### 实验4.2 手写数字集分类实验

1. **实验目的**

利用给定的mnist数据集，训练一个网络对各个图片进行识别，采用测试集测试分类器的性能。

1. **实验内容**

1、数据下载**：**http://yann.lecun.com/exdb/mnist/

2、数据描述：Mnist数据由0-9共10类手写数字组成，训练集包括60000张图片，测试集包括10000张图片。

1. 实验环境：
   1. 编程语言：Python3。
   2. 编程框架：PyTorch
   3. 编辑器：PyCharm
   4. 模型训练环境：GPU

4、实验任务：

（1）基本任务：

a. 使用卷积神经网络或全连接神经网络对手写数字进行分类实现，准确率达到80%以上（80分）

b. 对典型错分图片示例（10分）

c. 绘制训练集和测试集的准确率、损失曲线（10分）

（2）扩展任务：

判断模型是否出现过拟合，若发生过拟合如何处理，详细写出处理过程，如果有实验验证结果，可以描述实验结果。（附加20分）

1. **实验过程**

1、数据处理的描述

* 数据处理
* 数据集：MNIST 数据集，包含 0-9 的手写数字图片。
* 训练集：60000 张图片。
* 测试集：10000 张图片。
* 处理方法：
* 数据加载：使用 PyTorch 的 torchvision.datasets.MNIST 和torch.utils.data.DataLoader加载数据。
* 数据预处理：将图片转换为张量并进行归一化，使用均值 0.14 和标准差 0.3。

2、模型结构的设计（画出网络层示意图：可采用表格或者是图示）

* 网络类型：卷积神经网络（CNN）。
* 结构组成：
* 卷积层：两个卷积层（Conv2d），分别有 1 和 10 个输入通道，10 和 20 个输出通道，卷积核大小均为 5。
* 激活层：ReLU 函数。
* 池化层：最大池化（MaxPool2d），池化窗口大小为 2。
* Dropout 层：随机丢弃一部分神经元，减少过拟合。
* 全连接层：两个线性层（Linear），将平坦化后的特征映射到 50 和 10 个输出节点。

CNN网络示意图表

| 层级 | 类型 | 输入通道 | 输出通道 | 卷积核/池化窗口 | 尺寸变化 | 激活函数 | Dropout |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 输入层 | - | - | - | 1x28x28 | - | - |
| 2 | 卷积层 | 1 | 10 | 5x5 | 10x24x24 | ReLU | - |
| 3 | 池化层 | - | - | 2x2 | 10x12x12 | - | - |
| 4 | 卷积层 | 10 | 20 | 5x5 | 20x8x8 | ReLU | 是 |
| 5 | 池化层 | - | - | 2x2 | 20x4x4 | - | - |
| 6 | 全连接层 | - | - | - | - | ReLU | - |
| 7 | Dropout | - | - | - | - | - | - |
| 8 | 全连接层 | - | - | - | - | - | - |
| 9 | 输出层 | - | - | - | - | LogSoftmax | - |

**说明**:

* 输入层接收 1x28x28 大小的图像（1 个颜色通道，28x28 像素）。
* 第一个和第二个卷积层分别有 1 和 10 个输入通道，10 和 20 个输出通道，卷积核大小都是 5x5。第二个卷积层后面跟着一个 Dropout 层。
* 池化层使用 2x2 的最大池化窗口。
* 第一个全连接层将展平后的特征映射到 50 个节点，第二个全连接层将 50 个特征映射到 10 个输出节点（代表 10 个数字类别）。
* 输出层使用Log-Softmax 函数生成 10 个类别的概率分布。

3、评价指标的计算公式

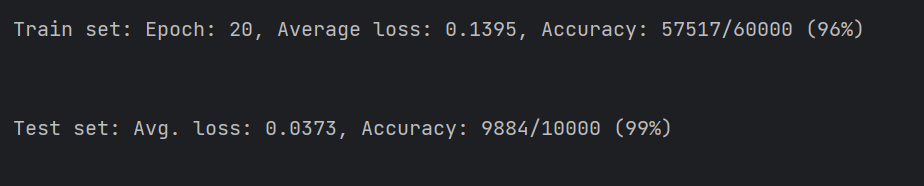
|  |  |  |
| --- | --- | --- |
| 真实预测 | 正样本 | 负样本 |
| 正样本 | TP | FP |
| 负样本 | FN | TN |

公式：Accuracy = (TP+TN)/(TP+FP+TN+FN)

4、扩展任务

* 通过观察训练和测试的损失及准确率曲线，分析模型是否出现过拟合。
* 如果发现过拟合，考虑增加 Dropout 层、使用数据增强或减少网络复杂度等方法进行处理。

