程序设计实训-hw3

江灿　2019011325　jiang-c19@mails.tsinghua.edu.cn

2.1 图片可视化

通过task-1.py文件，输出得

2.2 训练模型并使用它进行推理

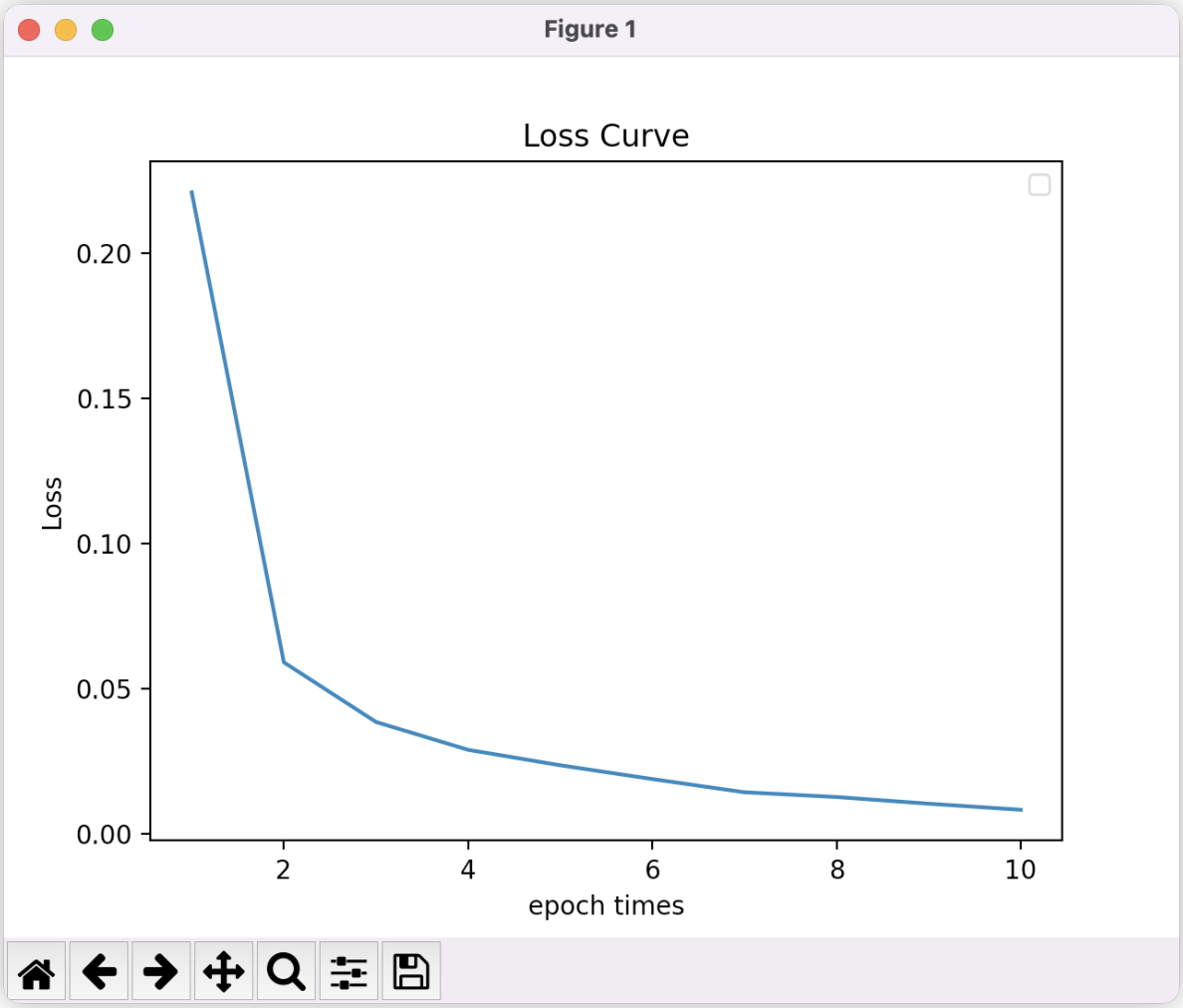
运行 `train.py` ，完成模型的训练和保存，然后运行 `inference.py`，完成模 型的推理。

2.3 绘制Loss曲线

绘制时导入matplotlib包进行绘制，绘制时先在每次训练时记录下loss，在一个epoch训练完成时，求得平均值，记录下来。最后在训练结束时，使用如下命令完成绘制

```
plt.figure()
    plt.plot(range(1, 11), average_loss)
    plt.xlabel('epoch times')
    plt.ylabel('Loss')
    plt.title('Loss Curve')
    plt.legend()
    plt.show()
```

