: 模型融合的作用和好处在于, 能够将每一张图片在每个模型中所选取出的特征综合起来, 增加我们所能筛选出的特征, 也就是增加我们最终模型的精确性。而其中每一个模型都可以说是身经百战, 通过这几个模型导出的特征向量, 可以高度概括一张图片有哪些内容。而只要有了每张图多个模型的特征向量, 我们的工作就能正式开始了。

将使用win10下的pytorch, 使用带有cuda功能的rtx2070将有效提高计算速度。环境配置, 安装cuda等等都将在本地完成。

2.4 基准指标

由于本项目所涉及的指标有一个很统一的标准, 就是在kaggle上的leaderboard, 而该leaderboard是一个由logloss函数得分排名得出的, 那么其中的杨培文的logloss得分是值得参考的。而我所使用的rtx2070应该足以媲美2016年的旗舰单张gpu, 再加上模型优化, 各项技术的进步, 应该无需过多考虑机器性能上的问题。杨培文的模型在验证集上正确率最高达到99.6%, 在leaderboard上排名为前2%, logloss得分为0.04008。所以本项目的目标定为, 最高准确率能达到99.2%, 达到kaggle的leaderboard前5%, logloss得分小于0.05。他的github上利用tensorflow的猫狗大战代码: https://github.com/ypwhs/dogs_vs_cats (https://github.com/ypwhs/dogs_vs_cats)

3.方法

3.1 数据处理方法

3.1.1 处理异常值

首先针对数据集中的坏的数据进行处理。具体处理过程请看explore.ipynb。

第一步：针对训练集创建cat, dog两个文件夹。将文件名过滤，并针对含有不同字符cat, dog的，在两个文件夹内分别创建符号链接（可以节省空间，而不用复制粘贴进去）。这样做的结果既自动完成了分类，又节省了空间。

并针对测试集，对每一个图片，生成一个新的以图片序号文件夹，并在文件夹内创建符号链接，以备后续在dataloader中自动记录下文件名。这样做的结果同样节省了空间，并且使得pytorch中imagefolder的特性得以利用。

第二步：定义一个模型resnet152，并冻结参数。定义一个数据重新调整大小，中央截取，归一化，转到tensor的处理方式。以下处理方式在这一阶段对训练集测试集都适用。

重新调整大小：（255*255），目的在于留出一定的空间做中央截取center crop。

中央截取：（224*224），目的在于获得适合Resnet这一架构输入的图像。

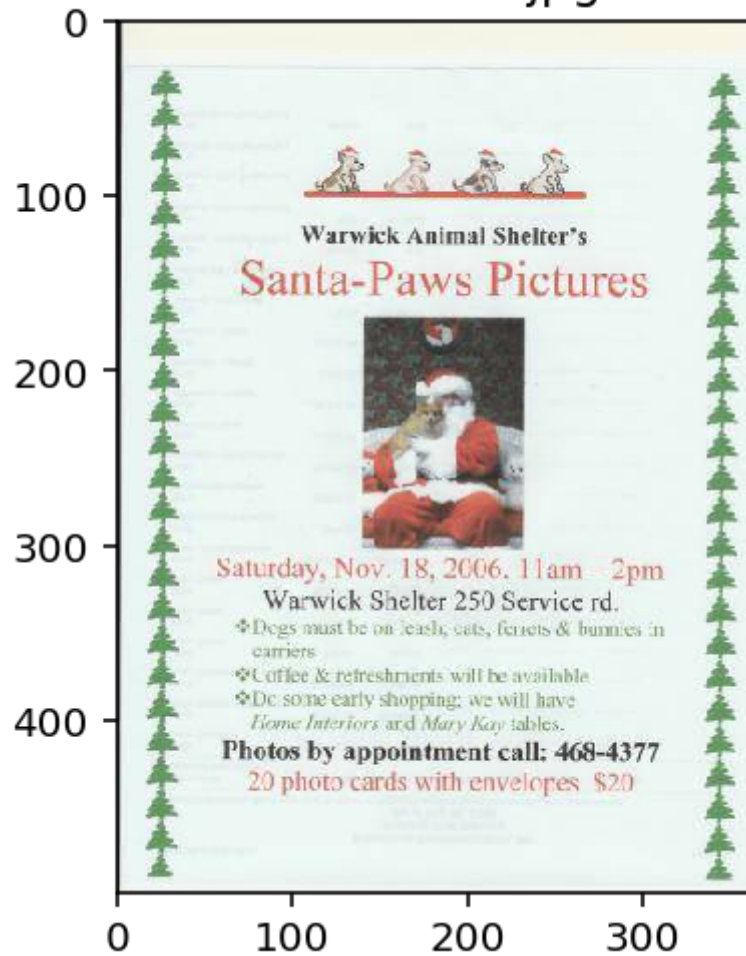
归一化：平均值设置为[0.471, 0.448, 0.408]，方差设置为[0.234, 0.239, 0.242]，平均值和方差中每个数对应一个位置，而计算公式为 $\text{input}[\text{channel}] = (\text{input}[\text{channel}] - \text{mean}[\text{channel}]) / \text{std}[\text{channel}]$ 。目的就是使得预处理的数据被限定在一定的范围内，从而避免奇异样本数据导致的不良影响。

