上机实验三：基于卷积神经网络的两位数字比较

计卓2101 高僖 U202115285

实验目的

本实验的目的是使用孪生神经网络（Siamese Network）进行手写字符体的相似性度量，通过训练模型来学习字符的相似性，并在测试集上进行评估。

数据集构成

训练集

训练集采用 MNIST 数据集的 10% 子集，包含手写字符图像和对应的标签。

测试集

测试集同样采用 MNIST 数据集的 10% 子集，包含手写字符图像和对应的标签。

神经网络架构

卷积神经网络（CNN）结构

本实验采用的神经网络是一个简单的孪生卷积神经网络（Siamese CNN），由两个相同的卷积神经网络组成，用于处理两个输入图像。网络的输入是两个手写字符图像，通过卷积和全连接层的处理，学习两个输入图像的相似性。损失函数采用 Binary Cross Entropy Loss，优化器使用 Adam。

```
CnnNetwork(
  (conv1): Conv2d(1, 32, kernel_size=5)
  (conv2): Conv2d(32, 64, kernel_size=5)
  (fc1): Linear(in_features=1024, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=1, bias=True)
)
```

卷积层 1

- 输入通道数：1（灰度图像）
- 输出通道数：32
- 卷积核大小：5x5
- 步长：默认为 1
- 输出大小：通过公式计算，((input_size - kernel_size) + 2 * padding) / stride + 1

激活函数 1

- ReLU（Rectified Linear Unit）

池化层 1

- 最大池化
- 池化核大小：2x2
- 步长：默认为 2

卷积层 2

- 输入通道数：32
- 输出通道数：64
- 卷积核大小：5x5
- 步长：默认为 1
- 输出大小：通过公式计算，((input_size - kernel_size) + 2 * padding) / stride + 1