绘制所得图像为



2.4 更换优化器

更改`train.py`中的相应代码，将优化器Adam改为SGD，更改代码如下

```
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

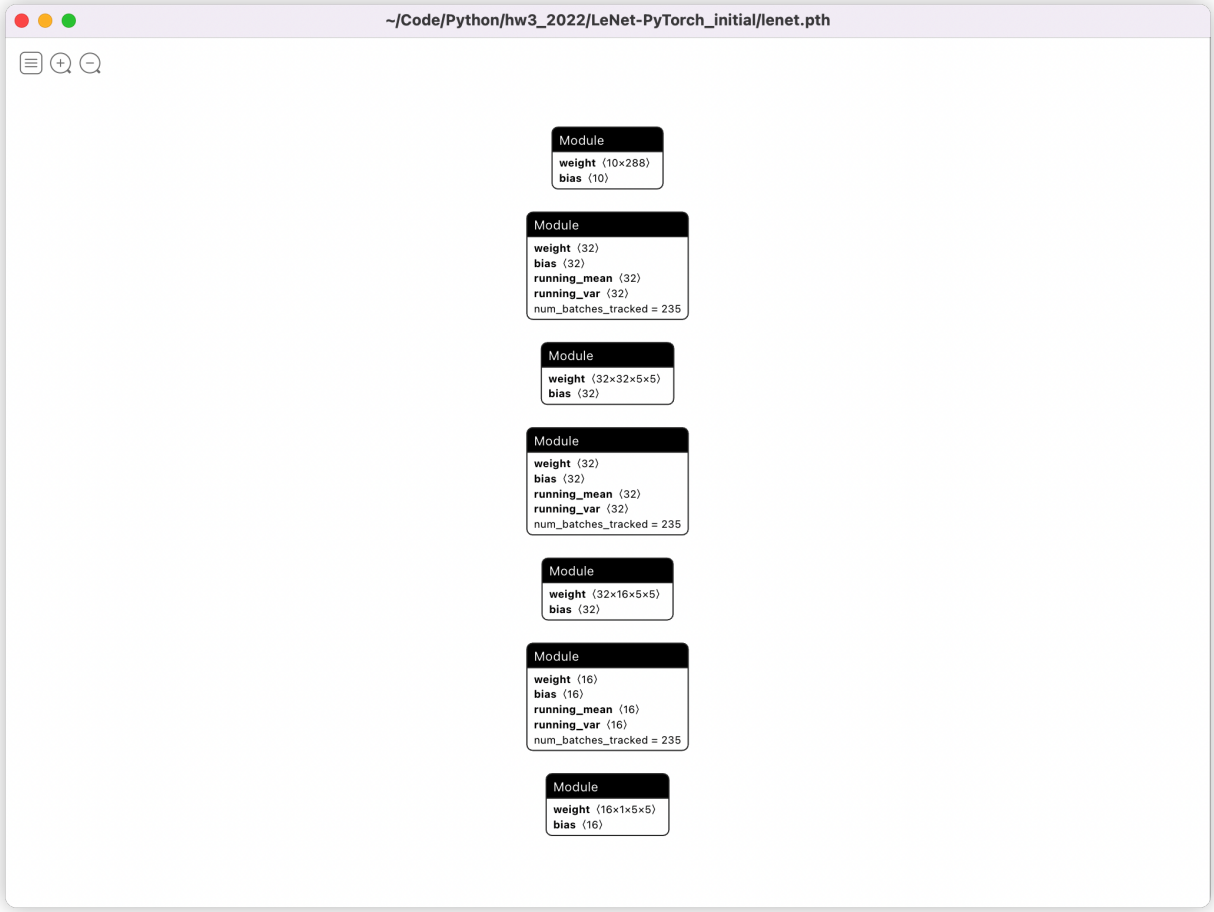
更改后，训练时间变长，且训练的Accuracy变低，可以看出不同的优化器在进行训练的时候会起到不同的作用，使用尽可能好的优化器或者自己造出一个更好的优化器可能是以后需要完成的事情。

2.5 添加数据预处理