转到tensor：因为模型只能处理tensor这一类型的数据，所以将图片转到tensor。

第三步：定义数个函数，找出resnet152无法识别为狗或者猫，并且识别为其他物体时偏差较大的图片，也就是resnet152预测结果与猫狗偏差过大的图片，将其symbolic link删去。我们不应该删除test中的数据，因为最终的结果必须有12500张图片的结果。具体函数内容请看explore.ipynb。

可视化：以下为所找到的异常值的代表，这一次的训练已将全部此类图片删除

cat.12272.jpg



cat.3566.jpg



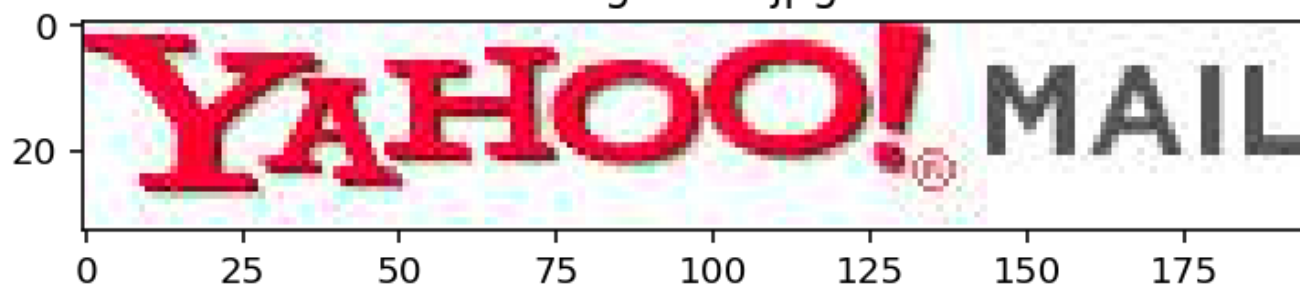
dog.10385.jpg



dog.9561.jpg



dog.4367.jpg



3.1.2 导出特征向量

第一步：定义两个数据随机翻转，随机旋转，调整大小，中央截取，归一化，转到tensor的处理方式。一个针对训练集，一个针对测试集。以下标注有的，意为该方法针对此集。

重新调整大小：(255*255)，目的在于留出一定的空间做中央截取center crop。——测试集，训练集。

中央截取：(224*224)，目的在于获得适合Resnet这一架构输入的图像。——测试集，训练集。

归一化：平均值设置为[0.471, 0.448, 0.408]，方差设置为[0.234, 0.239, 0.242]，平均值和方差中每个数对应一个位置，而计算公式为 $\text{input}[\text{channel}] = (\text{input}[\text{channel}] - \text{mean}[\text{channel}]) / \text{std}[\text{channel}]$ 。目的就是使得预处理的数据被限定在一定的范围内，从而避免奇异样本数据导致的不良影响。——测试集，训练集。

