中山大学计算机学院人工智能本科生实验报告（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	专业（方向）	学号	姓名
2班	计算机科学与技术	21307174	刘俊杰

一、实验题目

Week 15 卷积神经网络

二、实验内容

实验内容：

- 用卷积神经网络(CNN)实现手写数字识别，数据集为MNIST，网络结构自行设计，CNN原理可看理论课件ML23-4 P30-53

结果要求与展示：

- 测试集准确率（Acc）90%以上
-
- 画出训练过程中，Loss和Acc的变化曲线图

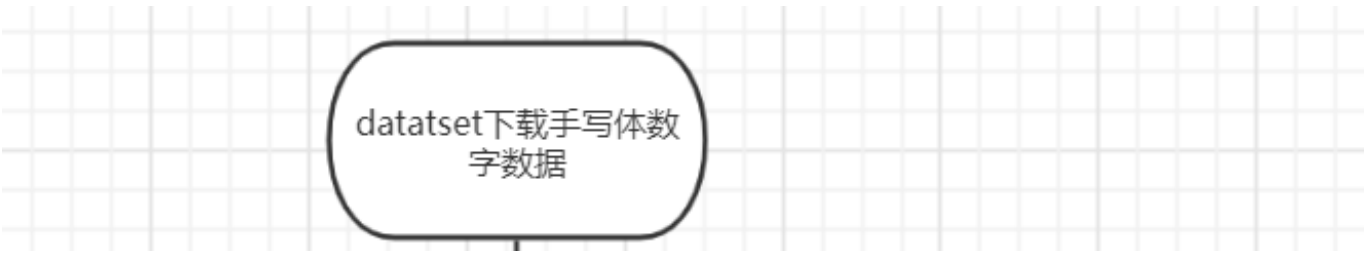
1. 算法原理

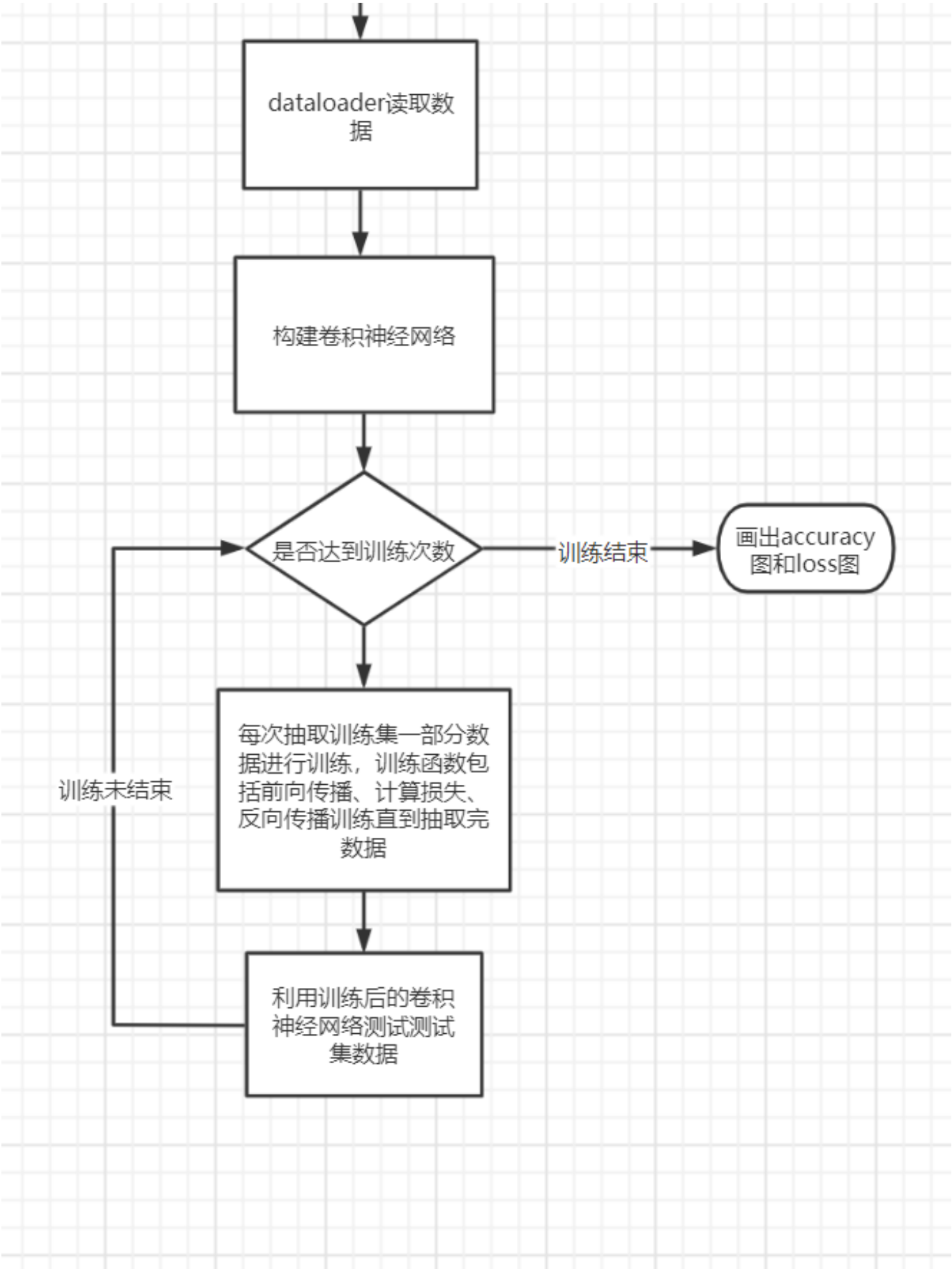
卷积神经网络与普通神经网络的区别在于，卷积神经网络包含了一个由卷积层和子采样层（池化层）构成的特征抽取器。在卷积神经网络的卷积层中，一个神经元只与部分邻层神经元连接。在CNN的一个卷积层中，通常包含若干个特征图(featureMap)，每个特征图由一些矩形排列的的神经元组成，同一特征图的神经元共享权值，这里共享的权值就是卷积核。卷积核一般以随机小数矩阵的形式初始化，在网络的训练过程中卷积核将学习得到合理的权值。共享权值（卷积核）带来的直接好处是减少网络各层之间的连接，同时又降低了过拟合的风险。子采样也叫做池化（pooling），通常有均值子采样（mean pooling）和最大值子采样（max pooling）两种形式。子采样可以看作一种特殊的卷积过程。卷积和子采样大大简化了模型复杂度，减少了模型的参数。