```
def data_augment_transform():
    data_augment = torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.RandomErasing(),#随机裁剪
        torchvision.transforms.RandomHorizontalFlip(p=0.5),#水平翻转
        torchvision.transforms.RandomVerticalFlip(),#垂直翻转
    ])
    return data_augment
```

1. 对于随机裁剪，多次实验后发现模型训练平均Loss略高于不使用随机裁剪，但是模型训练Accuracy最开始略低于不裁剪，最后却始终稍高于不进行裁剪操作（如98.91与98.89）。
2. 对于水平翻转，多次实验后发现模型训练平均Loss明显于不使用水平翻转，模型训练Accuracy低于不翻转。
3. 对于垂直翻转，多次实验后发现模型训练平均Loss明显于不使用垂直翻转，模型训练Accuracy低于不翻转。

2.6 模型导出及模型可视化


~/Code/Python/hw3_2022/LeNet-PyTorch_initial/lenet.pth
Module
weight (10×288)
bias (10)
Module
weight (32)
bias (32)
running_mean (32)
running_var (32)
num_batches_tracked = 235
Module
weight (32×32×5×5)
bias (32)
Module
weight (32)
bias (32)
running_mean (32)
running_var (32)
num_batches_tracked = 235
Module
weight (32×16×5×5)
bias (32)
Module
weight (16)
bias (16)
running_mean (16)
running_var (16)
num_batches_tracked = 235
Module
weight (16×1×5×5)
bias (16)

2.7 错误样例分析

1.

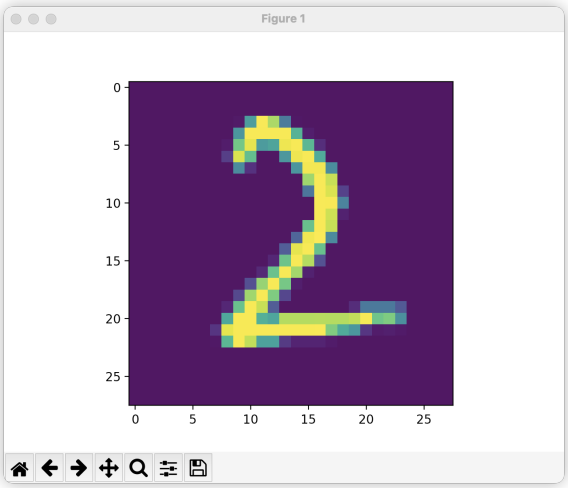
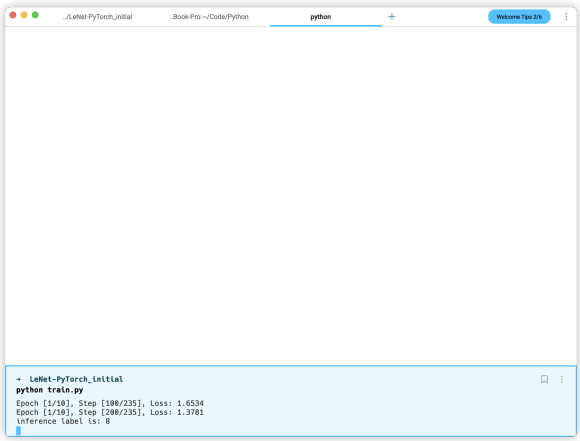


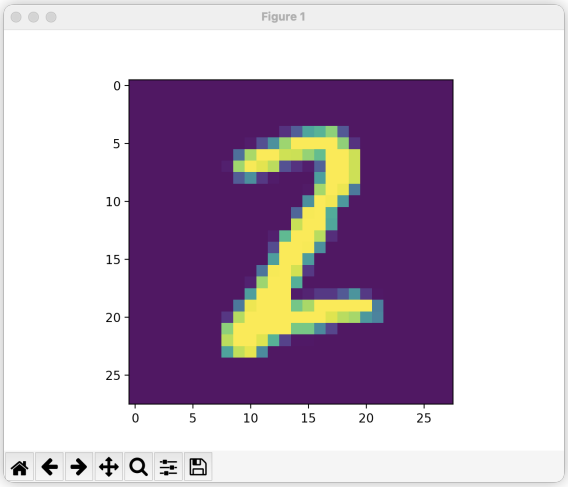
Figure 1
0
5
10
15
20
25
0
5
10
15
20
25

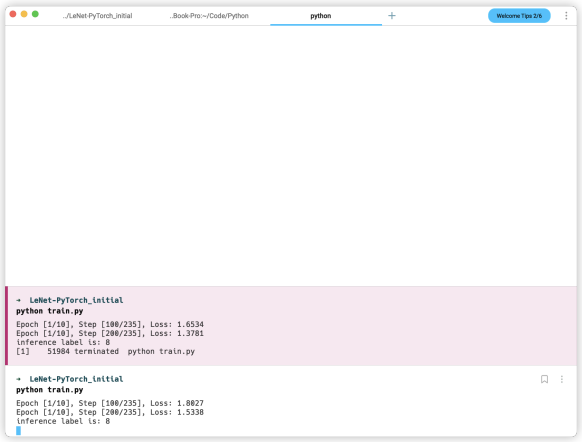

../LeNet-PyTorch_initial
..Book-Pro:~/Code/Python
python
Welcome Tips 2/6
→ LeNet-PyTorch_initial
python train.py
Epoch [1/10], Step [100/235], Loss: 1.6534
Epoch [1/10], Step [200/235], Loss: 1.3781
