人工智能实验报告 第9周

姓名:刘卓逸 学号:21307303

一.实验题目

hw7 机器学习：图像分类任务

二.实验内容

1.实验要求

在MNIST数据集上完成手写数字图片分类任务, 具体要求如下:

- 示例代码中已经给出从.pth文件加载数据集的代码(Tensor类型), 命名为(train_data, train_labels), (test_data, test_labels), 分别是训练图像, 训练标签, 测试图像和测试标签. 请基于这些Tensor完成训练任务以及测试任务.
- 用pytorch搭建卷积神经网络(在类中MyConvNet来实现), 在训练集上训练模型, 并在测试集上完成分类测试.
- 为了方便批改作业, MyConvNet的构造函数请不要使用任何形参.测试时至少用分类正确率来衡量性能(可以添加其他指标来衡量性能并在报告中呈现).
- 训练结束后, 务必使用torch.save()保存模型(即神经网络的参数等信息). 此次作业需要额外上传模型.模型的文件名格式为hw7_学号_姓名拼音.pth, 例如hw7_21000000_zhangsan.pth.
- 所有内容在同一个.py代码文件上实现.
- 作业提交时将.py代码文件和.pth模型文件提交到本科生实验hw7_code文件夹中, 实验报告提交到本科生实验hw7_report文件夹中. 请不要提交其他无关文件.

2.算法原理

（1）整体框架

```
定义好超参数
创建模型并初始化
对于每个epoch:
    将训练数据集拆分成训练集与验证集两部分
    用训练集进行训练:
        将数据输入模型得到预测结果
        计算预测结果与label之间的损失函数
        用反向传播计算梯度
        更新模型参数
    再用验证集求模型准确率:
        将数据输入模型得到预测结果
        计算平均损失函数与准确率
    保存较之前准确率有提升的模型
训练完成，用测试集测量模型实际准确率
```

（2）细节

1-神经网络框架

针对图像识分类问题，采用卷积神经网络。

思想是，用不同卷积核扫过整个图形可以提取出图像与卷积核相对应的特征。

如当卷积核为

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

卷积层输出在左0右1的竖直边缘为正，在左1右0的竖直边缘为负，在连续色块内部以及水平边缘就为0

2-损失函数

损失函数要用CrossEntropy,不能用平方误差。

在二分类问题中(多分类中one-hot向量计算损失时可看作若干个二分类问题)，若采用平方误差，在$f(x)=0$与$f(x)=1$时都会有梯度趋于0，导致离minimal很远的地方的梯度很小，训练就会很慢。

用crossentropy作为损失函数可避免上述问题

3-优化算法

采用Adam。Adam就是将RMSProp与Momentum结合

RMSProp是自适应动态学习率，当梯度小时调大学习率，梯度大时调小学习率。具体学习率如下

Momentum就是“惯性”思想，通过保留上一次的部分梯度，使得在梯度平衡点处也能更新参数，或者借着惯性冲出local minimal

参数θ_i迭代方式如下

$$\theta_i^{t+1} \leftarrow \theta_i^t - m_i^t * \eta/\sigma_i^t$$

m_i^t 来自Momentum思想，具体为 $m_i^t = \beta m_i^{t-1} + (1-\beta)g_i^t$

其中g_i^t是当前的梯度，β可以看做为“动量保留率”，βm_i^{t-1}就是保留的动量

η/σ_i^t 来自RMSProp, 期其中$\sigma_i^t = \sqrt{\alpha(\sigma_i^{t-1})^2 + (1-\alpha)|g_i^t|^2}$

结合考虑之前的梯度与当前梯度的值来动态调整学习率使梯度下降的幅度大致稳定。

3.代码展示

准备，import库

importPyTorch相关组件

```
_exp_name="hw7_21307303_liuzhuoyi"
#------ import ------
import torch
import torch.nn as nn
import torch.optim as optim
from torch.distributions import Categorical
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, random_split
```

封装Dataset

封装Dataset

```
#------ Dataset ------
class MyDataset(Dataset):
    def __init__(self,datas_dir="",label_dir=""):
        if datas_dir=="":
            return
        self.datas=torch.load(datas_dir)
```

```
        self.label=torch.load(label_dir)

    def __len__(self):
        return self.label.size()[0]

    def __getitem__(self,idx):
        return self.datas[idx],self.label[idx]
```