转到tensor：因为模型只能处理tensor这一类型的数据，所以将图片转到tensor。——测试集，训练集。

随机旋转，随机翻转：为了使模型更好的识别图片中的内容，用这种方法增强数据，使训练后的模型能够更好的泛化——训练集。

第二步：将训练数据和测试数据分别装载到datasets，并将数据分别装载到不同的数据生成器dataloader。

针对训练集的数据生成器：batch为16，并将洗牌shuffle关闭。batch为16目的在于针对我的显存正合适，shuffle设置为False的目的在于现在是在导出特征向量，并不需要针对其进行洗牌，训练最后的分类器时才需要用到，而此时需要利用训练集本身的特性来记录下loader中每张图片的标记。

针对测试集，batch为16，并将洗牌shuffle关闭。batch为16目的在于针对我的显存正合适，shuffle设置为False的目的在于测试时并不需要shuffle。

第三步：导入三个预训练好的模型vgg19_bn，resnet152，densenet161，将全连接层改为一层，并将全连接层输出数值个数改为2048。

接着冻结全部参数，使用三个模型，分别导出训练集，测试集的特征向量以及训练集的标记到文件。

3.2 执行过程

3.2.1 基础实验1

模型：resnet152。将全连接层改为linear层，全连接层输出改为2，冻结fc（全连接层）之前的所有参数。

激活函数：Logsoftmax。

损失函数：NLLLoss

optimizer：Adadelata。学习速率设置为默认1。

batch：16，不使用特征导出而直接使用原有数据。

过程：对分割好为cat和dog的数据，仅使用3.1.1第二步中的处理方式进行处理。接着使用定义好的训练函数对模型进行训练，尝试多次不同的epoch。最后预测测试集结果，导出csv预测文件提交到kaggle。

结果：logloss为17。

改进：1.将测试集的文件，以3.1.1中的第一步进行处理，改写输出csv的函数，这里需要注明一点的是，test_dataloader将会以test文件夹中的子文件夹作为标记，而test_datasets并不会按照文件名排序载入，而是与os.listdir('test2')所导出的顺序一致，这时将给每个文件创建一个新的文件夹，并以文件名标注，即可始终从os.listdir('test2')导出测试集文件名的顺序。

2.将全连接层输出改为1，激活函数改为sigmoid，损失函数改为BCELoss。因为激活函数和损失函数具有对应关系，Logsoftmax对应NLLLoss，适用于多分类；Sigmoid对应BCELoss，适用于二分类任务，所以在这里输出只需要1即可。关于函数的详细描述，请看proposal，我的开题报告。

3.2.2 基础实验2

模型：resnet152。将全连接层改为linear层，全连接层输出改为1，冻结fc（全连接层）之前的所有参数。

激活函数：Sigmoid。

损失函数：BCELoss。

optimizer: Adadelta, 学习速率设置为默认1。
batch: 16, 不使用特征导出而直接使用原有数据,

过程: 对分割好为cat和dog的数据, 开始使用3.1.2第一步中提到的增强数据处理方式。接着使用定义好的训练函数对模型进行训练, 尝试多次不同的epoch。最后测试集结果, 导出csv预测文件提交到kaggle。

结果: logloss为0.8

改进: 使用3.1.2中的方法, 使用预训练好的模型vgg19_bn, resnet152, densenet161导出特征向量, 并使用简单的relu层, 全连接层, Sigmoid激活函数作为一个分类模型, 用特征向量对该分类器进行训练。

3.3 完善

3.3.1 最终模型1

模型: 预训练好的模型vgg19_bn, resnet152, densenet161, 全连接层输出值为2048, 冻结所有参数; 简单分类器模型, relu+fc+sigmoid, 输出为1。

激活函数: Sigmoid。

损失函数: BCELoss。

optimizer: Adadelta, 学习速率设置为默认1。导出特征时的batch: 16。训练分类器时的batch: 16, 使用导出的特征向量对简单分类器进行训练 epoch: 8。

过程: 进行3.1.2中的导出特征向量操作(preprocess.ipynb), 再在main.ipynb中将每张图片对应的三个特征向量合并, 将新的向量分别载入新的train_datasets, test_datasets, 接着使用该train_datasets, test_datasets分别制作数据生成器train_loader, test_loader。在简单分类器模型上对train_loader进行训练, 并预测测试集结果, 导出csv预测文件提交到kaggle。

