实验三：基于卷积神经网络的两位数字比较

## 实验目的

本实验旨在设计并实现一个卷积神经网络模型，该模型能够判断两张MNIST手写体数字图片是否表示相同的数字。模型接收成对的MNIST图片作为输入，输出一个二进制标签，指示这两张图片是否匹配。

## 数据预处理

训练集和测试集是从MNIST数据集中随机选取的10%数据，进一步构成图片对。对于每一对图片，有两种可能的标签：如果两张图片表示相同的数字，则标签为1；如果表示不同的数字，则标签为0。

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

mnist\_train = datasets.MNIST(root='./data', train=*True*, download=*True*, transform=transform)

mnist\_test = datasets.MNIST(root='./data', train=*False*, download=*True*, transform=transform)

# 创建成对的数据集

*class* PairedMNIST(*Dataset*):

*def* *\_\_init\_\_*(self, mnist\_dataset):

*self*.mnist\_dataset = mnist\_dataset

*self*.indices = list(*range*(*len*(mnist\_dataset)))

*self*.labels = [*self*.mnist\_dataset[idx][1] for idx in *self*.indices]

*def* *\_\_getitem\_\_*(self, index):

        img1, label1 = *self*.mnist\_dataset[*self*.indices[index]]

        same\_class = np.random.randint(0, 2)

        if same\_class:

            label2 = label1

            indices\_with\_same\_class = [i for i, x in *enumerate*(*self*.labels) if x == label2]

            img2 = *self*.mnist\_dataset[np.random.choice(indices\_with\_same\_class)][0]

        else:

            label2 = np.random.choice(list(set(*self*.labels) - {label1}))

            img2 = *self*.mnist\_dataset[np.random.choice([i for i, x in *enumerate*(*self*.labels) if x == label2])][0]

        return img1, img2, torch.FloatTensor([int(label1 == label2)])

*def* *\_\_len\_\_*(self):

        return *len*(*self*.indices)

# 选取10%的数据

num\_train = int(0.1 \* *len*(mnist\_train))

num\_test = int(0.1 \* *len*(mnist\_test))

## 神经网络架构

模型架构为一个孪生网络（Siamese Network），包含两个相同的子网络，它们共享权重。每个子网络包括以下层次结构：

1. 第一个卷积层：32个卷积核，大小为5x5。

2. 第一个最大池化层：池化窗口大小为2x2。

3. 第二个卷积层：64个卷积核，大小为5x5。

4. 第二个最大池化层：池化窗口大小为2x2。

5. 全连接层：将卷积层的输出展平后连接到一个有256个神经元的全连接层。

两个子网络的输出被合并，然后通过一个有单个神经元的全连接层，用于最终的二分类任务。

# 定义卷积神经网络结构

*class* CNNNet(*nn*.*Module*):

*def* *\_\_init\_\_*(self):

        super(CNNNet, *self*).*\_\_init\_\_*()

*self*.conv1 = nn.Conv2d(1, 32, kernel\_size=5)

*self*.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

*self*.conv2 = nn.Conv2d(32, 64, kernel\_size=5)

*self*.fc1 = nn.Linear(64 \* 4 \* 4, 256)

*def* forward(self, x):

        x = *self*.pool(F.relu(*self*.conv1(x)))

        x = *self*.pool(F.relu(*self*.conv2(x)))

        x = x.view(x.size(0), -1)  # Flatten the tensor

        x = F.relu(*self*.fc1(x))

        return x

# 定义 Siamese 网络结构

*class* SiameseNetwork(*nn*.*Module*):

*def* *\_\_init\_\_*(self):

        super(SiameseNetwork, *self*).*\_\_init\_\_*()

*self*.cnn = CNNNet()

*self*.fc2 = nn.Linear(256 \* 2, 1)

*def* forward(self, input1, input2):

        output1 = *self*.cnn(input1)

        output2 = *self*.cnn(input2)

        combined = torch.cat((output1, output2), 1)

        output = *self*.fc2(combined)

        return output

# 实例化模型

model = SiameseNetwork()