封装Dataloader

为了保证测试集的泛用性而使模型不一味拟合测试集，测试集将完全不参与训练

训练中有train与valid两个环节，考虑将训练集拆分成训练集与验证集两部分

用在验证集中的准确率来选择模型，在测试集中评估模型泛用性

```
#------ Dataloader ------
n_workers=8 #用于数据加载的子进程数

def get_dataloader(datas_dir,label_dir, batch_size, n_workers,valid_sept=0):
    dataset=MyDataset(datas_dir,label_dir)
    if valid_sept>0:
        #分割成trainset与validset
        validlen=int(valid_sept*len(dataset))
        lengths=[len(dataset)-validlen,validlen]
        trainset,validset=random_split(dataset,lengths)
        train_loader=DataLoader(
            trainset,
            batch_size=batch_size,
            shuffle=True, #训练时将训练数据顺序打乱
            num_workers=n_workers, #用于数据加载的子进程数
            drop_last=True, #最后一个batch可能不满batch_size,抛弃掉
            pin_memory=True, #存在固定内存，加速
        )
        valid_loader=DataLoader(
            validset,
            batch_size=batch_size,
            shuffle=False, #验证时就没必要打乱了
            num_workers=n_workers,
            drop_last=True,
            pin_memory=True,
        )
        return train_loader,valid_loader
    else:
        loader=DataLoader(
            dataset,
            batch_size=batch_size,
            shuffle=False,
            num_workers=n_workers,
            drop_last=True,
            pin_memory=True,
```

```
        )
        return loader
```

神经网络框架

先2层卷积层再三层全连接层。

其中一些层的具体用法如下 W

```
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # 输入有in_channels层，RGB就有3层，黑白就只有1层
        # 输出是out_channels层
        # 卷积核的边长是kernel_size
        # stride，卷积核每次移动的步长
        # 在图片四周补充padding个像素以保持图片大小
        # 输入为[in_channels,length,width]，那输出为[out_channels,1+
(length+2*padding-kernel_size)/stride]

        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # 池化，在每个边长为kernel_size的分区内选一个最大的

        # nn.BatchNorm2d(64)
        # 纠正不同特征层的平均值，加快训练速度
```

```
#------ Model ------
#确保每次调用卷积算法返回确定性输出，即默认算法
torch.backends.cudnn.deterministic = True
#固定网络结构的模型优化以提高效率，否则会花费时间在选择最合适算法上
torch.backends.cudnn.benchmark = False

class MyConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        #input [1,28,28]
        self.cnn=nn.Sequential(
            nn.Conv2d(1,16,3,1,1), #[16,28,28]
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2,2,0), #[16,14,14]

            nn.Conv2d(16,32,3,1,1), #[32,14,14]
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2,2,0), #[32,7,7]
        )
        self.fc=nn.Sequential(
            nn.Linear(32*7*7,8*7*7),
            nn.ReLU(),
            nn.Linear(8*7*7,4*7*7),
            nn.ReLU(),
```

```
            nn.Linear(4*7*7,10),
        )
    def forward(self, x):
        if len(x.size())==3:
            x=x.unsqueeze(0)
        out=self.cnn(x)
        out=out.view(out.size()[0],-1)
        out=self.fc(out)
        return torch.softmax(out,dim=-1)
```

设定超参数

```
#------ hyper parameter ------
batch_size=16
n_epochs=32
patience=8 #连续8个epochs模型没有改善就直接退出
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

测试

```
#------ test ------
def test(data, labels, net):
    num_data = data.shape[0]
    num_correct = 0
    for i in range(num_data):
        feature = data[i]
        prob = net(feature).detach()
        dist = Categorical(prob)
        label = dist.sample().item()
        true_label = labels[i].item()
        if label == true_label:
            num_correct += 1

    return num_correct / num_data
def Testing():
    net= MyConvNet()
    test_data = torch.load('data\\hw7\\test_data.pth')
    test_labels = torch.load('data\\hw7\\test_labels.pth')
    net.load_state_dict(torch.load('hw7_21307303_liuzhuoyi.pth'))
    net.eval()
    return test(test_data, test_labels, net)
```

主函数

注释已经很详细了