结果: logloss为0.1

完善: 1.降低平均每张图的学习速率, 意思就是增大训练分类器时的batch, 由于学习速率和batch是有关系的(batch越小, 学习速率平均每张图的加成效果越大), 所以为了降低logloss, 在适当减小学习速率时适当的增大batch大小。2.在预处理数据时加入去3.1.1除异常值的过程

3.3.2 最终模型2

模型: 预训练好的模型vgg19_bn, resnet152, densenet161, 全连接层输出值为2048, 冻结所有参数; 简单分类器模型, relu+fc+sigmoid, 输出为1。

激活函数: Sigmoid。

损失函数: BCELoss。

optimizer: Adadelta, 学习速率: 0.1。导出特征时的batch: 128。训练分类器时的batch: 128, 使用已经去除异常值的导出的特征向量对简单分类器进行训练 epoch: 8。


过程: 进行3.1.1中的处理异常值操作, 接着利用三个模型导出特征向量, 加载特征向量, 合并特征向量, 生成新的data_loader, 在简单分类器模型上对训练集进行训练, 预测测试集结果, 导出csv预测文件提交到kaggle, 随机可视化结果, 可视化模型训练曲线。

结果: logloss为0.04

4.结果

4.1 模型的评价与验证

最终模型非常合理，很好的结合了各个模型的优点。并且导出特征的做法，使得训练时的速度飞快，即使不用 cuda也能完成训练。最终的结果是得分达到了leaderboard前2%，远超预期水平。就结果而言，各项参数应该无误才能得到这样的结果。不过依然还有继续提升的空间。尤其是容易调整的学习速率，可以看到速率是稳定下降的。



Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · 2 years ago

OverviewDataKernelsDiscussionLeaderboardRulesTeam












My SubmissionsLate Submission

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_1555909732.5513306.csv	4 minutes ago	0 seconds	0 seconds	0.04111

Complete

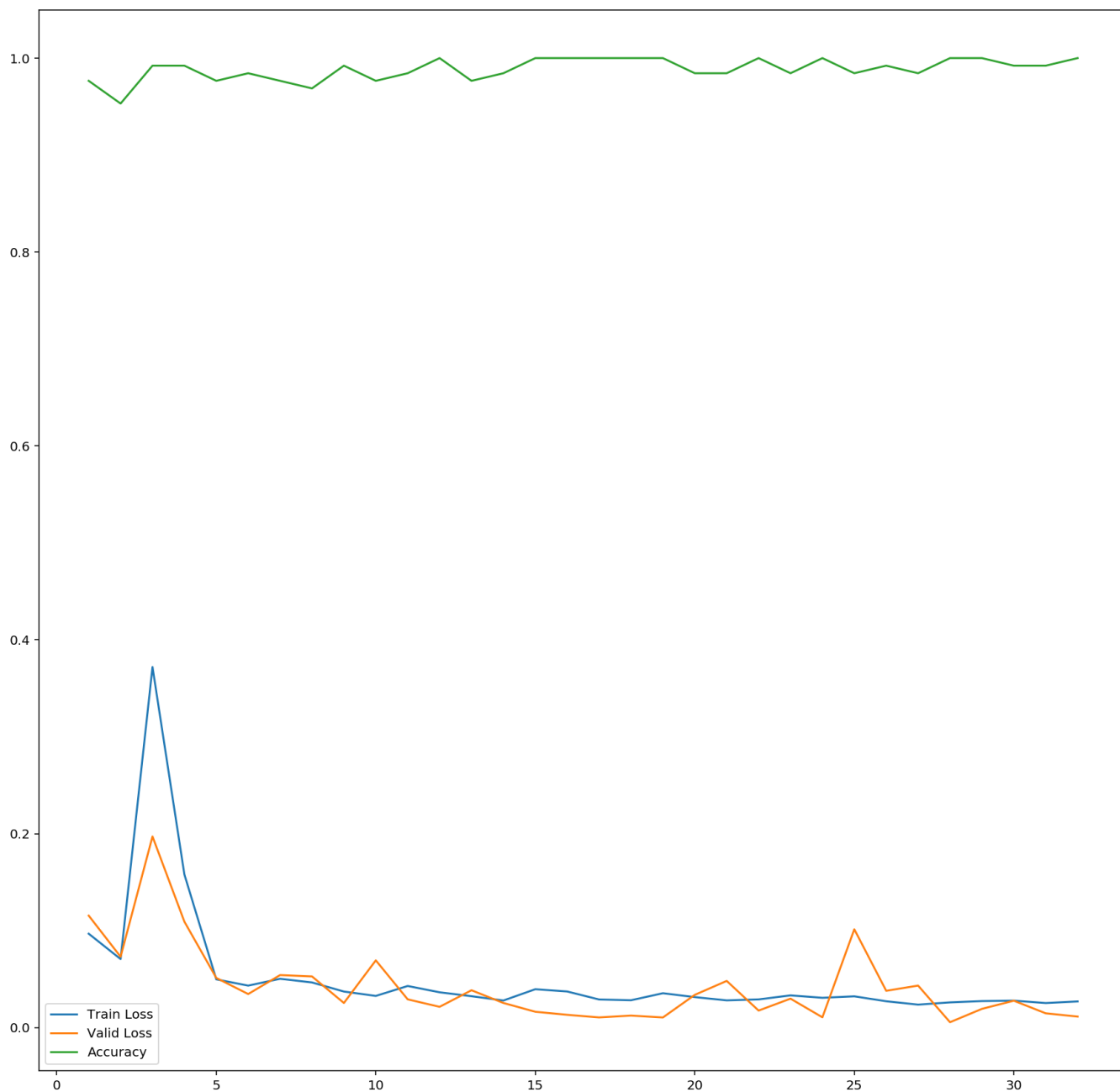
Jump to your position on the leaderboard