1. **实验结果**
2. 打印测试准确率；

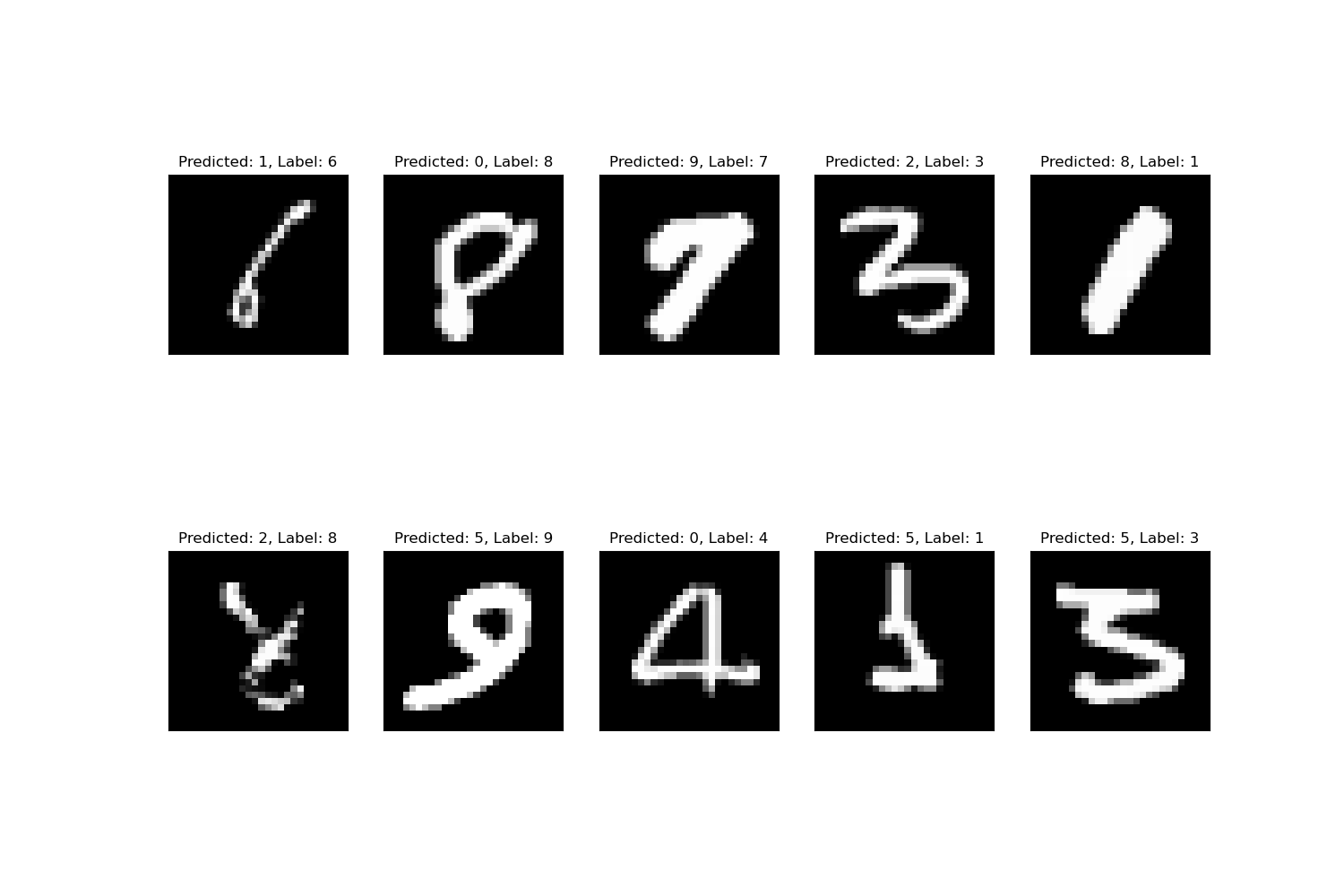


|  |
| --- |
| Train set: Epoch: 20, Average loss: 0.1395, Accuracy: 57517/60000 (96%) |
| Test set: Avg. loss: 0.0373, Accuracy: 9884/10000 (99%) |

