实验二 猫狗分类

实验内容

实验二是猫狗分类，使用`Kaggle`猫狗分类的原始数据集，实现模型最终的准确率达到75%及以上。本实验的目的是为了进一步掌握使用深度学习框架进行图像分类任务的具体流程：如读取数据、构造网络、训练和测试模型等等。实验报告将从实验环境、数据预处理、实验分析4个方面进行阐述。

实验环境

本实验采用`Anaconda3`+`Pycharm`框架进行开发，深度学习框架采用`PyTorch`框架，各类版本号如下：

- `python 3.7.10`
- `pytorch 1.8.1 GPU`
- `CUDA 10.2`
- `cudnn 8.2.0`
- `tensorboard 2.4.1`
- `tensorboardX 2.2`
- `hiddenlayer 0.3`

数据预处理

数据集分配

- 训练数据集：从原始训练数据集中选择16000张打乱的猫狗图片中选择70%作为训练数据集，共计11200张
- 验证数据集：从原始训练数据集中16000张打乱的猫狗图片中选择30%作为训练数据集，共计4800张
- 测试数据集：从原始训练数据集中选择4000张作为测试集

预处理

- 训练数据集进行数据增强

```
transforms.Compose([
    transforms.Resize(size=(256, 256)),
    transforms.RandomResizedCrop(size=(224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

- 测试与验证数据集不进行数据增强

```
transforms.Compose([
    transforms.Resize(size=(256, 256)),
    transforms.CenterCrop(size=(224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

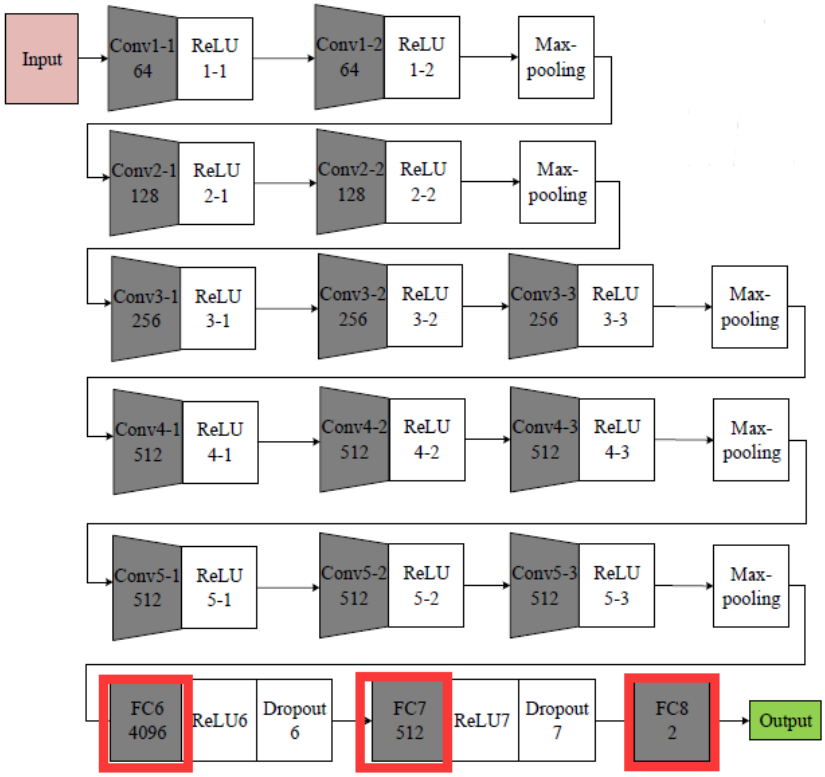
实验分析

构建卷积神经网络，分别复现 `VGG16` 与 `ResNet50` 两种经典的CNN网络，并分别使用 `Pytorch` 的预训练模型进行对比实验。下面分别对 `VGG` 以及 `ResNet50` 进行了介绍，并比较了实验结果。