本实验中读入手写体数字的图片，将图片转换为tensor输入到卷积神经网络中，本实验构造的卷积神经网络结构为:

卷积层->ReLU->最大池化->压缩张量->线性层->ReLu->线性层->log_softmax

2.流程图





3.关键代码展示

(1)搭建卷积神经网络

继承nn.Module类，重写初始化函数和前向传播函数，搭建的卷积神经网络结构为:卷积层->ReLU->最大池化->压缩张量->线性层->ReLu->线性层->log_softmax。 先通过卷积和池化减少网络各层之间的连接，同时又降低了过拟合的风险，简化了模型复杂度，减少了模型的参数，再通过压缩张量输入到两个线性层中，最后通过log_softmax将10个分类的输出转换为预测概率。

```
class CNN(nn.Module):#卷积神经网络
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1,10,5)#卷积层1:输入通道为1，输出通道为10，卷积核大小为5
        self.linear1 = nn.Linear(1440,100)#线性层1:输入通道为1440，输出通道为100
        self.linear2 = nn.Linear(100, 10)#线性层2:输入通道为100，输出通道为10(10分类)
    def forward(self,input):
        input_size = input.size(0)#获取输入的大小，这里也就是batch_size
        output = self.conv1(input)#batch_size*10*24*24
        output = F.relu(output)#激活函数，输出batch_size*10*24*24
        output = F.max_pool2d(output,2,2) #池化，输出batch_size*10*12*12
        output = output.view(input_size,-1)#调整output大小为batch_size*1440
        output = self.linear1(output)#输出为batch_size*100
        output = F.relu(output)#激活函数，输出为batch_size*100
        output = self.linear2(output)#输出为batch_size*10
        #在深度学习中，我们需要将神经网络的输出转化为预测结果，而由于输出值并非总是代表着概率，因此我们需要使用激活函数将其转化为概率值。输出值通过lg sofmax运算转化为概率值
        output = F.log_softmax(output, dim=1) #激活函数，输出为batch_size*10
        return output
```

