CVDL 第一次作业报告

一、作业概述

（1）题目：细粒度分类

（2）工具：pytorch + jupyter lab 远程连接服务器 + tensorboardX 可视化

（3）算力：1 块 GPU

（4）网络模型：以 VGG16 为基础进行修改

（5）训练算法：Adam（optim）+CrossEntropy（loss）+DataAugmentation

（6）kaggle_score：0.98888

二、工作过程描述:

写这次作业时，我先后完成了以下工作：

（1）学习 pytorch tutorial。

之前没有接触过 deep learning 和 pytorch，所以趁着五一假期，我学习了助教老师推荐的教程，并参考相关博客搭建了环境，并学会了通过 jupyter lab 远程访问 GPU 资源。

（2）下载、加载数据。

将数据集下载到服务器，然后利用 torch.utils.data.Dataset 和 ttorch.utils.data.DataLoader 加载数据，并编写了基本的 transform 和 imshow 函数，用于把图片数据打印出来。

（3）搭建出第一个网络，在 CPU 上运行。

利用 torch.nn.Module 搭建网络，并编写迭代训练的代码。第一次试水，当然写的非常简单啦（conv(3,1,5)+relu+fc+softmax 各一层）。

（4）**对 VGG16 进行修改，在 GPU 上运行。**

想法：自己从头搭建的模型很容易出现训练结果不理想的情况，于是我决定选择一款成熟经典的模型来作为基础。

之所以选择 VGG，有如下理由：

a. VGG 的结构非常清晰（conv、relu、maxpool 的简单堆叠）;

b. 参数不是太多（以 VGG16 为例，有 13 个卷积层和 3 个全连接层；而且 kernel_size 全部为 3；经过测试，我在 1 块 GPU 上对整个 VGG16 的参数训练 1 个 epoch 只需要 3.5 分钟左右，因此训练几十个 epoch 的情况下也只需要等待几个小时）

c. 非常经典和常见（VGG 于 2014 年提出来，由于其简洁性和实用性，此后的很多研究都以它为基础进行改进）

实现 VGG 网络后，基于输出的类别数目（180），将一个(input_size = 4096, output_size = 1000)的 FC 层改写为由 relu 层和 dropout 层连接的两个 FC 层：

```
(6): Sequential(
  (0): Linear(in_features=4096, out_features=1024, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.4, inplace=False)
  (3): Linear(in_features=1024, out_features=180, bias=True)
  (4): LogSoftmax()
)
```

（5）错误分析。

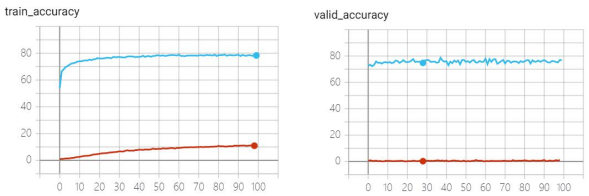
使用预训练的 VGG16 模型，第一次训练了 15 个 epoch 后，我发现情况不太对……在训练集上的准确率只提升到了 12%左右，在验证集上的准确率更是始终在 0.5%左右徘徊。

于是我决定暂停训练并查找了原因。

我发现，输出有 180 类的情况下，随机瞎猜的准确率就是 1/180=0.556%，因此说明网络在训练集上没有学到任何有用的规律。但事实上，预训练的 VGG 网络应该很好地提取图片的特征，因此说明输入网络的训练数据有问题。

于是经过排查，终于发现在加载数据集时，（由于 os.listdir 函数随机读取文件），图片和标签是随机匹配的。

改正后，训练集和验证集上的准确率均达到了 70%以上。



（6）多种训练算法的组合。

进行了如下的尝试：

a. 本题是一个分类任务，所以我选择了交叉熵作为误差指标。

b. 梯度下降算法尝试了 SGD 和 Adam 两种算法，并在 adam 算法下尝试了不同的学习率。对于 Adam 算法，我尝试了将学习率分别设为 0.001=1e3（default）和 0.0001=1e4，发现二者的结果可谓天壤之别——1e4 可以保证在前面 10 个 epoch 以内就收敛到很好的结果，而 1e3 在第一个 epoch 就低于 10%的准确率，令人震惊（我本以为 1e3 会更快地收敛）。对于这样的结果，我分析认为可能是由于 VGG 的预训练地很棒，所以当学习率很高、而 Adam 算法本身也会在训练初始时倾向于更激进地更新权重时，模型的误差直接从谷底蹦到了山顶（这个描述很生动 hhhh）。