激活函数 2

- ReLU（Rectified Linear Unit）

池化层 2

- 最大池化
- 池化核大小：2x2
- 步长：默认为 2

全连接层 1

- 输入大小：64x4x4（通过卷积和池化后的结果）
- 输出大小：256

激活函数 3

- ReLU（Rectified Linear Unit）

全连接层 2

- 输入大小：256
- 输出大小：1

损失函数

- Binary Cross Entropy Loss

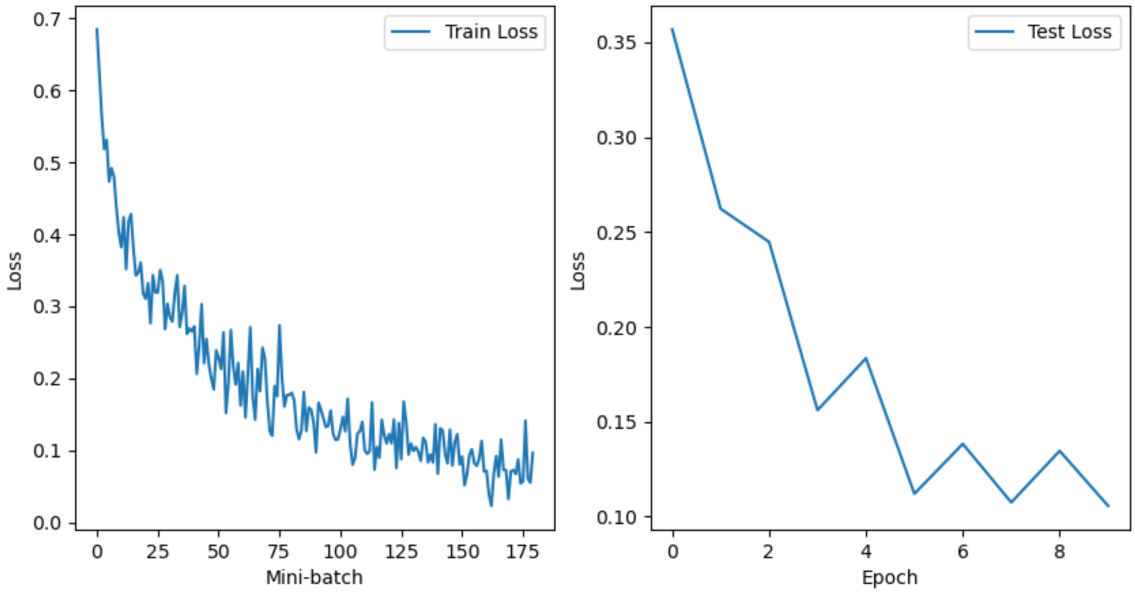
优化器

- Adam Optimizer

实验结果

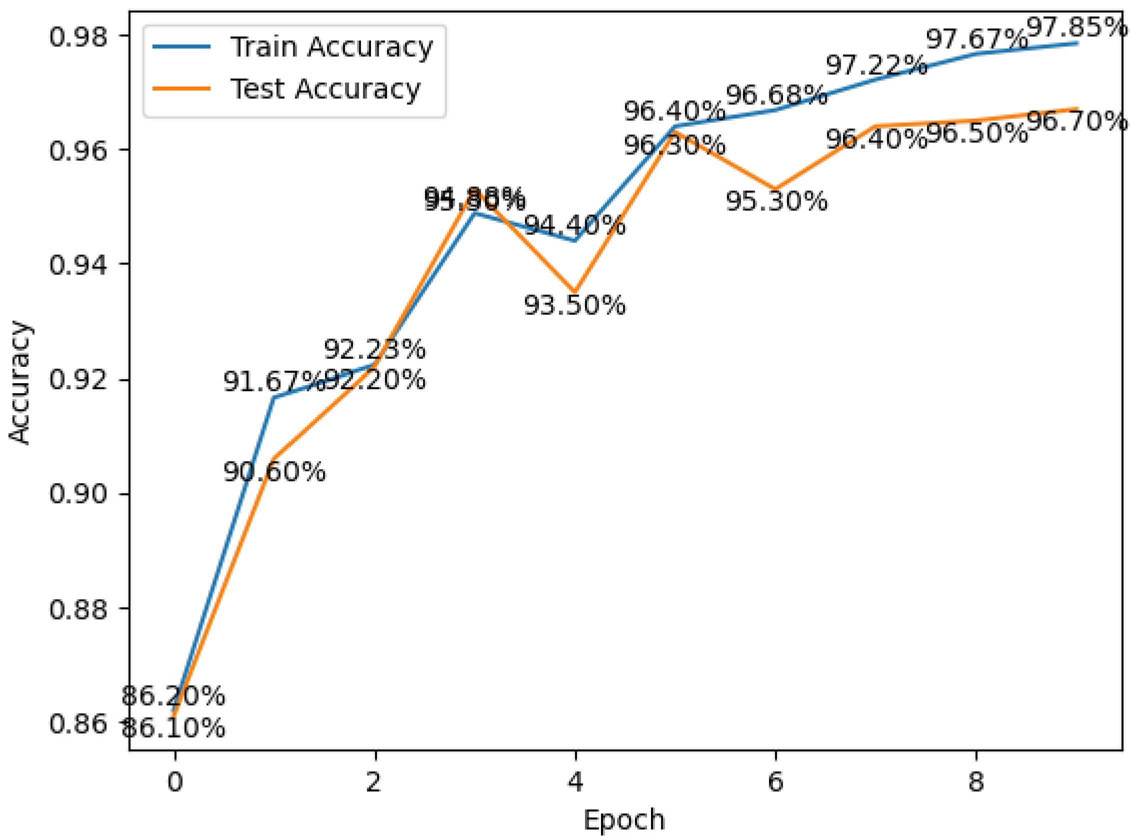
损失曲线

由于本人电脑性能有限，没有计算每一个mini-batch的test loss，但是计算了每个epoch的test loss。



准确率曲线

训练和测试准确率曲线如下图所示：



实验总结

通过实验，我成功地使用孪生神经网络进行手写字符的相似性学习，并在测试集上取得了良好的准确率。实验结果表明，孪生神经网络在字符相似性度量任务上取得了良好的效果。

训练结果概述

本实验使用了一个包含卷积神经网络（CNN）结构的模型，并在训练集和测试集上进行了为期10个epochs的训练。以下是每个epoch中部分训练过程的结果：

- **Epoch 1/10:**
  - 平均训练损失（Avg. Train Loss）：0.4368
  - 训练准确率（Train Accuracy）：86.20%
  - 测试损失（Test Loss）：0.3568
  - 测试准确率（Test Accuracy）：86.10%
- **Epoch 2/10:**
  - 平均训练损失（Avg. Train Loss）：0.2834
  - 训练准确率（Train Accuracy）：91.67%
  - 测试损失（Test Loss）：0.2623
  - 测试准确率（Test Accuracy）：90.60%

...