```python
#------ main ------
if __name__=="__main__":
    from tqdm.auto import tqdm #进度条可视化
    #模型实例化
    model=MyConvNet().to(device)
    #分类问题中,用cross-entropy来定义损失函数,用平方误差会导致离答案很远很近都梯度很小
    criterion = nn.CrossEntropyLoss()
    #Adam 动态学习率(加快收敛速度)+惯性梯度(避免local minimal)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.0003, weight_decay=1e-5)

    best_acc=0
    with open(f"./{_exp_name}_log.txt","w")as op:
        op.write("0 0 0 0 0\n")
    with open(f"./{_exp_name}_testlog.txt","w")as op:
        op.write("0 0\n")
    for epoch in range(n_epochs):
        #dataset/dataloader实例化
        train_loader,valid_loader=get_dataloader(
            datas_dir='data\\hw7\\train_data.pth',
            label_dir='data\\hw7\\train_labels.pth',
            batch_size=batch_size,
            n_workers=n_workers,
            valid_sept=0.1
        )
        test_loader=get_dataloader(
            datas_dir='data\\hw7\\test_data.pth',
            label_dir='data\\hw7\\test_labels.pth',
            batch_size=batch_size,
            n_workers=n_workers,
        )
        #----- Training ------
        #训练模式
        model.train()
        #训练记录
        train_loss=[]
        train_accs=[]
        for batch in tqdm(train_loader):
            #数据
            imgs,labels=batch
            #模型预测
            # imgs.size()=torch.Size([16, 1, 28, 28])
            logits=model(imgs.to(device))
            #计算损失
            loss=criterion(logits,labels.to(device))
            #梯度归零
            optimizer.zero_grad()
            #反向传播 计算梯度
            loss.backward()
            # Clip the gradient norms for stable training.
            grad_norm = nn.utils.clip_grad_norm_(model.parameters(), max_norm=10)
            #更新参数
            optimizer.step()
            #计算准确率并记录
            acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()
            train_loss.append(loss.item())
            train_accs.append(acc)
```

```
        train_loss = sum(train_loss) / len(train_loss)
        train_acc = sum(train_accs) / len(train_accs)
        # 输出信息
        print(f"[ Train | {epoch + 1:03d}/{n_epochs:03d} ] loss = {train_loss:.5f},
acc = {train_acc:.5f}")

        with open(f"./{_exp_name}_log.txt","a") as op:
            op.write(f"{epoch + 1:03d} {train_loss:.5f} {train_acc:.5f} ")
        # ------ Validation ------
        #无梯度信息模式
        model.eval()
        #信息记录
        valid_loss=[]
        valid_accs=[]
        for batch in tqdm(valid_loader):
            imgs, labels = batch
            # 验证时不需要计算梯度
            with torch.no_grad():
                logits = model(imgs.to(device))
            #求损失函数
            loss = criterion(logits, labels.to(device))
            # 求准确率
            acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()
            # 记录.
            valid_loss.append(loss.item())
            valid_accs.append(acc)
        # 求平均loss与平均准确率
        valid_loss = sum(valid_loss) / len(valid_loss)
        valid_acc = sum(valid_accs) / len(valid_accs)
        # 更新日志
        print(f"[ Valid | {epoch + 1:03d}/{n_epochs:03d} ] loss = {valid_loss:.5f},
acc = {valid_acc:.5f}",end="")
        if valid_acc > best_acc:
            print(" -> best")
        else:
            print("")

        with open(f"./{_exp_name}_log.txt","a") as op:
            op.write(f"{valid_loss:.5f} {valid_acc:.5f}\n")

        # 保存模型
        if valid_acc > best_acc:
            print(f"Best model found at epoch {epoch}, saving model")
            torch.save(model.state_dict(), f"{_exp_name}.pth") # only save best to
prevent output memory exceed error
            best_acc = valid_acc
            stale = 0
            with open(f"./{_exp_name}_test_log.txt","a") as op:\
                op.write(f"{epoch + 1:03d} {Testing():.5f}\n")
        else:
            stale += 1
            if stale > patience:
                print(f"No improvment {patience} consecutive epochs, early
stopping")
                break
```