(2)训练神经网络

每次从训练集中抽取一部分样本进行训练，首先调用model.train()启用 batch normalization 和 dropout ，再清空上一轮梯度，通过前向传播计算输出，计算损失并进行反向传播

```
def train(model,train_loader,optimizer,epoch):#神经网络训练函数
    model.train()#启用 batch normalization 和 dropout
    for data,target in  train_loader:#data数据，target预测的目标
        optimizer.zero_grad()#清空上一轮的梯度
        output= model(data)#前向传播
        loss = F.nll_loss(output, target)#计算损失,10分类的的输出都是负数，nll_loss取target对应的输出取正求均值，也就是损失在预测是target的概率越高时越低
        loss.backward()#反向传播
        optimizer.step()#执行优化步骤，通过梯度下降法来更新参数的值
```

(3)卷积神经网络测试

调用model.eval()不启用atch normalization 和 dropout，加快运行时间和 减少占用空间,每次取测试机一批数据预测，计算该批测试的损失和正确个数

```
def test(model,test_loader):#测试函数
    model.eval()#不启用 batch normalization 和 dropout
    test_loss = 0#本轮测试的损失
    correct = 0#测试正确的个数
    with torch.no_grad():#不自动求导，大大节约了显存或者说内存
        for data,target in test_loader:
            output = model(data)#输出为10中数字的预测概率
            test_loss += F.nll_loss(output, target, reduction='sum').item() #将本
轮的损失相加
            pred = output.max(1, keepdim=True)[1] #取概率最大的预测狮子
            correct += pred.eq(target.view_as(pred)).sum().item()#取本轮预测正确的
个数
    test_loss /= len(test_loader.dataset)#求平均损失
    print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
    accuracy = correct / len(test_loader.dataset)
    return accuracy,test_loss
```