- **Epoch 10/10:**
  - 平均训练损失（Avg. Train Loss）：0.0652
  - 训练准确率（Train Accuracy）：97.85%
  - 测试损失（Test Loss）：0.1057
  - 测试准确率（Test Accuracy）：96.70%

结果分析

- 随着训练的进行，模型在训练集和测试集上的损失逐渐减小，准确率逐渐提高，表现出模型学习的良好趋势。
- 模型在训练集上的准确率较高，达到了97.85%，并且在测试集上也取得了96.70%的准确率，表明模型具有很好的泛化性能。
- 训练集和测试集上的损失和准确率的变化趋势基本一致，说明模型没有出现过拟合或欠拟合的问题。

模型评估

- 该模型在处理图像分类任务中表现出色，经过10个epochs的训练后，在测试集上达到了很高的准确率。
- 通过可视化损失曲线和准确率曲线，我们可以清晰地观察到模型的训练过程，并验证模型的收敛性和泛化性。
- 最终结果表明，所采用的神经网络架构和训练策略对于解决该图像分类问题是有效的。

```
import torch
from torchvision import datasets, transforms
```

```python
from torch.utils.data import DataLoader, Dataset
import numpy as np
import random
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt

# 添加设备选择
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

#将数据转化为tensor类型 并且进行标准化
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,),(0.5,))
])

#从torch数据集里加载手写字符体
train_data = datasets.MNIST(root='./data', train=True,  download=True,
transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

#根据要求抽取10%的数据
num_train = int(0.1 * len(train_data))
num_test = int(0.1 * len(test_data))

#生成索引 索引范围 生成数量 是否允许重复
train_indices = np.random.choice(range(len(train_data)), num_train, replace=False)
test_indices = np.random.choice(range(len(test_data)), num_test, replace=False)

#利用索引取出数据
train_data_sub = torch.utils.data.Subset(train_data, train_indices)
test_data_sub = torch.utils.data.Subset(test_data, test_indices)

class SiameseMNIST(Dataset):
    def __init__(self, dataset):
        self.dataset = dataset

    def __getitem__(self, index):
        img1, label1 = self.dataset[index]
        #p=0.5选择类别是否相同
        same = random.randint(0, 1)
        if same:
            while True:
                #随机取出一个 直到满足要求
                index2 = np.random.choice(range(len(self.dataset)))
                img2, label2 = self.dataset[index2]
                if label2 == label1:
                    break
        else:
            while True:
                img2, label2 =
self.dataset[np.random.choice(range(len(self.dataset)))]
```

```
                if label2 != label1:
                    break
        return img1, img2, torch.from_numpy(np.array([int(same)],
dtype=np.float32))

    def __len__(self):
        return len(self.dataset)

#二元组数据集
train_siamese = SiameseMNIST(train_data_sub)
test_siamese = SiameseMNIST(test_data_sub)

#放入迭代器
train_loader = DataLoader(train_siamese, batch_size=32, shuffle=True)
test_loader = DataLoader(test_siamese, batch_size=32, shuffle=False)

class CnnNetwork(nn.Module):
    def __init__(self):
        super(CnnNetwork, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.fc1 = nn.Linear(1024, 256)
        self.fc2 = nn.Linear(256, 1)

    def forward_once(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        return x

    def forward(self, input1, input2):
        output1 = self.forward_once(input1)