```
    print("final test acc = ",Testing())
    # 如果已经训练完成，则直接读取网络参数．注意文件名改为自己的信息
```

4.创新点&优化

用Adam优化算法与CrossEntropy损失函数来优化训练过。

采用分割训练集的方式来训练与验证，将测试集留到最后评估，加强了评估结果的泛用性。

5.可改进点

因为人写字有可能写歪，可以对输入图像作随机的左右旋转或线性变换，实现数据增强。

然而考虑到输入图像只有28*28大小，且读进来就已经是tensor了，故没有做数据增强。

三.实验结果及分析

具体训练过程

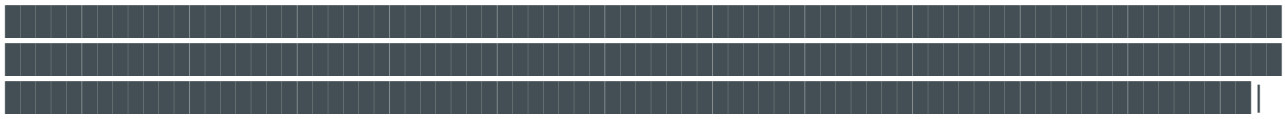
训练速度为 30s/epoch

在训练集上最高准确率为0.991

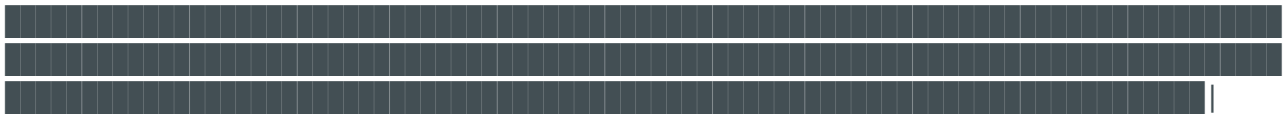
在验证集上最高准确率为0.9923，该模型在测试集上的表现准确率为0.9882

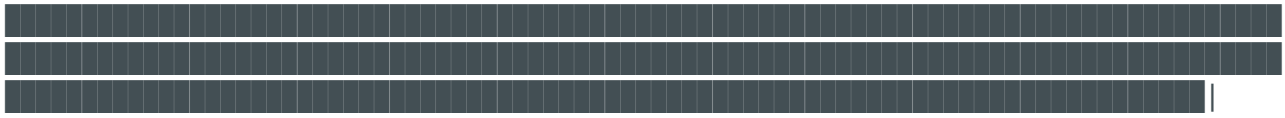
```
(base) PS E:\File\Documents\Lesson\2023AI> python
./Reports/hw7/hw7_21307303_liuzhuoyi.py
100%|
                                                                                
                                                                                
                                                                                |
3375/3375 [00:20<00:00, 163.98it/s]
[ Train | 001/032 ] loss = 1.52057, acc = 0.94613
100%|
                                                                                