VGG16-原始训练

网络结构

`VGG16` 是典型的块结构，结构规整，具有很强的拓展性。`VGG16` 网络模型中的卷积层均使用3×3的卷积核，且均为步长为1的same卷积，池化层均使用2×2的池化核，步长为2。原始论文里的分类是1000类，而在猫狗分类中只需要将其分为两类。因此修改一下VGG16的三个全连接层，如下所示：



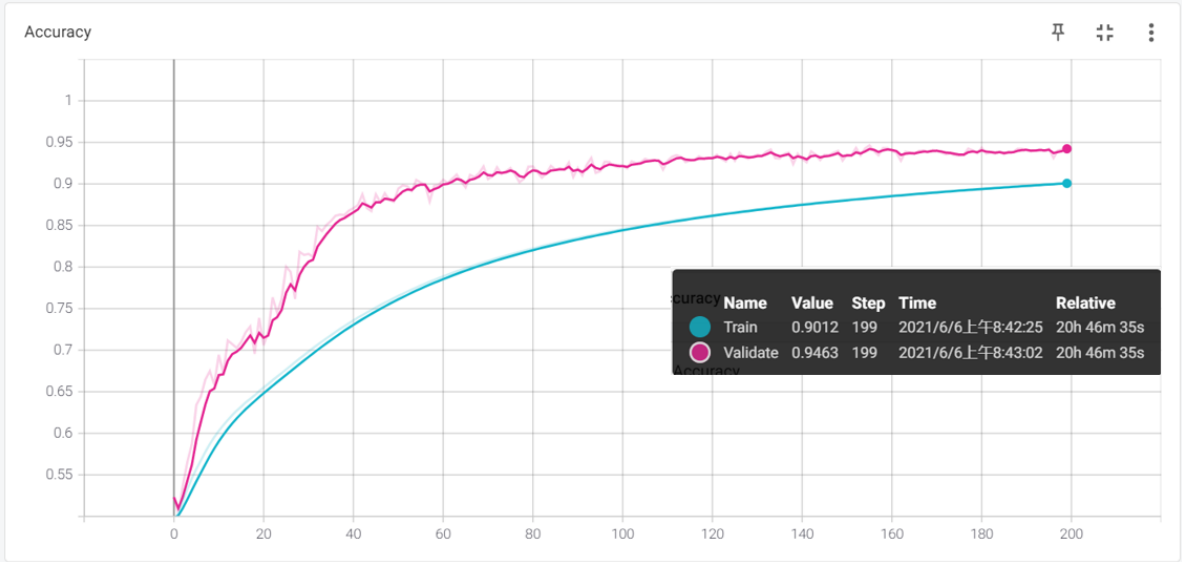
其中具体模型参数设置如下所示：

- 梯度更新：`SGD` 随机梯度下降，且动量 `momentum` 为0.01；
- 损失函数：采用 `CrossEntropyLoss` 交叉熵损失函数；
- 超参设置：
  - 批量大小 `batch_size`：10
  - 学习率 `lr`：0.01，并在5轮损失不下降的平台区降低10倍
  - 迭代次数 `epochs`：200