        output2 = self.forward_once(input2)
        distance = (output1 - output2) ** 2
        output = self.fc2(distance)
        return output

# 创建孪生网络实例
siamese_network = CnnNetwork().to(device)

# 选择损失函数和优化器
criterion = nn.BCEWithLogitsLoss()  # Binary Cross Entropy Loss
optimizer = optim.Adam(siamese_network.parameters(), lr=0.001)

# 创建空列表，用于存储训练和测试的损失以及准确率
train_losses = []
test_losses = []
train_accuracies = []
test_accuracies = []

# 训练参数
num_epochs = 10
```

```
# 训练循环
for epoch in range(num_epochs):
    siamese_network.train()  # 设置模型为训练模式
    total_loss = 0.0

    for batch_idx, (img1, img2, target) in enumerate(train_loader):
        # 将输入数据移动到设备（GPU或CPU）
        img1, img2, target = img1.to(device), img2.to(device), target.to(device)

        # 清零梯度
        optimizer.zero_grad()

        # 前向传播
        output = siamese_network(img1, img2)

        # 计算损失
        loss = criterion(output, target)

        # 反向传播和优化
        loss.backward()
        optimizer.step()

        # 累计损失
        total_loss += loss.item()

        # 每个mini-batch结束后记录训练损失
        if batch_idx % 10 == 9:  # 每10个mini-batch记录一次
            avg_train_loss = total_loss / 10
            print(f"Epoch {epoch + 1}/{num_epochs}, Batch {batch_idx +
1}/{len(train_loader)}, Avg. Train Loss: {avg_train_loss:.4f}")
            train_losses.append(avg_train_loss)
            total_loss = 0.0

    # 在每个epoch结束后记录最终的训练准确率
    with torch.no_grad():
        correct = 0
        total = 0

        for batch_idx, (img1, img2, target) in enumerate(train_loader):
            # 将输入数据移动到设备（GPU或CPU）
            img1, img2, target = img1.to(device), img2.to(device),
target.to(device)

            # 前向传播
            output = siamese_network(img1, img2)

            # 预测标签
            predictions = torch.sigmoid(output) > 0.5

            # 统计准确率
            total += target.size(0)
            correct += (predictions == target).sum().item()

        accuracy = correct / total
```

```
        print(f"Epoch {epoch + 1}/{num_epochs}, Train Accuracy: {accuracy * 
100:.2f}%")
        train_accuracies.append(accuracy)

    # 在测试集上评估模型
    siamese_network.eval()  # 设置模型为评估模式
    with torch.no_grad():
        correct = 0
        total = 0
        test_loss = 0.0

        for batch_idx, (img1, img2, target) in enumerate(test_loader):
            # 将输入数据移动到设备（GPU或CPU）
            img1, img2, target = img1.to(device), img2.to(device), 
target.to(device)

            # 前向传播
            output = siamese_network(img1, img2)

            # 计算损失
            loss = criterion(output, target)
            test_loss += loss.item()

            # 预测标签
            predictions = torch.sigmoid(output) > 0.5

            # 统计准确率
            total += target.size(0)
            correct += (predictions == target).sum().item()