                                                                                
                                                                                |
375/375 [00:08<00:00, 43.70it/s]
[ Valid | 001/032 ] loss = 1.49634, acc = 0.96533 -> best
```

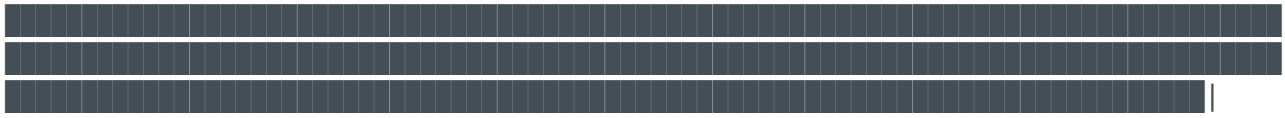
```
Best model found at epoch 0, saving model
100%|
|
3375/3375 [00:19<00:00, 170.56it/s]
[ Train | 002/032 ] loss = 1.48606, acc = 0.97576
100%|
|
375/375 [00:09<00:00, 39.36it/s]
[ Valid | 002/032 ] loss = 1.47761, acc = 0.98417 -> best
Best model found at epoch 1, saving model
100%|
|
3375/3375 [00:19<00:00, 171.66it/s]
[ Train | 003/032 ] loss = 1.48151, acc = 0.97987
100%|
|
375/375 [00:08<00:00, 41.74it/s]
[ Valid | 003/032 ] loss = 1.48714, acc = 0.97467
100%|
|
3375/3375 [00:19<00:00, 177.16it/s]
[ Train | 004/032 ] loss = 1.47886, acc = 0.98241
100%|
|
375/375 [00:08<00:00, 44.14it/s]
[ Valid | 004/032 ] loss = 1.47511, acc = 0.98533 -> best
Best model found at epoch 3, saving model
100%|
|
3375/3375 [00:18<00:00, 179.11it/s]
[ Train | 005/032 ] loss = 1.47772, acc = 0.98352
100%|
|
375/375 [00:08<00:00, 44.14it/s]
[ Valid | 005/032 ] loss = 1.47590, acc = 0.98517
100%|
|
3375/3375 [00:19<00:00, 177.39it/s]
[ Train | 006/032 ] loss = 1.47650, acc = 0.98467
```

100%|

375/375 [00:08<00:00, 43.22it/s]
[Valid | 006/032] loss = 1.47502, acc = 0.98617 -> best
Best model found at epoch 5, saving model
100%|

3375/3375 [00:19<00:00, 171.58it/s]