14	HMen		0.03928	12	2y
15	a.ewais		0.03994	50	2y
16	yangpeiwen		0.04008	27	2y
17	Anjith George		0.04077	46	2y
18	jbliss12345678		0.04127	13	2y
19	NK255		0.04134	6	2y
20	nash		0.04183	5	2y
21	travail		0.04214	61	2y
22	Damodar		0.04280	24	2y
23	jbargu		0.04290	11	2y
24	Rongcheng Lin		0.04337	39	2y

该模型的鲁棒性较强，因为综合了三个模型，并且训练数据量不小，微小的输入数据和训练数据的偏差将不会极大地影响结果。经过验证，剔除了异常值和没有剔除异常值的logloss结果相差无几。这个模型得出的结论非常可信，就logloss的得分来看，准确率高于预期。

4.2 合理性分析

我的基准模型预期为前5%，但最终的结果为前2%，超出了我所设定的目标。在测试集上的准确率也非常高，确确实实的达到了项目的要求，解决了这一经典猫狗大战的问题。那么对于训练曲线，如下图所示，已经收敛，由于采用的是training accuracy，并且是在特征向量上进行训练，所以accuracy这一条线一开始就很高。



5. 项目结论

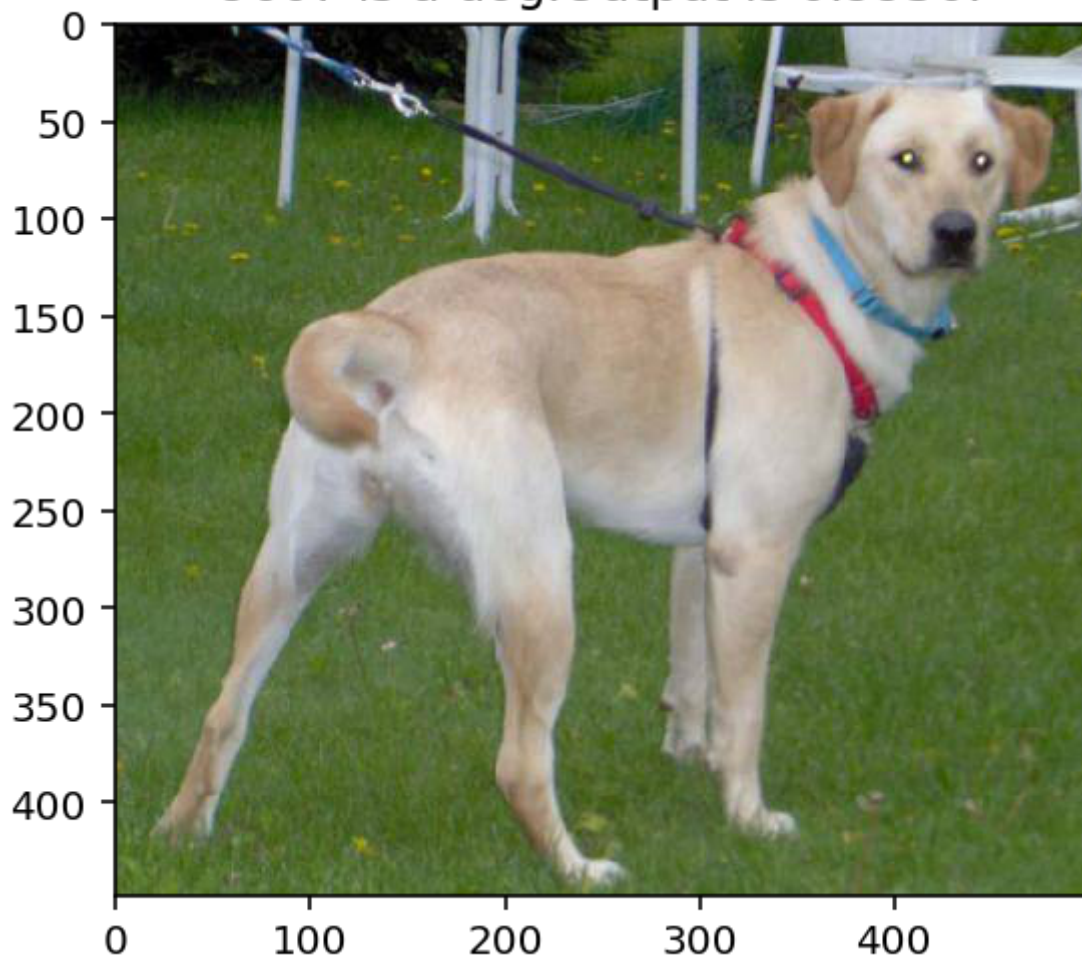
5.1 结果可视化

结果实例，可在main.ipynb里看到更多。

5788 is a cat. Output is 0.0050.