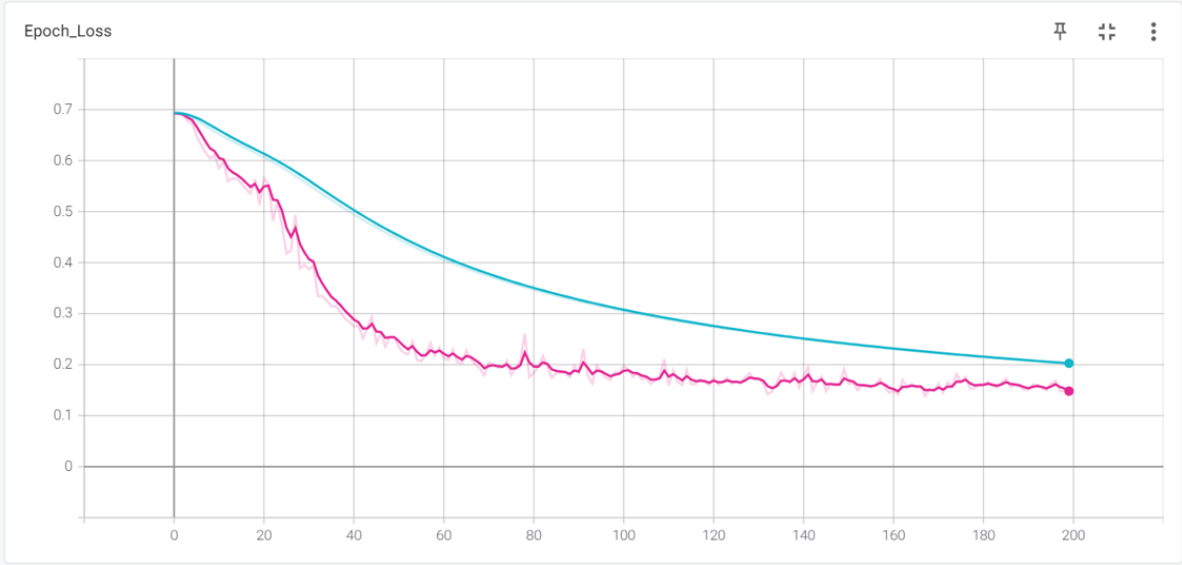
实验结果

在训练到第66轮时，训练集准确率就已经达到了**80%**，最终得到的表现最好的模型是在迭代结束之后的模型，训练准确率达到了**90.12%**，验证集准确率达到了**94.63%**，并且最后在测试集上的准确率为**94.83%**。

- 训练与验证准确率



- 训练与验证损失



VGG16-预训练模型

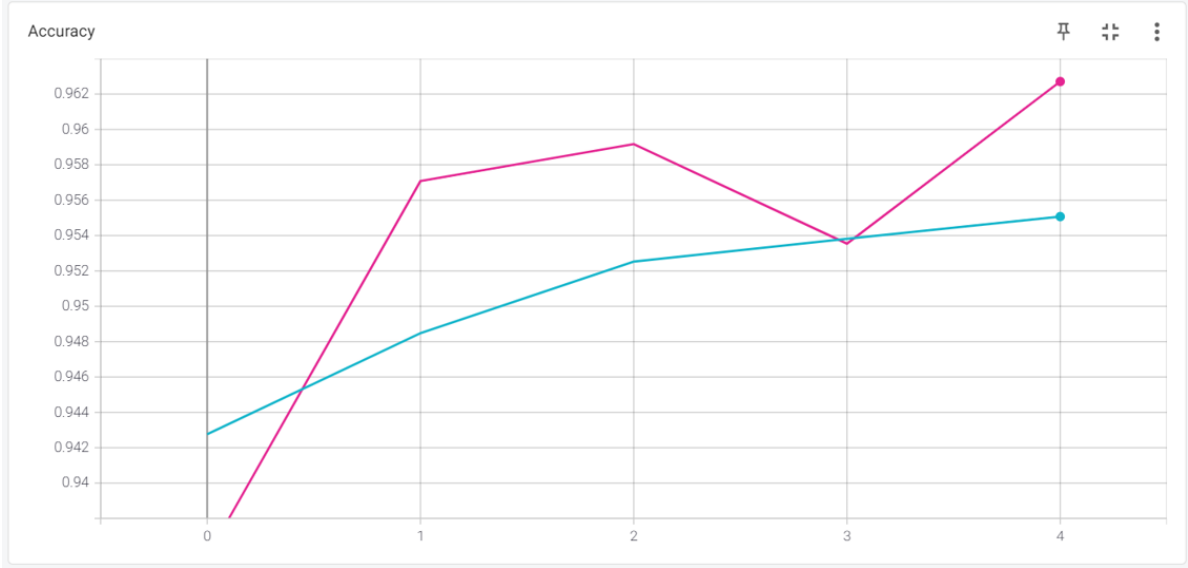
使用`Pytorch`中所带的`VGG16`的预训练模型，并将其进行微调，具体代码如下：

```
vgg16 = models.vgg16(pretrained=True)
for parameter in vgg16.parameters():
    parameter.requires_grad = False
vgg16.classifier = nn.Sequential(
        nn.Linear(512 * 7 * 7, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(),
        nn.Linear(4096, 512),
        nn.ReLU(inplace=True),
        nn.Dropout(),
        nn.Linear(512, 2),
)
```