在 20 个 epoch 结束时，模型在测试集上达到了 99% 的准确率。

1. 可视化典型错分图片；

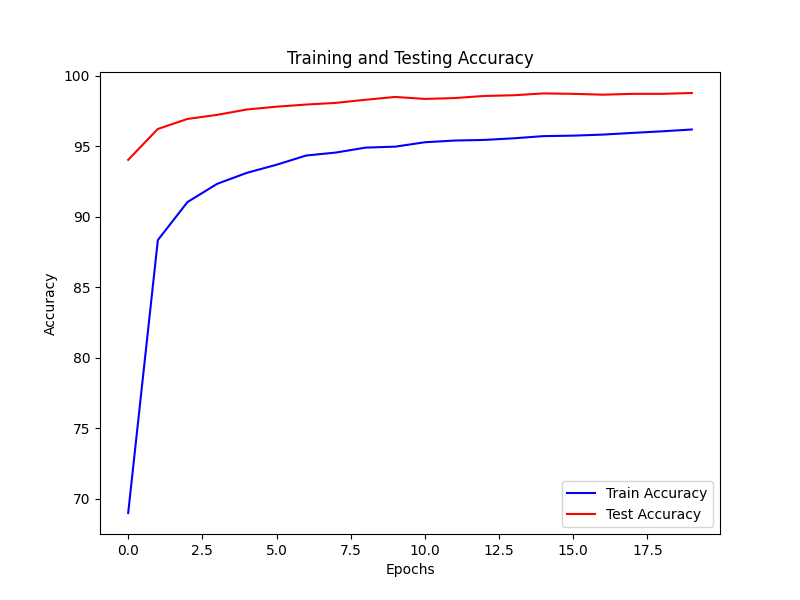
展示了模型错分的典型手写数字图片，包括它们的预测标签和真实标签。



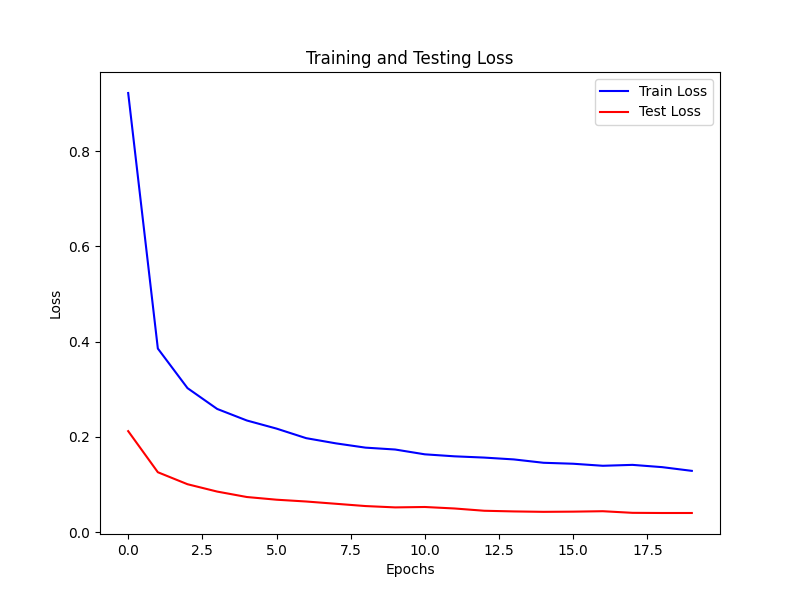
1. 绘制训练和测试曲线；

绘制了训练集和测试集上的损失及准确率曲线，展示了模型性能随着训练进展的变化情况。

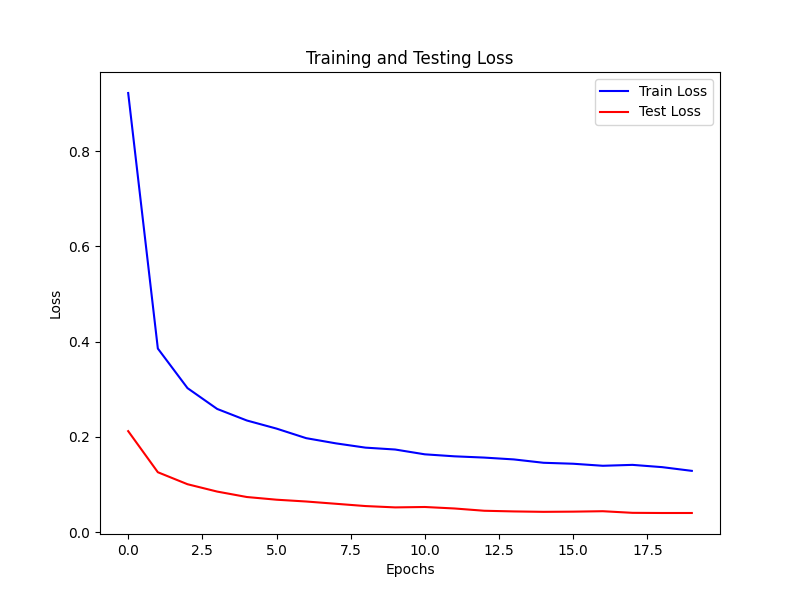
Accuracy曲线：



Loss曲线：



4、扩展任务。



根据损失和准确率曲线分析，由training的loss曲线，我们可以看到loss随着训练epoch增长不断下降，模型没有明显的过拟合迹象。测试准确率稳定在较高水平，且训练损失持续下降。在15轮左右loss基本不变，loss没有出现随着训练轮数增加反而上涨的情况，因此没有出现过拟合。过拟合一般是因为数据集数量太少或数据集质量不佳导致了模型过度学习了数据集中一些没有用的参数部分，因此导致模型的loss随着训练过程不降反增，一般可以选择对数据集进行处理，dropout随机删除一些特征或者增加数据集的方法来缓解过拟合。

1. **实验总结**

本次实验中，我成功实现了基于卷积神经网络的手写数字识别模型。通过对 MNIST 数据集的处理和模型的训练，我深入了解了卷积神经网络的工作原理和结构设计。实验过程中，我学会了如何使用 PyTorch 进行数据预处理、模型搭建、训练调优以及结果的可视化。

在模型评估方面，我学习了如何计算和解读准确率，以及如何通过可视化工具分析模型的性能和行为。此外，我也掌握了如何识别和处理过拟合的问题，尽管在本次实验中模型表现良好，未出现过拟合现象。