9607 is a dog. Output is 0.9950.



对于结果的可视化，我不光选择了显示csv图表，并且选择使用random功能来随机抽样查看预测结果，只需要改变seed的大小，就可以轻松查看下一批图片。

就可视化的结果而言，logloss得分很大程度上反映了预测的准确性，在反复检查了几组之后，并没有发现分类错误的图片。

结果可视化，可在main文件中找到。

5.2 对项目的思考

整个项目中，最让人上瘾的，就是困在某个点不出来之后单纯的思索。

在最开始所做的工作非常平常，查看数据，思考数据，在脑内产生一个大概的流程，然后根据这一流程对数据进行预处理，需要格外注意的是，图像的维度，格式，以及模型所要求的维度，格式，这一点上我踩了非常多的坑，也千百次的查看pytorch的documentation从坑里跳了出来，维度，格式，甚至tensor的类型都是需要格外注意的，比如BCELoss所需要的精度和NLLLoss需要的精度完全不同。幸好官网有转换tensor类型的函数。对于预处理，还有一个难点就是，imagefolder并不会按照文件名来排序，反而是有些乱七八糟，但imagefolder的排序，和使用os列出文件名的顺序是一样的，所以幸好可以这样操作，否则只能另辟蹊径。

预处理告一段落之后，就会进入模型的选择，由于项目的不同，我们可能需要选择不同的模型，迁移学习是应对普通分类问题非常好的方法，需要注意的是，每个模型前都需要pretrained=True才可以用已经预训练的模型。使用混合模型对数据进行预处理，是一个非常不错的选择，三个臭皮匠，顶一个诸葛亮。不过还是一样，这时我们依然需要十分注意预处理之后的结果，应该是怎么样结合起来的。