        # 记录测试损失
        test_loss = test_loss / len(test_loader)
        print(f"Epoch {epoch + 1}/{num_epochs}, Test Loss: {test_loss:.4f}")
        test_losses.append(test_loss)

        accuracy = correct / total
        print(f"Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy * 
100:.2f}%")
        test_accuracies.append(accuracy)
```

```
Epoch 1/10, Batch 10/188, Avg. Train Loss: 0.6843
Epoch 1/10, Batch 20/188, Avg. Train Loss: 0.6244
Epoch 1/10, Batch 30/188, Avg. Train Loss: 0.5631
Epoch 1/10, Batch 40/188, Avg. Train Loss: 0.5183
Epoch 1/10, Batch 50/188, Avg. Train Loss: 0.5310
Epoch 1/10, Batch 60/188, Avg. Train Loss: 0.4733
Epoch 1/10, Batch 70/188, Avg. Train Loss: 0.4917
```

```
Epoch 1/10, Batch 80/188, Avg. Train Loss: 0.4803
Epoch 1/10, Batch 90/188, Avg. Train Loss: 0.4374
Epoch 1/10, Batch 100/188, Avg. Train Loss: 0.4020
Epoch 1/10, Batch 110/188, Avg. Train Loss: 0.3822
Epoch 1/10, Batch 120/188, Avg. Train Loss: 0.4237
Epoch 1/10, Batch 130/188, Avg. Train Loss: 0.3516
Epoch 1/10, Batch 140/188, Avg. Train Loss: 0.4181
Epoch 1/10, Batch 150/188, Avg. Train Loss: 0.4281
Epoch 1/10, Batch 160/188, Avg. Train Loss: 0.3801
Epoch 1/10, Batch 170/188, Avg. Train Loss: 0.3430
Epoch 1/10, Batch 180/188, Avg. Train Loss: 0.3469
Epoch 1/10, Train Accuracy: 86.20%
Epoch 1/10, Test Loss: 0.3568
Epoch 1/10, Test Accuracy: 86.10%
Epoch 2/10, Batch 10/188, Avg. Train Loss: 0.3607
Epoch 2/10, Batch 20/188, Avg. Train Loss: 0.3177
Epoch 2/10, Batch 30/188, Avg. Train Loss: 0.3105
Epoch 2/10, Batch 40/188, Avg. Train Loss: 0.3322
Epoch 2/10, Batch 50/188, Avg. Train Loss: 0.2766
Epoch 2/10, Batch 60/188, Avg. Train Loss: 0.3432
Epoch 2/10, Batch 70/188, Avg. Train Loss: 0.3193
Epoch 2/10, Batch 80/188, Avg. Train Loss: 0.3193
Epoch 2/10, Batch 90/188, Avg. Train Loss: 0.3504
Epoch 2/10, Batch 100/188, Avg. Train Loss: 0.3354
Epoch 2/10, Batch 110/188, Avg. Train Loss: 0.2685
Epoch 2/10, Batch 120/188, Avg. Train Loss: 0.3036
Epoch 2/10, Batch 130/188, Avg. Train Loss: 0.2847
Epoch 2/10, Batch 140/188, Avg. Train Loss: 0.2789
Epoch 2/10, Batch 150/188, Avg. Train Loss: 0.3190
Epoch 2/10, Batch 160/188, Avg. Train Loss: 0.3432
Epoch 2/10, Batch 170/188, Avg. Train Loss: 0.2716
Epoch 2/10, Batch 180/188, Avg. Train Loss: 0.2906
Epoch 2/10, Train Accuracy: 91.67%
Epoch 2/10, Test Loss: 0.2623
Epoch 2/10, Test Accuracy: 90.60%
Epoch 3/10, Batch 10/188, Avg. Train Loss: 0.3283
Epoch 3/10, Batch 20/188, Avg. Train Loss: 0.2618
Epoch 3/10, Batch 30/188, Avg. Train Loss: 0.2693
Epoch 3/10, Batch 40/188, Avg. Train Loss: 0.2652
Epoch 3/10, Batch 50/188, Avg. Train Loss: 0.2722
Epoch 3/10, Batch 60/188, Avg. Train Loss: 0.2063
Epoch 3/10, Batch 70/188, Avg. Train Loss: 0.2454
Epoch 3/10, Batch 80/188, Avg. Train Loss: 0.3031
Epoch 3/10, Batch 90/188, Avg. Train Loss: 0.2212