[Train | 007/032] loss = 1.47591, acc = 0.98524
100%|

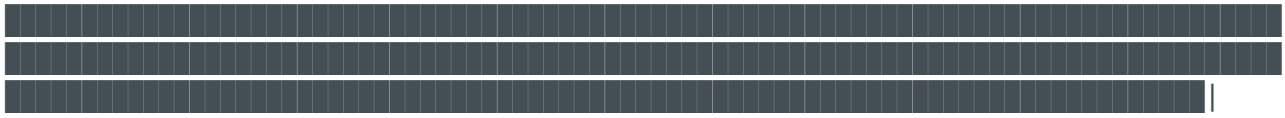
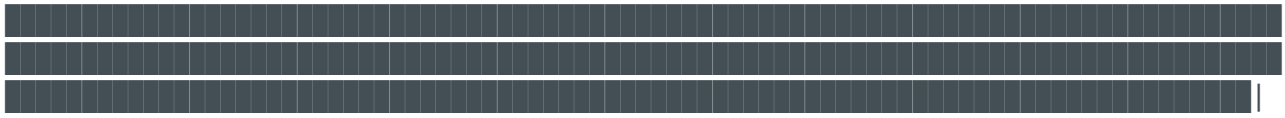
375/375 [00:08<00:00, 44.30it/s]
[Valid | 007/032] loss = 1.47793, acc = 0.98333
100%|

3375/3375 [00:19<00:00, 176.20it/s]
[Train | 008/032] loss = 1.47548, acc = 0.98569
100%|

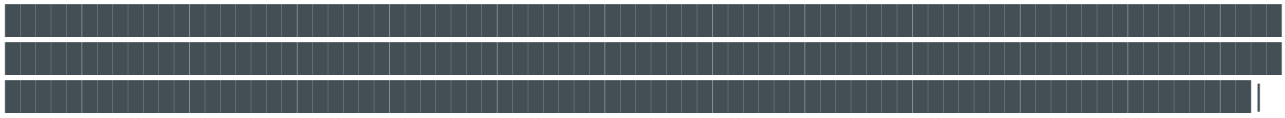
375/375 [00:08<00:00, 43.10it/s]
[Valid | 008/032] loss = 1.47095, acc = 0.99050 -> best
Best model found at epoch 7, saving model
100%|

3375/3375 [00:19<00:00, 177.53it/s]
[Train | 009/032] loss = 1.47377, acc = 0.98741
100%|

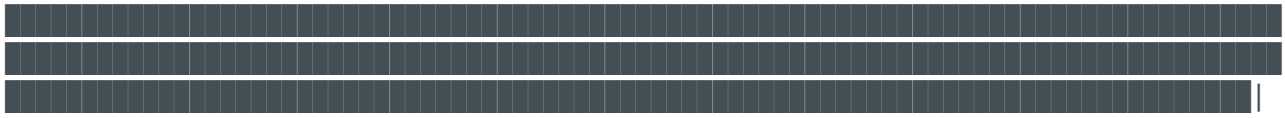
375/375 [00:08<00:00, 42.28it/s]
[Valid | 009/032] loss = 1.47342, acc = 0.98750
100%|

3375/3375 [00:19<00:00, 173.88it/s]
[Train | 010/032] loss = 1.47358, acc = 0.98756
100%|

375/375 [00:08<00:00, 43.09it/s]
[Valid | 010/032] loss = 1.47086, acc = 0.99033
100%|


3375/3375 [00:18<00:00, 180.30it/s]
[Train | 011/032] loss = 1.47392, acc = 0.98724
100%|

375/375 [00:08<00:00, 43.91it/s]
[Valid | 011/032] loss = 1.47297, acc = 0.98783
100%|

3375/3375 [00:18<00:00, 178.81it/s]
[Train | 012/032] loss = 1.47243, acc = 0.98867
100%|

375/375 [00:08<00:00, 43.41it/s]
[Valid | 012/032] loss = 1.47703, acc = 0.98400
100%|

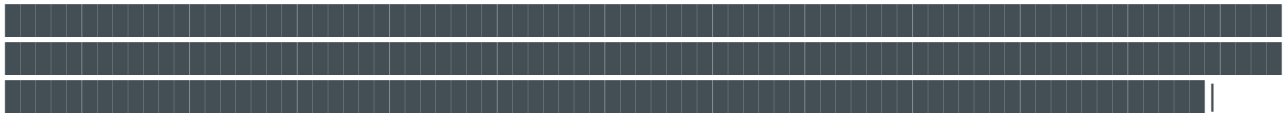
3375/3375 [00:19<00:00, 177.28it/s]
[Train | 013/032] loss = 1.47294, acc = 0.98815
100%|

375/375 [00:08<00:00, 43.53it/s]
[Valid | 013/032] loss = 1.47093, acc = 0.99000
100%|