c. 我的训练策略是，先冻结卷积层、只训练作为分类器的全连接层，再训练整个网络、改进卷积层特征提取的参数。可能是由于 VGG 的预训练地很棒，前后两个步骤都分别在大约 10 个 epoch 就收敛了。

（7）数据扩充。

经过上述尝试，测试集准确率达到了 97.555%。我决定再尝试一下数据增广。

我使用的增广方法是利用 torchvision.transforms 设置对图片的转换方法，这样在每个 epoch 中用于训练的数据都不一样（因为我在 transforms 中用到了很多随机的方法）。

第一次尝试的结果并不理想，验证集的准确率降低到了 90%甚至更低。我决定将使用转换算法处理后的图片都打印出来，结果发现很多转换后的图片中都只有鸟的身体。于是我得到启发：由于数据集的原始图片中，鸟头都在图片的边缘处，而我使用了随机旋转再放缩的处理方法，所以鸟头（嘴）就很容易被切割掉了，又由于对于鸟类的识别来说，头部的特征最丰富的也最重要，所以准确率就大大降低了。

吸取了上述经验之后，我修改了转换函数：减小旋转的幅度和缩放的比例，并减少这一类处理的比例，更多地采用图片翻转和亮度、饱和度微调（不是颜色哦）的方法。

最终将测试集的准确率提高到了 98%以上。（训练集的准确率收敛到 95&以下，这是因为训练集进行数据扩充后难度增加了，所以准确率反倒降低；但是泛化能力增强使得在验证集和测试集上的准确率提高了）

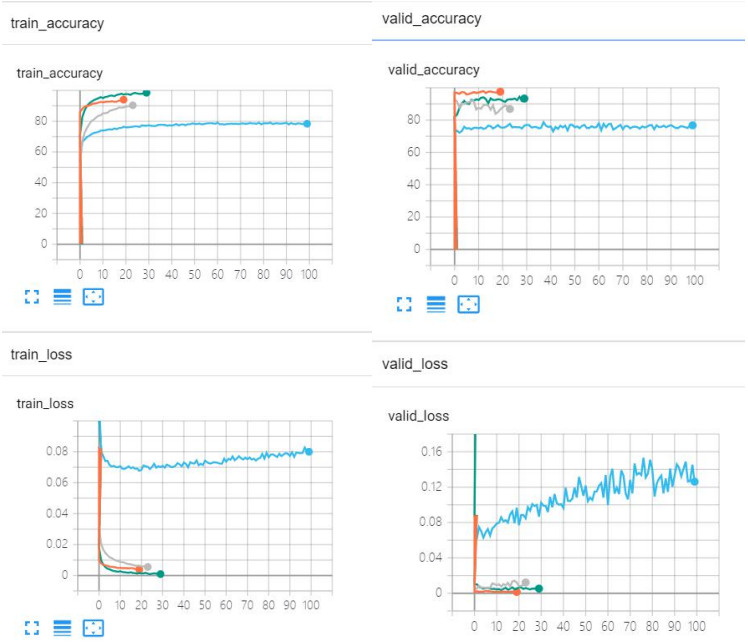
结果如下图：

（1）浅蓝色是冻结卷积层、只训练 FC 层，验证集准确率在 70%-80%；

（2）绿色是在（1）的基础上对整个网络进行训练，验证集准确率在 90%-95%；

（3）灰色是在（2）的基础上第一次对数据增广（不理想），验证集准确率在 85%-90%；

（4）橙色是在（2）的基础上第二次对数据增广，验证集准确率在 95%-99%；



（8）**总结并撰写报告。**