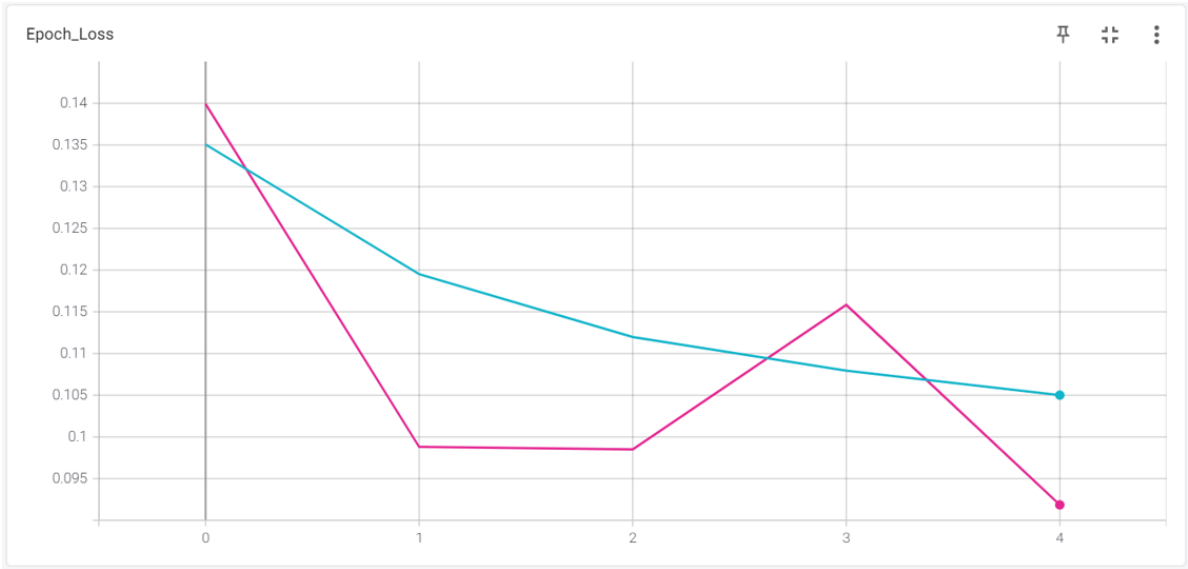
实验结果

使用预训练模型所设置的模型参数除了迭代次数设置为**5**外基本不变。下图是使用`VGG16`预训练模型的实验结果，可以看到预训练模型十分强大，在一个epoch时就已经能够使得训练准确率达到**94%**以上，最终得到的最好的模型是在`epoch=4`时得到的，训练准确率达到了**95.51%**，验证准确率达到了**96.27%**，并且最后在测试集上的准确率为**98.28%**。