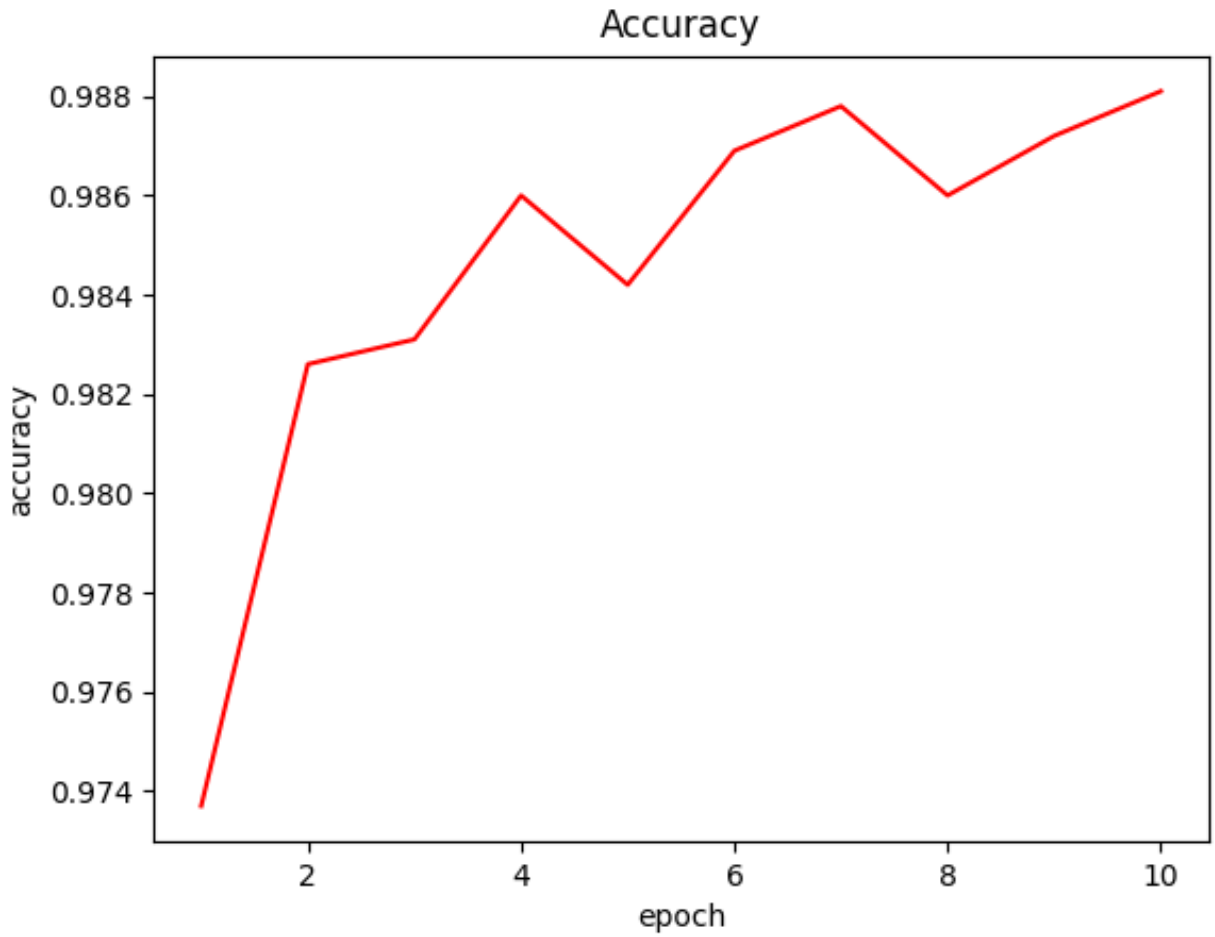
4.创新点&优化

无

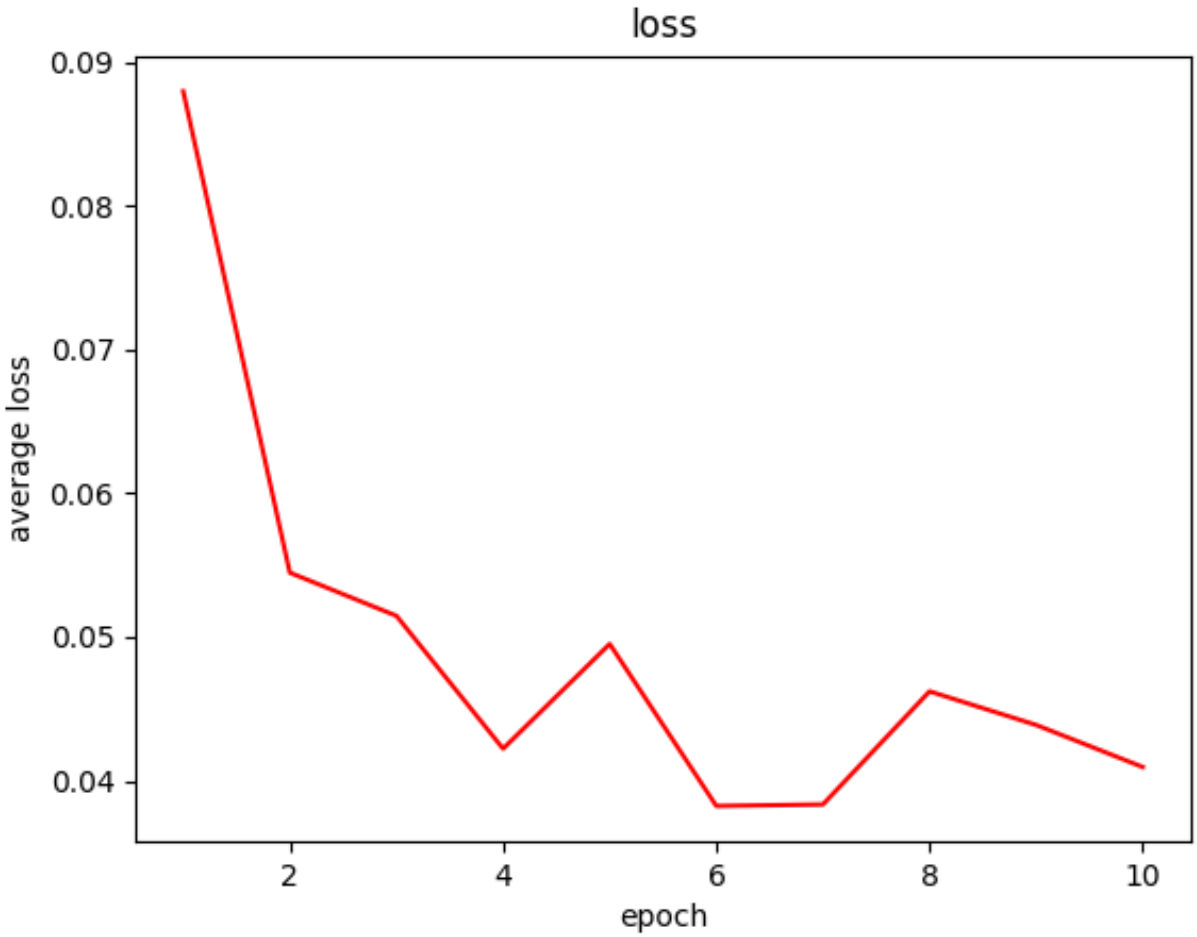
三、实验结果及分析

1. 实验结果展示示例(实验结果放入result文件夹中)