总的来说，这次实验加深了我对深度学习在图像分类任务中应用的理解，为我未来在更复杂场景下应用神经网络打下了坚实的基础。

**附件：**解晗轲-2021301931-10042101-代码

|  |
| --- |
| import torch import torchvision from torch.utils.data import DataLoader import matplotlib.pyplot as plt import torch.nn as nn import torch.nn.functional as F import torch.optim as optim import os os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'   n\_epochs = 20 batch\_size\_train = 64 batch\_size\_test = 1000 learning\_rate = 0.01 momentum = 0.5 log\_interval = 10 random\_seed = 1 torch.manual\_seed(random\_seed) device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") #数据加载，利用torch库中现有的dataloader train\_loader = torch.utils.data.DataLoader(  torchvision.datasets.MNIST('./data/', train=True, download=True,  transform=torchvision.transforms.Compose([  torchvision.transforms.ToTensor(),  torchvision.transforms.Normalize(  (0.14,), (0.3,))  ])),  batch\_size=batch\_size\_train, shuffle=True) test\_loader = torch.utils.data.DataLoader(  torchvision.datasets.MNIST('./data/', train=False, download=True,  transform=torchvision.transforms.Compose([  torchvision.transforms.ToTensor(),  torchvision.transforms.Normalize(  (0.1307,), (0.3081,))  ])),  batch\_size=batch\_size\_test, shuffle=True)   # 定义网络结构 class Net(nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  # 第一个卷积层，接受1个输入通道，输出10个通道，卷积核大小为5  self.conv1 = nn.Conv2d(1, 10, kernel\_size=5)  # 第二个卷积层，接受10个输入通道，输出20个通道，卷积核大小为5  self.conv2 = nn.Conv2d(10, 20, kernel\_size=5)  # Dropout层，随机丢弃一些神经元，防止过拟合  self.conv2\_drop = nn.Dropout2d()  # 第一个全连接层，将320维的数据映射到50维  self.fc1 = nn.Linear(320, 50)  # 第二个全连接层，将50维数据映射到10维（例如，10个类别）  self.fc2 = nn.Linear(50, 10)   # 定义网络的前向传播路径  def forward(self, x):  # 通过第一个卷积层，然后应用2x2的最大池化和ReLU激活函数  x = F.relu(F.max\_pool2d(self.conv1(x), 2))  # 通过第二个卷积层，应用dropout，然后再次应用2x2的最大池化和ReLU激活函数  x = F.relu(F.max\_pool2d(self.conv2\_drop(self.conv2(x)), 2))  # 展平多维输入数据  x = x.view(-1, 320)  # 通过第一个全连接层，然后应用ReLU激活函数  x = F.relu(self.fc1(x))  # 应用dropout  x = F.dropout(x, training=self.training)  # 通过第二个全连接层  x = self.fc2(x)  # 应用log\_softmax，对输出进行对数似然变换  return F.log\_softmax(x, dim=1)   network = Net().to(device) optimizer = optim.SGD(network.parameters(), lr=learning\_rate,  momentum=momentum)   # 存储准确率和损失 train\_losses = [] train\_acc = [] test\_losses = [] test\_acc = []  def train(epoch):  network.train() # 将模型设置为训练模式  train\_loss = 0 # 初始化累计训练损失为0  correct = 0 # 初始化正确预测的样本数为0  total = 0 # 初始化总样本数为0   for batch\_idx, (data, target) in enumerate(train\_loader): # 遍历训练数据  data, target = data.to(device), target.to(device) # 将数据和标签转移到设备（GPU或CPU）  optimizer.zero\_grad() # 清除之前的梯度  output = network(data) # 得到模型输出  loss = F.nll\_loss(output, target) # 计算损失  train\_loss += loss.item() # 累加损失  loss.backward() # 反向传播  optimizer.step() # 更新模型参数   pred = output.data.max(1, keepdim=True)[1] # 获取最大概率的预测结果  correct += pred.eq(target.data.view\_as(pred)).sum().item() # 计算正确预测的数量  