Epoch 3/10, Batch 100/188, Avg. Train Loss: 0.2549
Epoch 3/10, Batch 110/188, Avg. Train Loss: 0.2195
Epoch 3/10, Batch 120/188, Avg. Train Loss: 0.1997
Epoch 3/10, Batch 130/188, Avg. Train Loss: 0.1847
Epoch 3/10, Batch 140/188, Avg. Train Loss: 0.2390
Epoch 3/10, Batch 150/188, Avg. Train Loss: 0.2284
Epoch 3/10, Batch 160/188, Avg. Train Loss: 0.2131
```

```
Epoch 3/10, Batch 170/188, Avg. Train Loss: 0.2638
Epoch 3/10, Batch 180/188, Avg. Train Loss: 0.1520
Epoch 3/10, Train Accuracy: 92.23%
Epoch 3/10, Test Loss: 0.2448
Epoch 3/10, Test Accuracy: 92.20%
Epoch 4/10, Batch 10/188, Avg. Train Loss: 0.1927
Epoch 4/10, Batch 20/188, Avg. Train Loss: 0.2673
Epoch 4/10, Batch 30/188, Avg. Train Loss: 0.2163
Epoch 4/10, Batch 40/188, Avg. Train Loss: 0.1917
Epoch 4/10, Batch 50/188, Avg. Train Loss: 0.2215
Epoch 4/10, Batch 60/188, Avg. Train Loss: 0.1625
Epoch 4/10, Batch 70/188, Avg. Train Loss: 0.2095
Epoch 4/10, Batch 80/188, Avg. Train Loss: 0.1458
Epoch 4/10, Batch 90/188, Avg. Train Loss: 0.2050
Epoch 4/10, Batch 100/188, Avg. Train Loss: 0.2711
Epoch 4/10, Batch 110/188, Avg. Train Loss: 0.1746
Epoch 4/10, Batch 120/188, Avg. Train Loss: 0.1428
Epoch 4/10, Batch 130/188, Avg. Train Loss: 0.2129
Epoch 4/10, Batch 140/188, Avg. Train Loss: 0.1823
Epoch 4/10, Batch 150/188, Avg. Train Loss: 0.2428
Epoch 4/10, Batch 160/188, Avg. Train Loss: 0.2276
Epoch 4/10, Batch 170/188, Avg. Train Loss: 0.1677
Epoch 4/10, Batch 180/188, Avg. Train Loss: 0.1269
Epoch 4/10, Train Accuracy: 94.88%
Epoch 4/10, Test Loss: 0.1561
Epoch 4/10, Test Accuracy: 95.30%
Epoch 5/10, Batch 10/188, Avg. Train Loss: 0.1205
Epoch 5/10, Batch 20/188, Avg. Train Loss: 0.1893
Epoch 5/10, Batch 30/188, Avg. Train Loss: 0.1754
Epoch 5/10, Batch 40/188, Avg. Train Loss: 0.2737
Epoch 5/10, Batch 50/188, Avg. Train Loss: 0.2008
Epoch 5/10, Batch 60/188, Avg. Train Loss: 0.1611
Epoch 5/10, Batch 70/188, Avg. Train Loss: 0.1769
Epoch 5/10, Batch 80/188, Avg. Train Loss: 0.1770
Epoch 5/10, Batch 90/188, Avg. Train Loss: 0.1800
Epoch 5/10, Batch 100/188, Avg. Train Loss: 0.1699
Epoch 5/10, Batch 110/188, Avg. Train Loss: 0.1295
Epoch 5/10, Batch 120/188, Avg. Train Loss: 0.1157
Epoch 5/10, Batch 130/188, Avg. Train Loss: 0.1281
Epoch 5/10, Batch 140/188, Avg. Train Loss: 0.1813
Epoch 5/10, Batch 150/188, Avg. Train Loss: 0.1270
Epoch 5/10, Batch 160/188, Avg. Train Loss: 0.1597
Epoch 5/10, Batch 170/188, Avg. Train Loss: 0.1566
Epoch 5/10, Batch 180/188, Avg. Train Loss: 0.1382
Epoch 5/10, Train Accuracy: 94.40%
Epoch 5/10, Test Loss: 0.1835
Epoch 5/10, Test Accuracy: 93.50%
Epoch 6/10, Batch 10/188, Avg. Train Loss: 0.0972
Epoch 6/10, Batch 20/188, Avg. Train Loss: 0.1663
Epoch 6/10, Batch 30/188, Avg. Train Loss: 0.1570
Epoch 6/10, Batch 40/188, Avg. Train Loss: 0.1449
```

```
Epoch 6/10, Batch 50/188, Avg. Train Loss: 0.1326