10次训练后测试的准确率：

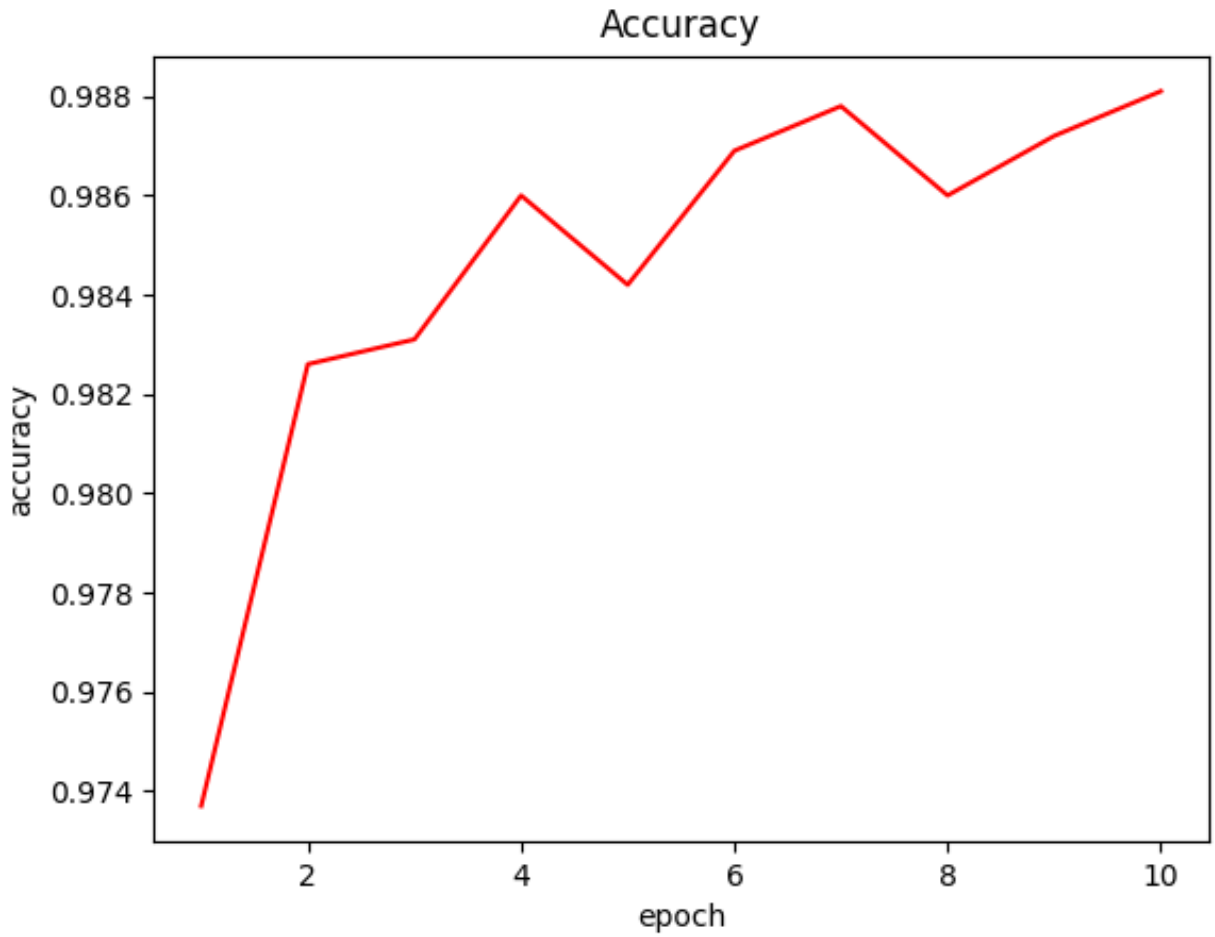


10次训练后测试的平均损失:

