#### 作业4：CNN网络设计

班级：数科2004 姓名：刘一 学号： 2020005923

**一、作业要求**

设计一个对MNIST处理的CNN网络，网络结构和参数要求如下表，其中每一次MaxPooling后都添加系数为0.25的Dropout。

**二、运行环境：**

操作系统：

Windows11

软件环境：

ppython 3.7.7

conda 4.14.0

torch 1.12.0

torchvision 0.13.0

tensorboard 2.11.2

**三、操作步骤**

1. 导入所需要的库

|  |
| --- |
| import torch  from torchvision import datasets, transforms  from torch.utils.data import DataLoader  from torch.autograd import Variable  from torch import nn, optim  from torch import nn |

1. 定义超参数

|  |
| --- |
| # 定义超参数  # 样本量大小  batch\_size = 1  # 学习率  learning\_rate = 0.05  # 迭代的轮数  num\_epoches = 50 |

3. 定义CNN模型

|  |
| --- |
| class CNN(nn.Module):      def \_\_init\_\_(self):          super(CNN, self).\_\_init\_\_()          # 二维卷积层，设置输入通道与输出通道，核大小，填充行数与列数          self.conv1 = nn.Conv2d(1, 16, kernel\_size=5, stride=1,padding=2)          self.conv2 = nn.Conv2d(16, 32, kernel\_size=2, stride=1,padding=1)          self.conv3 = nn.Conv2d(32, 64, kernel\_size=5, stride=1,padding=2)          self.conv4 = nn.Conv2d(64, 128, kernel\_size=5, stride=1, padding=2)          '''self.conv1 = nn.Conv2d(1, 6, kernel\_size=5, padding=2)          self.conv2 = nn.Conv2d(6, 16, kernel\_size=5)          self.conv3 = nn.Conv2d(16, 120, kernel\_size=5)          '''          # 采用最大池化          self.MP = nn.MaxPool2d(kernel\_size=(2, 2), stride=2)          # ReLU函数的定义          self.relu = nn.ReLU()          # 对全连接层输入与输出的张量的大小的设置          self.fc1 = nn.Linear(1, 128)          self.fc2 = nn.Linear(128, 10)          # 运用softmax函数          self.logsoftmax = nn.LogSoftmax()          self.dropout=nn.Dropout(p=0.25)      # 正向传递      def forward(self, x):          in\_size = x.size(0)          # 这里对输入进行逐层处理          out = self.relu(self.conv1(x))          print("第一层卷积后的形状:",out.shape)          out = self.MP(out)          print("第一层池化后的形状:",out.shape)          out=self.dropout(out)          out = self.conv2(out)          print("第二层卷积后的形状:", out.shape)          out = self.MP(self.relu(out))          print("第二层池化后的形状:", out.shape)          out = self.dropout(out)          out=self.conv3(out)          print("第三层卷积后的形状:", out.shape)          out = self.MP(self.relu(out))          print("第三层池化后的形状:", out.shape)          out = self.dropout(out)          out = self.conv4(out)          print("第四层卷积后的形状:", out.shape)          out = self.MP(self.relu(out))          print("第四层池化后的形状:", out.shape)          out = self.dropout(out)          # 返回一个输出的视图，该视图的维度为输入的大小\*自动计算的大小          out = out.view(in\_size, -1)          out = self.relu(out)          out = self.fc2(out)          print("最终的输出形状:",out.shape)          return self.logsoftmax(out) |

4.加载数据，传入模型，计算损失，反向传播，迭代更新参数

|  |
| --- |
| # 数据预处理。transforms.ToTensor()将图片转换成PyTorch中处理的对象Tensor,并且进行标准化（数据在0~1之间）  # transforms.Normalize()做归一化。它进行了减均值，再除以标准差。两个参数分别是均值和标准差  # transforms.Compose()函数则是将各种预处理的操作组合到了一起  data\_tf = transforms.Compose(      [transforms.ToTensor(),       transforms.Normalize([0.5], [0.5])])  # 加载指定的MNIST数据集  train\_dataset = datasets.MNIST(      root='./data', train=True, transform=data\_tf, download=True)  # 数据加载器  train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  # 选择CNN模型  model = CNN()  # 将交叉熵函数作为损失函数  criterion = nn.CrossEntropyLoss()  # 使用随机梯度下降法  optimizer = optim.SGD(model.parameters(), lr=learning\_rate)  # 训练模型  epoch = 0  # 利用数据加载器进行迭代  for data in train\_loader:      data[0]=data[0].reshape(1,28, 28)      img, label = data      # 将img转化为variable类型，使之可以进行反向传播      img = Variable(img)      if torch.cuda.is\_available():          img = img.cuda()          label = label.cuda()      else:          img = Variable(img)          label = Variable(label)      out = model(img)      # 计算损失      loss = criterion(out, label)      print\_loss = loss.data.item()      # 清除历史梯度，因为梯度在不断累积，而每次迭代不需要继续这种累积      optimizer.zero\_grad()      # 从损失开始进行反向传播      loss.backward()      # 进行一次参数的更新      optimizer.step()      # 进而迭代轮数增加1      epoch += 1      if epoch % 1== 0:          print('epoch: {}, loss: {:.4}'.format(epoch, loss.data.item()))          print(">>>"+"-"\*20)      if epoch==num\_epoches:          break |

**四、结果分析**

|  |
| --- |
| 第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 1, loss: 2.507  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 2, loss: 2.384  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 3, loss: 2.248  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 4, loss: 2.183  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 5, loss: 2.239  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 6, loss: 1.99  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 7, loss: 1.822  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 8, loss: 2.86  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 9, loss: 2.76  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 10, loss: 2.502  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 11, loss: 2.108  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 12, loss: 2.202  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 13, loss: 1.962  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 14, loss: 1.733  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 15, loss: 3.021  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 16, loss: 2.594  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 17, loss: 1.852  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 18, loss: 2.254  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 19, loss: 2.496  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 20, loss: 1.729  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 21, loss: 2.266  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 22, loss: 2.096  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 23, loss: 2.346  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 24, loss: 1.715  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 25, loss: 3.081  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 26, loss: 2.688  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 27, loss: 2.34  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 28, loss: 1.886  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 29, loss: 2.27  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 30, loss: 2.237  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 31, loss: 1.81  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 32, loss: 1.382  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 33, loss: 3.464  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 34, loss: 1.456  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 35, loss: 2.885  >>>--------------------  第一层卷积后的形状: torch.Size([16, 28, 28])  第一层池化后的形状: torch.Size([16, 14, 14])  第二层卷积后的形状: torch.Size([32, 15, 15])  第二层池化后的形状: torch.Size([32, 7, 7])  第三层卷积后的形状: torch.Size([64, 7, 7])  第三层池化后的形状: torch.Size([64, 3, 3])  第四层卷积后的形状: torch.Size([128, 3, 3])  第四层池化后的形状: torch.Size([128, 1, 1])  最终的输出形状: torch.Size([1, 10])  epoch: 36, loss: 2.257  >>>-------------------- |

以上是训练50轮之后的效果，展示了每一次的数据经过不同层之后的形状，以及每一轮的损失，可以观察到，随着训练次数的增加，损失总体不断减小。