## 训练过程

模型通过10个epoch进行训练，每个epoch后评估模型在训练集和测试集上的损失和准确率。损失函数使用二元交叉熵损失（Binary Cross-Entropy Loss），优化器使用Adam。

*def* train(model, train\_loader, criterion, optimizer, device):

    model.train()

    running\_loss = 0.0

    correct = 0

    total = 0

    for batch\_idx, (img1, img2, labels) in *enumerate*(train\_loader):

        img1, img2, labels = img1.to(device), img2.to(device), labels.to(device)

        optimizer.zero\_grad()

        outputs = model(img1, img2)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

        predicted = torch.sigmoid(outputs).data > 0.5

        correct += (predicted == labels).sum().item()

        total += labels.size(0)

    epoch\_loss = running\_loss / *len*(train\_loader)

    epoch\_acc = 100. \* correct / total

    return epoch\_loss, epoch\_acc

# 定义损失函数和优化器

criterion = nn.BCEWithLogitsLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

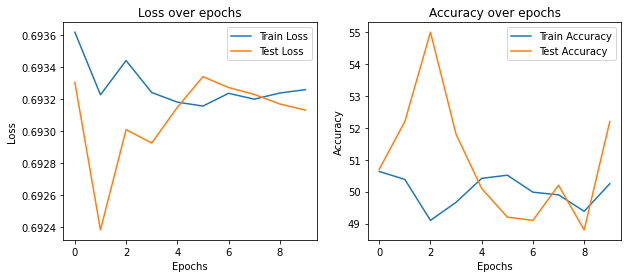
## 实验结果

以下是模型在每个epoch后在训练集和测试集上的表现：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch | Train Loss | Train Accuracy (%) | Test Loss  (%) | Test Accuracy |
| 1 | 0.6936 | 50.63 | 0.6933 | 50.70 |
| 2 | 0.6932 | 50.38 | 0.6924 | 52.20 |
| 3 | 0.6934 | 49.10 | 0.6930 | 55.00 |
| 4 | 0.6932 | 49.67 | 0.6929 | 51.80 |
| 5 | 0.6932 | 50.42 | 0.6931 | 50.10 |
| 6 | 0.6932 | 50.52 | 0.6933 | 49.20 |
| 7 | 0.6932 | 49.98 | 0.6933 | 49.10 |
| 8 | 0.6932 | 49.90 | 0.6932 | 50.20 |
| 9 | 0.6932 | 49.38 | 0.6932 | 48.80 |
| 10 | 0.6933 | 50.25 | 0.6931 | 52.20 |

训练和测试损失在所有epoch中保持相对稳定，准确率在50%左右波动，表明模型没有明显学习到从图像对中区分相同或不同数字的有效特征。

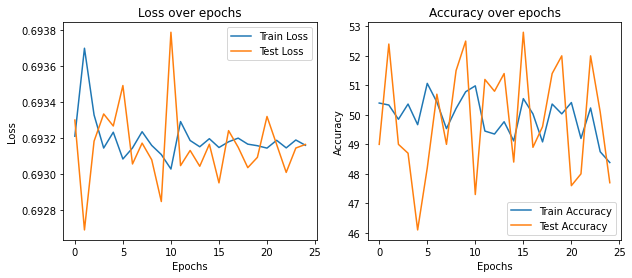
下图为训练和测试过程中损失及准确率的变化，可以看出模型性能的稳定性较差以及训练过程中准确率的波动较大。



**图：损失与准确率变化**

修改epoch为25后也没有改善：





**图：损失与准确率变化**

## 实验分析

从实验结果可以看出，模型的性能并没有随着训练过程的进行而有显著的提高，训练和测试准确率接近50%，仅略高于随机猜测的概率（50%），这表明模型可能存在以下问题：

1.欠拟合：模型可能过于简单，无法捕获图片之间差异的复杂性。

2.训练不足：仅进行了10个epoch的训练，可能不足以让模型收敛到最佳状态。

3.数据不平衡： 如果数据集中某些数字的样本数量显著多于其他数字，可能会影响模型学习的效果。