Epoch 6/10, Batch 60/188, Avg. Train Loss: 0.1342
Epoch 6/10, Batch 70/188, Avg. Train Loss: 0.1554
Epoch 6/10, Batch 80/188, Avg. Train Loss: 0.1231
Epoch 6/10, Batch 90/188, Avg. Train Loss: 0.1148
Epoch 6/10, Batch 100/188, Avg. Train Loss: 0.1155
Epoch 6/10, Batch 110/188, Avg. Train Loss: 0.1292
Epoch 6/10, Batch 120/188, Avg. Train Loss: 0.1466
Epoch 6/10, Batch 130/188, Avg. Train Loss: 0.1266
Epoch 6/10, Batch 140/188, Avg. Train Loss: 0.1717
Epoch 6/10, Batch 150/188, Avg. Train Loss: 0.1092
Epoch 6/10, Batch 160/188, Avg. Train Loss: 0.0801
Epoch 6/10, Batch 170/188, Avg. Train Loss: 0.0895
Epoch 6/10, Batch 180/188, Avg. Train Loss: 0.1232
Epoch 6/10, Train Accuracy: 96.40%
Epoch 6/10, Test Loss: 0.1121
Epoch 6/10, Test Accuracy: 96.30%
Epoch 7/10, Batch 10/188, Avg. Train Loss: 0.1269
Epoch 7/10, Batch 20/188, Avg. Train Loss: 0.1397
Epoch 7/10, Batch 30/188, Avg. Train Loss: 0.1001
Epoch 7/10, Batch 40/188, Avg. Train Loss: 0.0953
Epoch 7/10, Batch 50/188, Avg. Train Loss: 0.0987
Epoch 7/10, Batch 60/188, Avg. Train Loss: 0.1667
Epoch 7/10, Batch 70/188, Avg. Train Loss: 0.0734
Epoch 7/10, Batch 80/188, Avg. Train Loss: 0.1048
Epoch 7/10, Batch 90/188, Avg. Train Loss: 0.0899
Epoch 7/10, Batch 100/188, Avg. Train Loss: 0.1428
Epoch 7/10, Batch 110/188, Avg. Train Loss: 0.1210
Epoch 7/10, Batch 120/188, Avg. Train Loss: 0.1097
Epoch 7/10, Batch 130/188, Avg. Train Loss: 0.1229
Epoch 7/10, Batch 140/188, Avg. Train Loss: 0.1098
Epoch 7/10, Batch 150/188, Avg. Train Loss: 0.1428
Epoch 7/10, Batch 160/188, Avg. Train Loss: 0.0756
Epoch 7/10, Batch 170/188, Avg. Train Loss: 0.1381
Epoch 7/10, Batch 180/188, Avg. Train Loss: 0.0880
Epoch 7/10, Train Accuracy: 96.68%
Epoch 7/10, Test Loss: 0.1383
Epoch 7/10, Test Accuracy: 95.30%
Epoch 8/10, Batch 10/188, Avg. Train Loss: 0.1679
Epoch 8/10, Batch 20/188, Avg. Train Loss: 0.1400
Epoch 8/10, Batch 30/188, Avg. Train Loss: 0.0943
Epoch 8/10, Batch 40/188, Avg. Train Loss: 0.1095
Epoch 8/10, Batch 50/188, Avg. Train Loss: 0.0994
Epoch 8/10, Batch 60/188, Avg. Train Loss: 0.1048
Epoch 8/10, Batch 70/188, Avg. Train Loss: 0.0984
Epoch 8/10, Batch 80/188, Avg. Train Loss: 0.0857
Epoch 8/10, Batch 90/188, Avg. Train Loss: 0.1176
Epoch 8/10, Batch 100/188, Avg. Train Loss: 0.1118
Epoch 8/10, Batch 110/188, Avg. Train Loss: 0.0835
Epoch 8/10, Batch 120/188, Avg. Train Loss: 0.0942
Epoch 8/10, Batch 130/188, Avg. Train Loss: 0.0834
```

```
Epoch 8/10, Batch 140/188, Avg. Train Loss: 0.1365
Epoch 8/10, Batch 150/188, Avg. Train Loss: 0.0677
Epoch 8/10, Batch 160/188, Avg. Train Loss: 0.1307
Epoch 8/10, Batch 170/188, Avg. Train Loss: 0.1277
Epoch 8/10, Batch 180/188, Avg. Train Loss: 0.0950
Epoch 8/10, Train Accuracy: 97.22%
Epoch 8/10, Test Loss: 0.1075
Epoch 8/10, Test Accuracy: 96.40%
Epoch 9/10, Batch 10/188, Avg. Train Loss: 0.0821