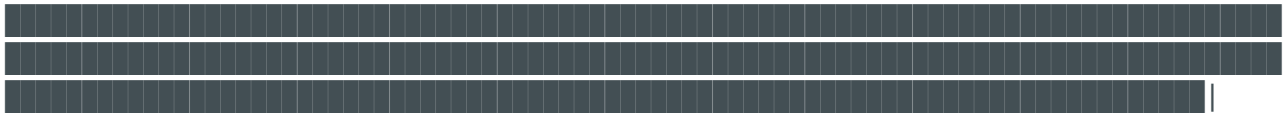
3375/3375 [00:18<00:00, 180.98it/s]
[Train | 014/032] loss = 1.47201, acc = 0.98907
100%|

375/375 [00:08<00:00, 43.30it/s]
[Valid | 014/032] loss = 1.47532, acc = 0.98517
100%|

3375/3375 [00:19<00:00, 173.54it/s]
[Train | 015/032] loss = 1.47306, acc = 0.98809
100%|


375/375 [00:08<00:00, 42.34it/s]
[Valid | 015/032] loss = 1.47119, acc = 0.98967
100%|

3375/3375 [00:19<00:00, 176.51it/s]
[Train | 016/032] loss = 1.47108, acc = 0.99015
100%|

375/375 [00:08<00:00, 43.34it/s]
[Valid | 016/032] loss = 1.46710, acc = 0.99400 -> best
Best model found at epoch 15, saving model
100%|

3375/3375 [00:19<00:00, 175.54it/s]
[Train | 017/032] loss = 1.47076, acc = 0.99028
100%|

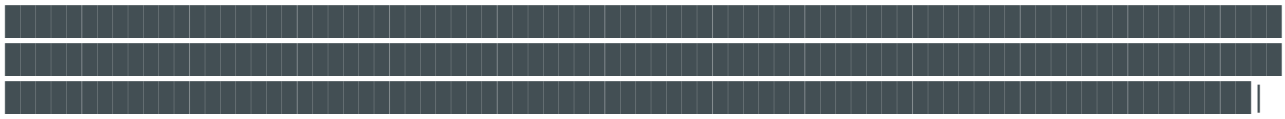
375/375 [00:08<00:00, 44.55it/s]
[Valid | 017/032] loss = 1.47508, acc = 0.98617
100%|

3375/3375 [00:18<00:00, 178.46it/s]
[Train | 018/032] loss = 1.47089, acc = 0.99024
100%|

375/375 [00:08<00:00, 44.44it/s]
[Valid | 018/032] loss = 1.47034, acc = 0.99050
100%|

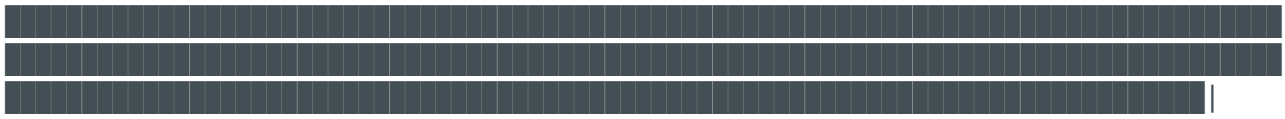
3375/3375 [00:18<00:00, 180.46it/s]
[Train | 019/032] loss = 1.47123, acc = 0.99000
100%|

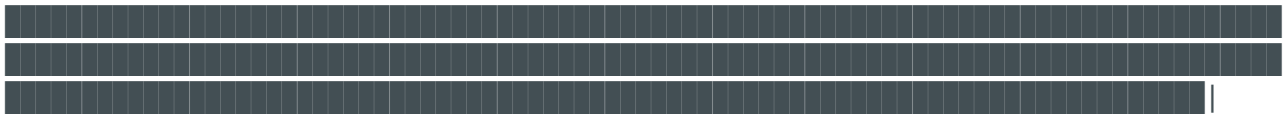
375/375 [00:08<00:00, 44.53it/s]
[Valid | 019/032] loss = 1.46834, acc = 0.99267
100%|

3375/3375 [00:19<00:00, 176.20it/s]
[Train | 020/032] loss = 1.47098, acc = 0.99015

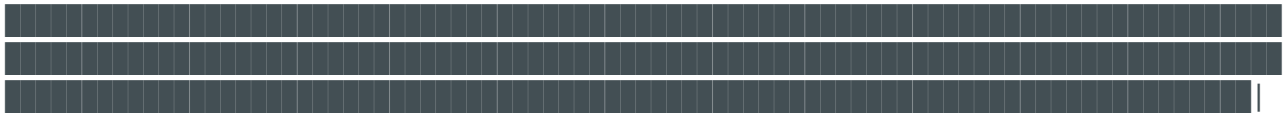
100%|

375/375 [00:08<00:00, 44.29it/s]
[Valid | 020/032] loss = 1.47224, acc = 0.98933