- 训练与验证准确率



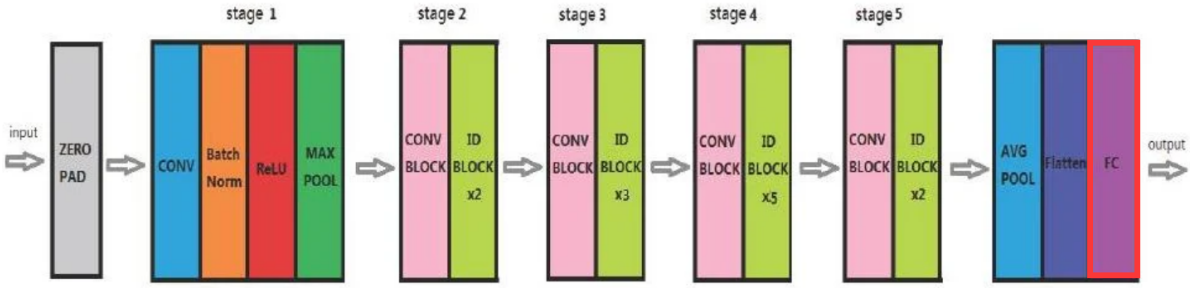
- 训练与验证损失



ResNet50-原始训练

网络结构

`ResNet50`通过捷径连接能够实现更快的收敛速度，且采用BN能够在一定程度上防止过拟合而不再使用丢弃法等策略。通过在全连接层修改输出的类别数为2，`ResNet50`的网络结构如下所示：



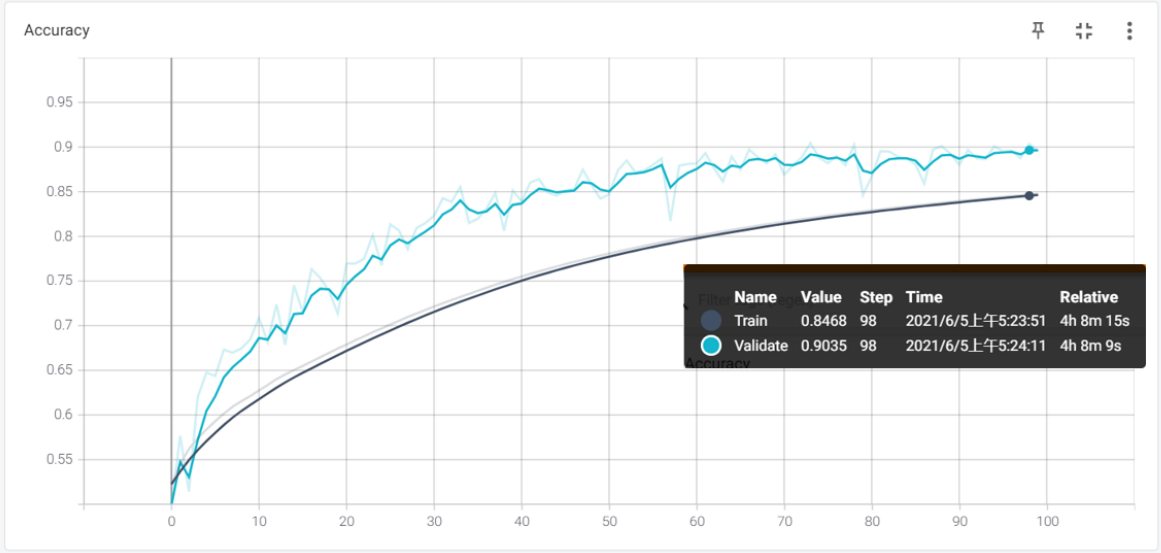
其中具体模型参数设置如下所示：

- 梯度更新：`SGD`随机梯度下降，且动量`momentum`为0.01；
- 损失函数：采用`CrossEntropyLoss`交叉熵损失函数；
- 超参设置：
  - 批量大小`batch_size`：10
  - 学习率`lr`：0.1
  - 迭代次数`epochs`：100