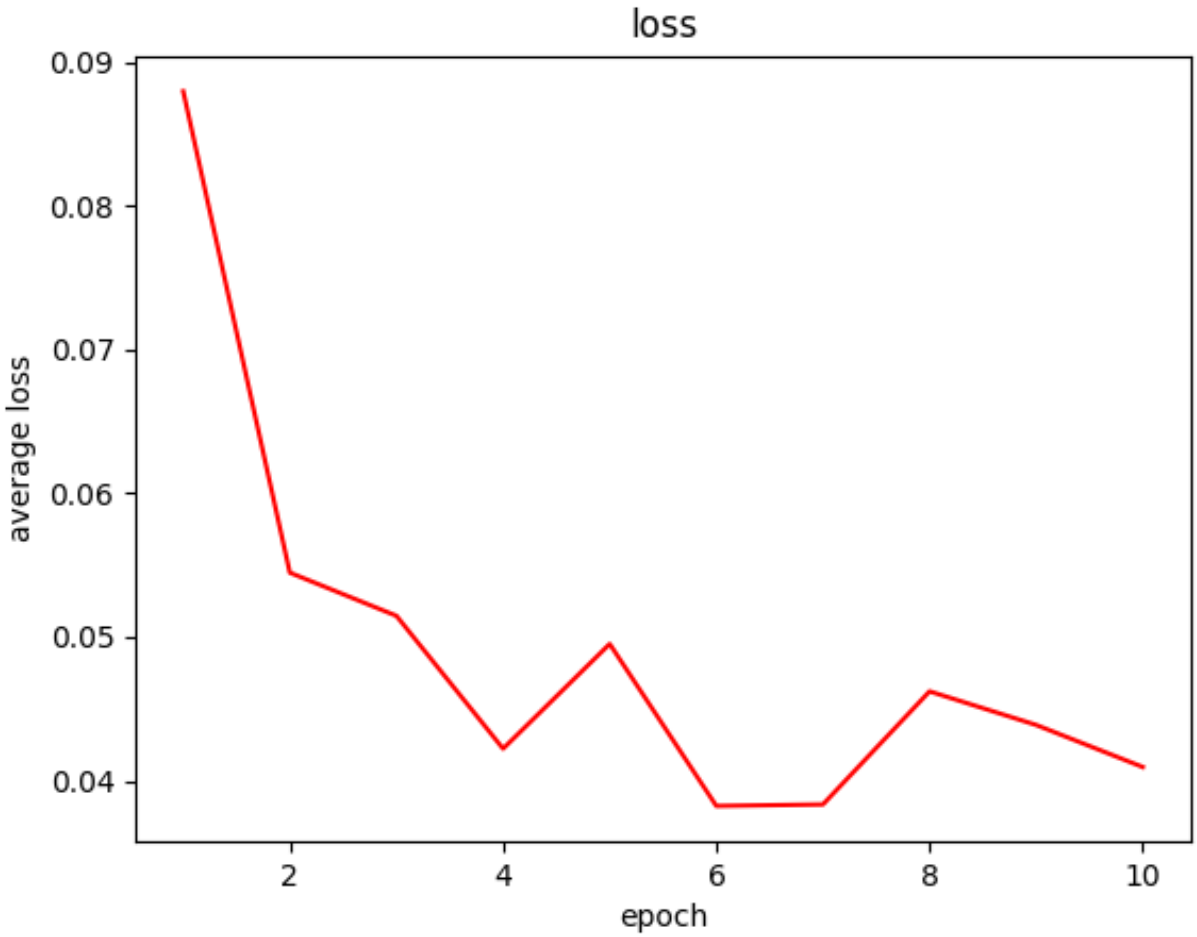


2、评价指标展示及分析

10次训练后测试的准确率：



10次训练后测试的平均损失:



分析:

(1)可以看到第一次训练后的准确率就达到了97%,在第二次训练后测试的准确率达到98%，后面训练的准确率也有提升但比较缓慢。这是因为在第一次训练中分多批次训练已经将神经网络优化得很好了，在后面的训练中对神经网络的优化影响小了。

(2)平均损失在第一次训练后就已经很小了 ，在第二次训练后就有较大下滑后，损失就变化不多了，原因和上一点一致。

(3)可见在第一轮训练神经网络就达到较好的效果，这是因为在第一次训练的多批反向传播中，神经网络参数优化达到一定程度后，后续训练对神经网络的优化就提升不大了。

四、参考资料

MINIST数据集 https://blog.csdn.net/Iron802/article/details/121826385

MNIST数据集手写数字识别（CNN） https://blog.csdn.net/sikh_0529/article/details/126901302