100%|

3375/3375 [00:19<00:00, 173.17it/s]
[Train | 021/032] loss = 1.47030, acc = 0.99083
100%|

375/375 [00:08<00:00, 44.41it/s]
[Valid | 021/032] loss = 1.47114, acc = 0.99000
100%|

3375/3375 [00:19<00:00, 176.41it/s]
[Train | 022/032] loss = 1.46988, acc = 0.99131
100%|

375/375 [00:08<00:00, 43.33it/s]
[Valid | 022/032] loss = 1.47196, acc = 0.98933
100%|

3375/3375 [00:18<00:00, 179.10it/s]
[Train | 023/032] loss = 1.47049, acc = 0.99069
100%|

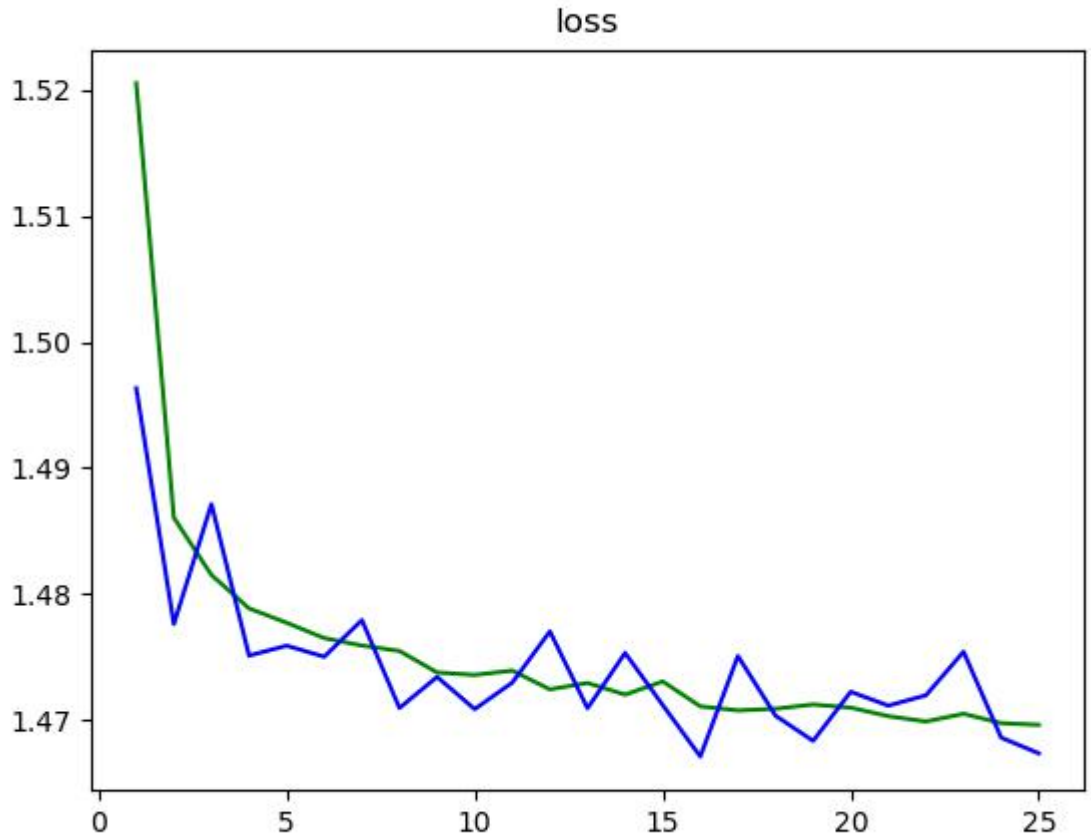
375/375 [00:08<00:00, 44.44it/s]
[Valid | 023/032] loss = 1.47542, acc = 0.98567
100%|

3375/3375 [00:18<00:00, 179.22it/s]
[Train | 024/032] loss = 1.46975, acc = 0.99143
100%|

375/375 [00:08<00:00, 44.26it/s]
[Valid | 024/032] loss = 1.46860, acc = 0.99267
100%|

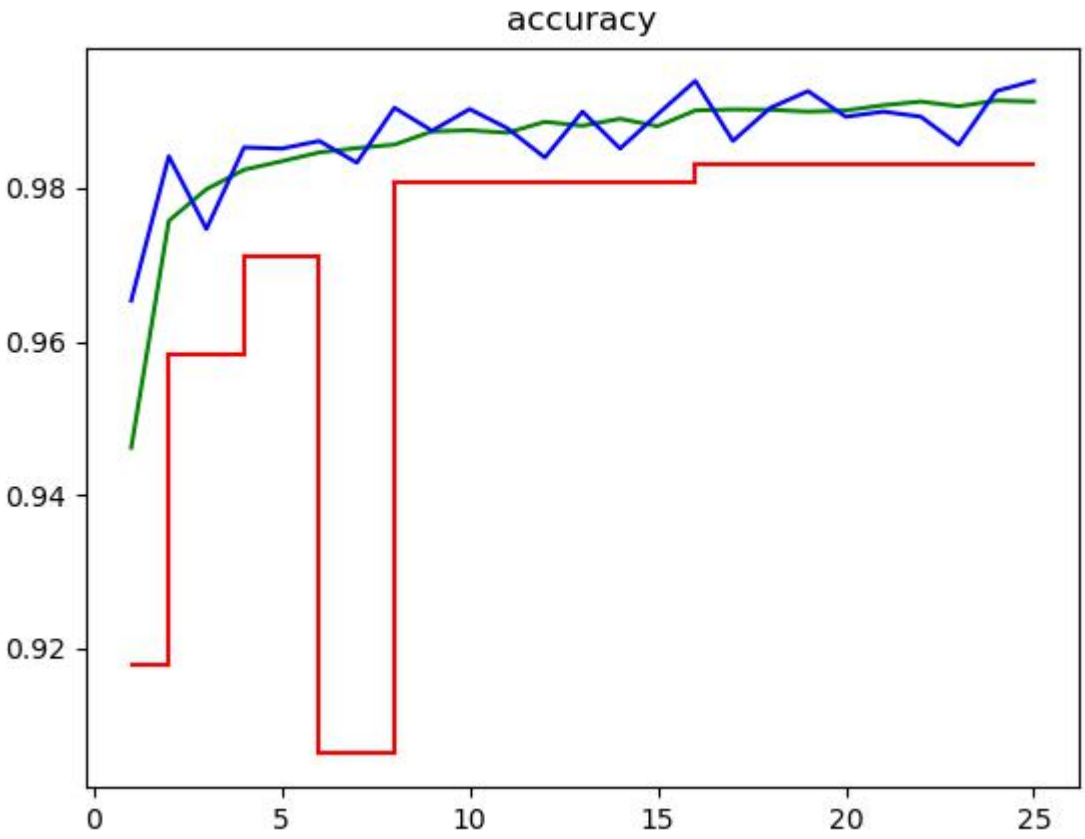

```
100%|
3375/3375 [00:18<00:00, 178.32it/s]
[ Train | 025/032 ] loss = 1.46961, acc = 0.99131
100%|
375/375 [00:08<00:00, 44.51it/s]
[ Valid | 025/032 ] loss = 1.46735, acc = 0.99400
No improvment 8 consecutive epochs, early stopping
final test acc =  0.9832
```

训练过程可视化

损失函数(绿线train 蓝线valid)



准确率(绿线train 蓝线valid 红线test)



测试准确率有可能会突然变差

评测指标展示及分析

测试集完全没有参与训练过程的前提下，对测试集的准确率高达0.9832，该模型很好的完成了手写数字分类的任务

四.实验总结

通过这次实验，我学会了如何构建机器学习的代码，对Dataset、Dataloader、神经网络的搭建有了深刻的体会。我深入了解了卷积神经网络的原理与使用方法，也锻炼了对PyTorch的使用。

参考资料

李宏毅2021/2022春机器学习课程 https://www.bilibili.com/video/BV1Wv411h7kN/

李宏毅2023春机器学习课程 https://www.bilibili.com/video/BV1TD4y137mP/