inference label is: 8

推断为8

可能是这种上面带有向下的弧度的2会看起来很像8，可能学习的时候通过类似的没有被封口的会被误认为8

2.

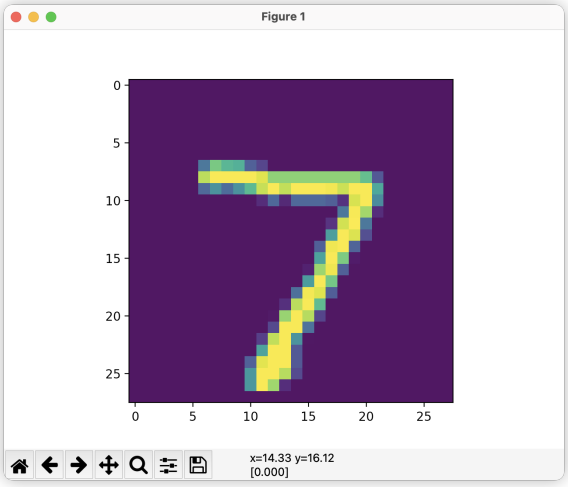


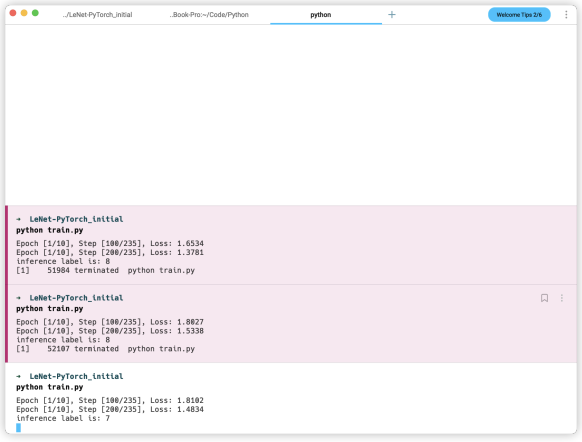


推断为8

同上

3.

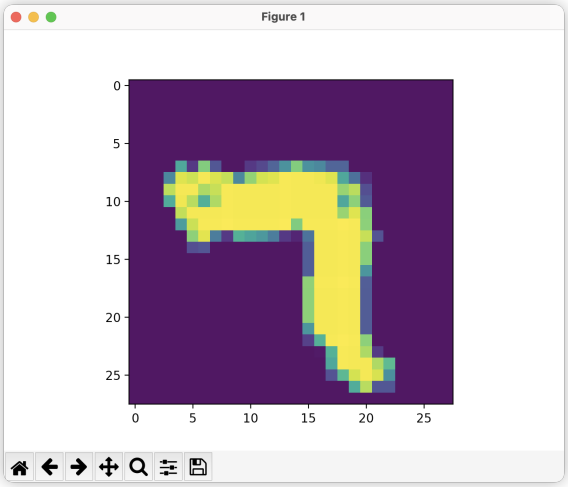


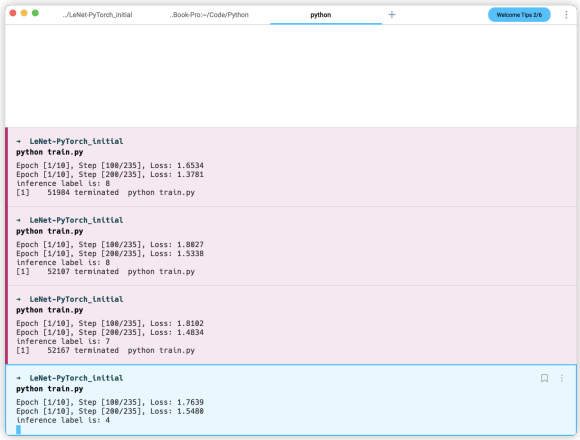


推断为7

我不知道原因 但是的确有这个输出

4.

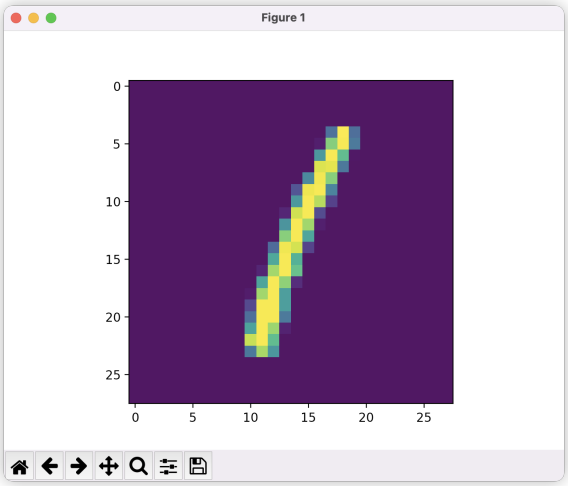


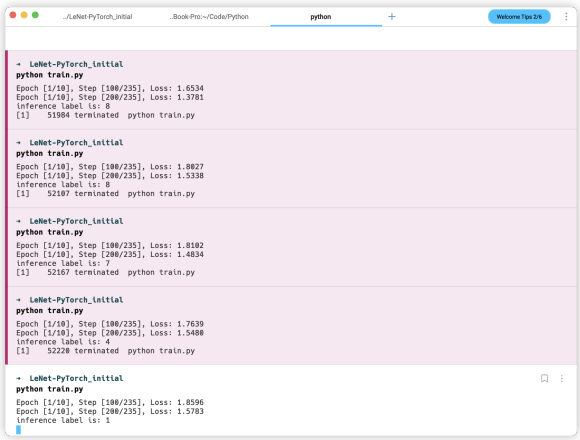


推断为4

7的首端有些粗，可能通过左上角部分pixel较密集，和一横一竖来识别一个4，导致误判

5.

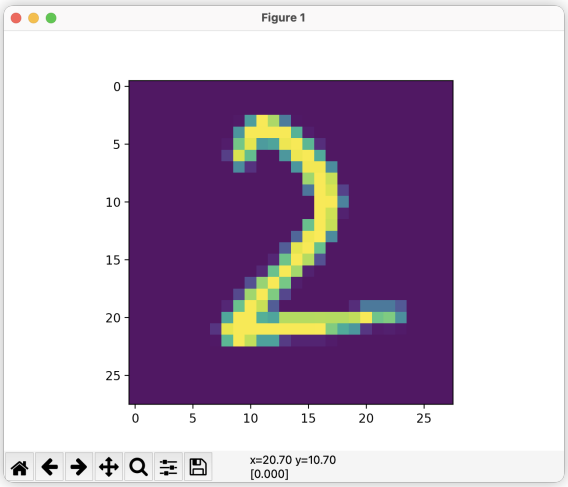


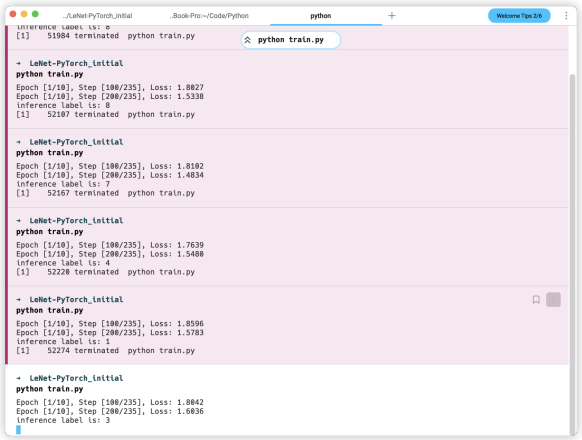


推断为1

同第三点，我觉得应该就是1吧

6.





推断为3

这张图在第一次出现过，不过这次被判断为3可能是2尾部的像素点较多（？），或者说优化器以及算法的原因，导致对于2的识别错误率更高。可能是由于2和8&3的相似程度的确较高，和3的区别在于最底下是否是横线，和8的区别在于是否有从左上到右下到连接，可能对于这点的识别做的不是很好。