对于模型最后的全连接层，是较为轻松的部分，唯一的难点在于，激活函数和loss函数的选择。对于二分类问题使用sigmoid，多分类问题使用logsoftmax，各自相对应的loss函数为BCELoss，CrossEntropyLoss。而学习速率的设置，优化器的选择，训练函数的编写则相对比较简单。

最后则是将所得到的模型进行训练，对参数进行调优，到这里其实已经没有太多可以调了。接着就是将最后的小模型，也就是分类器，应用于预训练好的测试集，运行起来飞快。在以后的训练当中，先导出特征，再慢慢考虑其他事务，是一个非常不错和优先的选择。

最终得出的结果我很满意。不过对于在日常使用的场景下解决问题，它似乎并不太适用，一是需要三个预训练好的模型导出特征；二是需要再编写一个针对单张图片，甚至视频的程序，总的来说就是比较笨重，需要重写新的代码适应新的需求。但是对于项目本身的目标来说，可以说很圆满的完成，对于大型数据集的分类任务，这样做既高效，准确率又高，并且仅需改改代码就可以复用。

5.3 需要作出的改进

如果以现有的模型作为基准模型。所能做出的改变有以下几个方面。

一个是尝试在三个预训练混合模型的基础上，往上再叠加一个新的模型导出的特征。

第二个是在学习速率上下功夫，pytorch有一个叫scheduler的优化器辅助工具，专门优化学习速率，在项目中我也只是尝试了一下，并没有深入研究，若想使模型继续学习，这是一条可行的道路。

还有一个是更加困难复杂一些的思路，就是将预训练中每一层卷积层提取出的特征，对其进行处理，将多个类似的特征，像是混合模型一样叠加起来，针对猫和狗身体的某一个部位。进行区分，比如区分眼睛的层，区分耳朵的层，区分嘴巴的层，仅将这些有用的层提取出来，无效的层去除掉，然后再进行叠加。

参考文献

For VGG: Karen Simonyan & Andrew Zisserman. (2014 Sep 4). Very Deep Convolutional Networks for Large-Scale Image Recognition.

For Resnet:

Kaiming He, Xiangyu Zhang, Shaoqing Ren & Jian Sun. (2015 Dec 10). Deep Residual Learning for Image Recognition.

For Densenet:

Gao Huang, Zhuang Liu, Laurens van der Maaten & Kilian Q. Weinberger.(2016 Aug 25). Densely Connected Convolutional Networks.

Pytorch Transfer Learning Tutorial:

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
(https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html)

For Sigmoid, Logsoftmax, BCELoss & CrossEntropyLoss:

<https://pytorch.org/docs/stable/nn.html?highlight=sigmoid#torch.nn.Sigmoid>
(<https://pytorch.org/docs/stable/nn.html?highlight=sigmoid#torch.nn.Sigmoid>)
<https://pytorch.org/docs/stable/nn.html?highlight=logsoftmax#torch.nn.LogSoftmax>
(<https://pytorch.org/docs/stable/nn.html?highlight=logsoftmax#torch.nn.LogSoftmax>)
<https://pytorch.org/docs/stable/nn.html#torch.nn.BCELoss>
(<https://pytorch.org/docs/stable/nn.html#torch.nn.BCELoss>)
<https://pytorch.org/docs/stable/nn.html?highlight=crossentropy#torch.nn.CrossEntropyLoss>
(<https://pytorch.org/docs/stable/nn.html?highlight=crossentropy#torch.nn.CrossEntropyLoss>)

Mixed Model Training:

Yang Peiwen:https://github.com/ypwhs/dogs_vs_cats (https://github.com/ypwhs/dogs_vs_cats)

About Advantages of Models:

<https://cloud.tencent.com/developer/news/320509> (<https://cloud.tencent.com/developer/news/320509>)
<https://zhuanlan.zhihu.com/p/37189203> (<https://zhuanlan.zhihu.com/p/37189203>)