total += target.size(0) # 累计样本数量   if batch\_idx % log\_interval == 0: # 每隔一定批次打印训练状态  print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(  epoch, batch\_idx \* len(data), len(train\_loader.dataset),  100. \* batch\_idx / len(train\_loader), loss.item()))   train\_loss /= len(train\_loader) # 计算平均训练损失  train\_losses.append(train\_loss) # 将平均损失添加到列表中  accuracy = 100. \* correct / total # 计算训练准确率  train\_acc.append(accuracy) # 将训练准确率添加到列表中  print('\nTrain set: Epoch: {}, Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(  epoch, train\_loss, correct, total, accuracy)) # 打印训练集的平均损失和准确率    def test():  network.eval() # 将模型设置为评估模式  test\_loss = 0 # 初始化累计测试损失为0  correct = 0 # 初始化正确预测的样本数为0  total = 0 # 初始化总样本数为0  misclassified\_examples = [] # 初始化错分样本列表   with torch.no\_grad(): # 不计算梯度，以节省计算资源  for data, target in test\_loader: # 遍历测试数据  data, target = data.to(device), target.to(device) # 将数据和标签转移到设备  output = network(data) # 得到模型输出  test\_loss += F.nll\_loss(output, target, reduction='sum').item() # 累加损失  pred = output.data.max(1, keepdim=True)[1] # 获取最大概率的预测结果  correct += pred.eq(target.data.view\_as(pred)).sum().item() # 计算正确预测的数量  total += target.size(0) # 累计样本数量   # 收集错分的图片  wrong\_idx = (pred != target.view\_as(pred)).nonzero()[:, 0]  for idx in wrong\_idx:  misclassified\_examples.append((data[idx], pred[idx], target[idx]))   test\_loss /= len(test\_loader.dataset) # 计算平均测试损失  test\_losses.append(test\_loss) # 将平均损失添加到列表中  accuracy = 100. \* correct / total # 计算测试准确率  test\_acc.append(accuracy) # 将测试准确率添加到列表中  print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(  test\_loss, correct, total, accuracy)) # 打印测试集的平均损失和准确率   return misclassified\_examples # 返回错分样本列表   # 在训练和测试循环中 misclassified\_examples = [] for epoch in range(1, n\_epochs + 1):  train(epoch)  misclassified\_examples = test() # 更新错分图片集  def plot\_misclassified\_images(misclassified, num\_images=10):  fig = plt.figure(figsize=(15, 10))  for i, (img, pred, label) in enumerate(misclassified[:num\_images]):  img, pred, label = img.to('cpu'), pred.to('cpu'), label.to('cpu')  img = img.squeeze().numpy() # 转换为 NumPy 数组  ax = fig.add\_subplot(2, 5, i+1)  ax.imshow(img, cmap='gray', interpolation='none')  ax.set\_title(f"Predicted: {pred.item()}, Label: {label.item()}")  ax.set\_xticks([])  ax.set\_yticks([])  plt.show()  plot\_misclassified\_images(misclassified\_examples, num\_images=10)  # 绘制损失曲线 fig1 = plt.figure(figsize=(8, 6)) plt.plot(train\_losses, color='blue') plt.plot(test\_losses, color='red') plt.xlabel('Epochs') plt.ylabel('Loss') plt.title('Training and Testing Loss') plt.legend(['Train Loss', 'Test Loss']) plt.show()  # 绘制准确率曲线 fig2 = plt.figure(figsize=(8, 6)) plt.plot(train\_acc, color='blue') plt.plot(test\_acc, color='red') plt.xlabel('Epochs') plt.ylabel('Accuracy') plt.title('Training and Testing Accuracy') plt.legend(['Train Accuracy', 'Test Accuracy']) plt.show() |