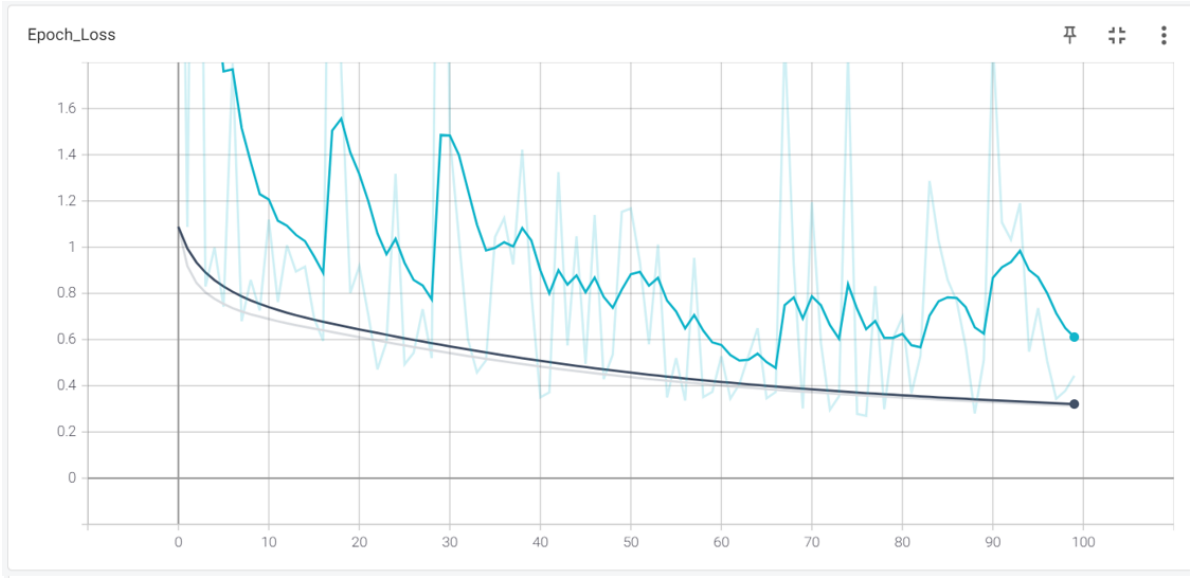
实验结果

在训练到第73轮时验证准确率就达到了**90%**，验证准确率高于测试准确率的原因可能是由于只有训练数据集进行了增强。最终得到的最好的模型是在`epoch=98`时得到的，训练准确率达到了**84.68%**，验证准确率达到了**90.35%**，并且最后在测试集上的准确率为**92.27%**。

- 训练与验证准确率



- 训练与验证损失



ResNet50-预训练模型

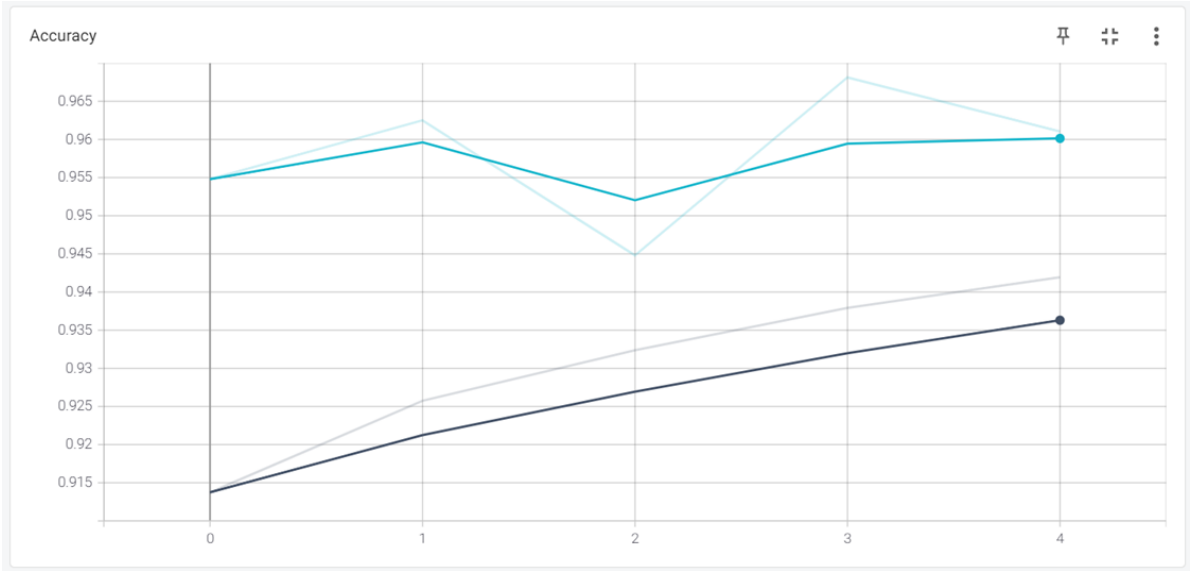
使用`Pytorch`中所带的`ResNet50`的预训练模型，并将其进行微调，具体代码如下：

```
resnet50 = models.resnet50()
numFit = resnet50.fc.in_features
resnet50.fc = nn.Linear(numFit, 2)
```