Epoch 9/10, Batch 20/188, Avg. Train Loss: 0.1290
Epoch 9/10, Batch 30/188, Avg. Train Loss: 0.0792
Epoch 9/10, Batch 40/188, Avg. Train Loss: 0.1119
Epoch 9/10, Batch 50/188, Avg. Train Loss: 0.1227
Epoch 9/10, Batch 60/188, Avg. Train Loss: 0.0805
Epoch 9/10, Batch 70/188, Avg. Train Loss: 0.0918
Epoch 9/10, Batch 80/188, Avg. Train Loss: 0.0517
Epoch 9/10, Batch 90/188, Avg. Train Loss: 0.0671
Epoch 9/10, Batch 100/188, Avg. Train Loss: 0.0943
Epoch 9/10, Batch 110/188, Avg. Train Loss: 0.1017
Epoch 9/10, Batch 120/188, Avg. Train Loss: 0.0823
Epoch 9/10, Batch 130/188, Avg. Train Loss: 0.0788
Epoch 9/10, Batch 140/188, Avg. Train Loss: 0.0887
Epoch 9/10, Batch 150/188, Avg. Train Loss: 0.1132
Epoch 9/10, Batch 160/188, Avg. Train Loss: 0.0708
Epoch 9/10, Batch 170/188, Avg. Train Loss: 0.0718
Epoch 9/10, Batch 180/188, Avg. Train Loss: 0.0390
Epoch 9/10, Train Accuracy: 97.67%
Epoch 9/10, Test Loss: 0.1347
Epoch 9/10, Test Accuracy: 96.50%
Epoch 10/10, Batch 10/188, Avg. Train Loss: 0.0232
Epoch 10/10, Batch 20/188, Avg. Train Loss: 0.0680
Epoch 10/10, Batch 30/188, Avg. Train Loss: 0.0923
Epoch 10/10, Batch 40/188, Avg. Train Loss: 0.0638
Epoch 10/10, Batch 50/188, Avg. Train Loss: 0.1151
Epoch 10/10, Batch 60/188, Avg. Train Loss: 0.0730
Epoch 10/10, Batch 70/188, Avg. Train Loss: 0.0729
Epoch 10/10, Batch 80/188, Avg. Train Loss: 0.0326
Epoch 10/10, Batch 90/188, Avg. Train Loss: 0.0710
Epoch 10/10, Batch 100/188, Avg. Train Loss: 0.0728
Epoch 10/10, Batch 110/188, Avg. Train Loss: 0.0672
Epoch 10/10, Batch 120/188, Avg. Train Loss: 0.0876
Epoch 10/10, Batch 130/188, Avg. Train Loss: 0.0544
Epoch 10/10, Batch 140/188, Avg. Train Loss: 0.0570
Epoch 10/10, Batch 150/188, Avg. Train Loss: 0.1413
Epoch 10/10, Batch 160/188, Avg. Train Loss: 0.0611
Epoch 10/10, Batch 170/188, Avg. Train Loss: 0.0554
Epoch 10/10, Batch 180/188, Avg. Train Loss: 0.0967
Epoch 10/10, Train Accuracy: 97.85%
Epoch 10/10, Test Loss: 0.1057
Epoch 10/10, Test Accuracy: 96.70%
```

png

png

```
# 绘制训练损失曲线
plt.figure(figsize=(10, 5))  # 设置图表大小
plt.subplot(1, 2, 1)  # 将画布分成1行2列，当前是第1列
plt.plot(train_losses, label='Train Loss')
plt.xlabel('Mini-batch')
plt.ylabel('Loss')
plt.legend()

# 绘制测试损失曲线
plt.subplot(1, 2, 2)  # 将画布分成1行2列，当前是第2列
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')  # 如果想以Epoch为横坐标，可以改成 'Epoch'
plt.ylabel('Loss')
plt.legend()

# 保存图表为图片文件
plt.savefig('loss_curves.png')

# 显示图表
plt.show()
```

png

```
# 绘制准确率曲线
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# 在每个点上添加具体数值
for i, (train_acc, test_acc) in enumerate(zip(train_accuracies, test_accuracies)):
    plt.text(i, train_acc, f'{train_acc*100:.2f}%', ha='center', va='bottom')
    plt.text(i, test_acc, f'{test_acc*100:.2f}%', ha='center', va='top')

# 保存图表为图片文件
plt.savefig('accuracy_curves.png')

# 显示图表
plt.show()
```

png