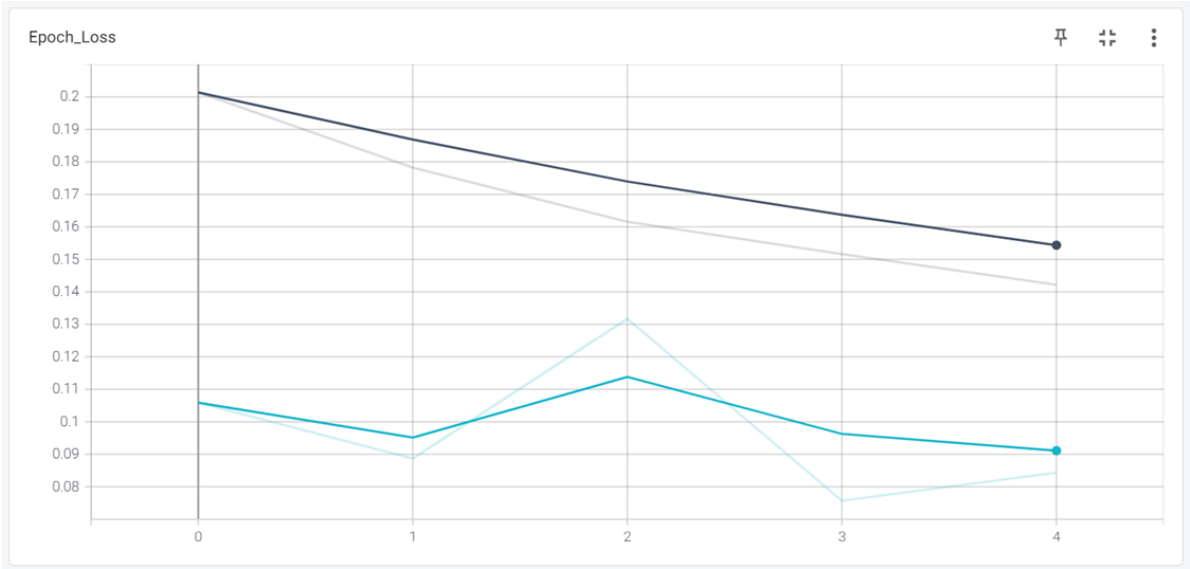
实验结果

使用预训练模型所设置的模型参数除了迭代次数设置为**5**外基本不变。下图是使用`ResNet50`预训练模型的实验结果，可以看到预训练模型十分强大，在一个epoch时就已经能够使得训练准确率达到**90%**以上，最终得到的最好的模型是在`epoch=4`时得到的，训练准确率达到了**94.19%**，验证准确率达到了**96.10%**，并且最后在测试集上的准确率为**97.02%**。

- 训练与验证准确率



- 训练与验证损失



小结

通过这次实验，我学会了如何利用`Pytorch`深度学习框架复现经典的CNN网络如`VGG16`、`ResNet`，以及如何使用预训练模型进行微调。更加深刻的理解了卷积神经网络，并且对于实验中的出现的各种问题，例如学习率的设置问题、损失不下降的问题、过拟合的问题都有了一定的解决思路，对于后续在专业领域的研究中应用深度学习起到了很好的实践作用。最后依然是特别感谢徐老师及助教的培养与付出！