华中科技大学

2023

计算机视觉实验　课程设计报告

专　　业：	计算机科学与技术
班　　级：	计卓 2101 班
学　　号：	U202112071
姓　　名：	王彬
邮　　件：	wangbin2002@hust.edu.cn
完成日期：	2024.1.3

目 录

实验三：基于卷积神经网络的两位数字比较

1.1 任务描述

设计一个卷积神经网络，输入为两张 MNIST 手写体数字图片，如果两张图片为**同一个数字**（注意，非同一张图片），输出为 1，否则为 0。

从 MNIST 数据集的训练集中选取 10%作为本实验的训练图片，从 MNIST 数据集的测试集中选取 10%作为本实验的测试图片。请将该部分图片经过适当处理形成一定数量的用于本次实验的训练集和测试集。

注意事项：

1. 深度学习框架任选。
2. 实验报告需包含训练集和测试集构成、神经网络架构、每一轮 mini-batch 训练后的模型在训练集和测试集上的损失、最终的训练集和测试集准确率，以及对应的实验分析。

1.2 实验内容

(一) 数据集获取与重排序

我们首先加载 MNIST 数据集并进行预处理，将其导入模型中。这里我们直接使用 torch.utils.data.DataLoader()方法对 train_dataset 和 test_dataset 的训练集与测试集进行加载和重排序。这里需要注意生成图片对，即每两张图片为一组，通过这样导入 DataLoader，再进行取出。我们的代码如下：

```
#使用 cuda 训练
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
#用于加载图片对数据
class MNISTDataset(torch.utils.data.Dataset):
    def __init__(self, dataset):
        self.dataset = dataset

    def __getitem__(self, idx):
        img1, label1 = self.dataset[idx]
        img2, label2 = self.dataset[np.random.choice(len(self.dataset))]
        labels = (label1 == label2) # 标签是否相同
        return img1, img2, torch.tensor(labels, dtype=torch.float32)

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

# 使用 MNISTDataset
transform = transforms.Compose([transforms.ToTensor()])
mnist_train = MNIST(root='/data/', train=True, download=True,
transform=transform)
mnist_test = MNIST(root='/data/', train=False, download=True,
transform=transform)

train_dataset = MNISTDataset(mnist_train)
test_dataset = MNISTDataset(mnist_test)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

(二) 构建孪生神经网络模型

我们设计孪生神经网络模型。这里需要注意的是，输入的两张图片首先进过多层卷积、激活后，得到两个嵌入量，再由这两个嵌入量计算其欧几里得距离，以此

作为其相似量的衡量。我们对于卷积层使用同样的权值，也就是说，卷积层的参数是共享的。在我们的模型中，卷积层都是先经过两层卷积+激活+池化层后，进入全连接层，再将所求得值进行相减。

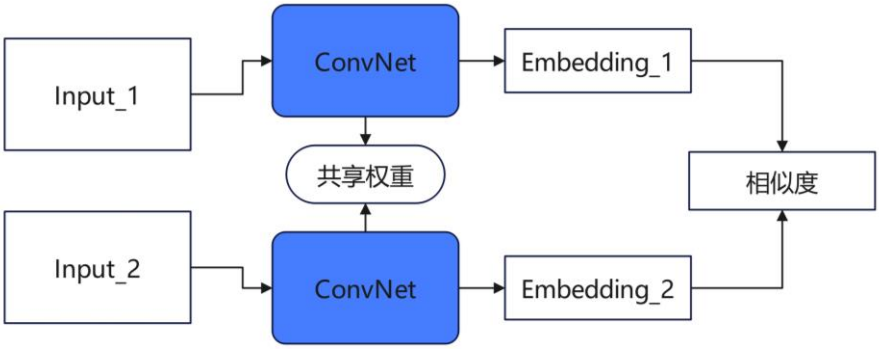

图 1 残差神经网络

我们的模型定义为：

```
# 定义孪生卷积神经网络模型
class SiameseNet(nn.Module):
    def __init__(self):
        super(SiameseNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
kernel_size=(3, 3))
        self.pool1 = nn.MaxPool2d(kernel_size=(2, 2))
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=(3, 3))
        self.pool2 = nn.MaxPool2d(kernel_size=(2, 2))
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(in_features=1600, out_features=128)
        self.fc2 = nn.Linear(in_features=128, out_features=1)

    def forward_one(self, x):
        x = self.conv1(x)
        x = nn.functional.relu(x)
```

```
        x = self.pool1(x)
        x = self.conv2(x)
        x = nn.functional.relu(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = nn.functional.relu(x)
        return x

    def forward(self, x1, x2):
        x1 = self.forward_one(x1)
        x2 = self.forward_one(x2)
        # 计算其几何距离
        x = torch.abs(x1 - x2)
        x = self.fc2(x)
        x = torch.sigmoid(x)
        return x
```

(三) 训练与结果绘制

由于篇幅限制，训练代码详见附录中的代码文件（./train_cuda.py）在测试集上可达到 99.47%准确率。

这里需要注意的是，我们在训练一定轮数（既一个 batch 后）就会对测试集的所有样本进行测试，并将测试集的结果写入输出列表。因此我们下文中所有准确率，均为实际测试样本的准确率。

对于每个类别的训练和测试结果，也就是每种数字的识别正确率，详见附录中的结果文件（./Results/）。该结果中有多个列表，分别是对训练和测试集的预测准确率及其损失。训练和测试代码如下。

```
# 定义损失函数和优化器
model = SiameseNet()
model.to(device)
criterion = nn.BCEWithLogitsLoss()
```

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


for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
    correct_train = 0
    batch_idx = 0


    for input1, input2, labels in train_loader:
        input1, input2, labels = input1.to(device), input2.to(device),
labels.to(device)
        batch_idx += 1
        optimizer.zero_grad()
        outputs = model(input1, input2)
        loss = criterion(outputs, labels.view(-1, 1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        correct_train += (torch.round(torch.sigmoid(outputs)) ==
labels.view(-1, 1)).sum().item()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss:
{:.6f}'.format(
                epoch, batch_idx, len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            log_loss_4_mini_batch.append(loss.item())


    loss_train = total_loss / len(train_loader)
    acc_train = correct_train / len(train_dataset)
```

```
    model.eval()
    total_loss = 0.0
    correct_test = 0

    with torch.no_grad():
        for input1, input2, labels in test_loader:
            input1, input2, labels = input1.to(device), input2.to(device),
labels.to(device)
            outputs = model(input1, input2)
            loss = criterion(outputs, labels.view(-1, 1))
            total_loss += loss.item()
            correct_test += (torch.round(torch.sigmoid(outputs)) ==
labels.view(-1, 1)).sum().item()

    loss_test = total_loss / len(test_loader)
    acc_test = correct_test / len(test_dataset)
```

对于图形绘制，代码详见（./matplot.py）。我们调用 matplotlib.pyplot 画出 Loss 和 Acc 曲线。这里，绘图函数需要传入文件地址参数，以便对多个曲线统一绘制，并进行对比分析。我们还编写了一个直接对列表绘制曲线的方法，在本实验中我们主要使用该方法进行图形绘制。

1.3 实验结果及其分析

经过实验，我们得到的最高测试集准确率为 99.70%，在训练集上的准确率也可达到 99.78%。根据图 2 可见模型在训练集上的拟合程度越来越高，而在测试集上的拟合程度先达到高峰后有一定程度的下降，这是由于模型逐渐发生过拟合。

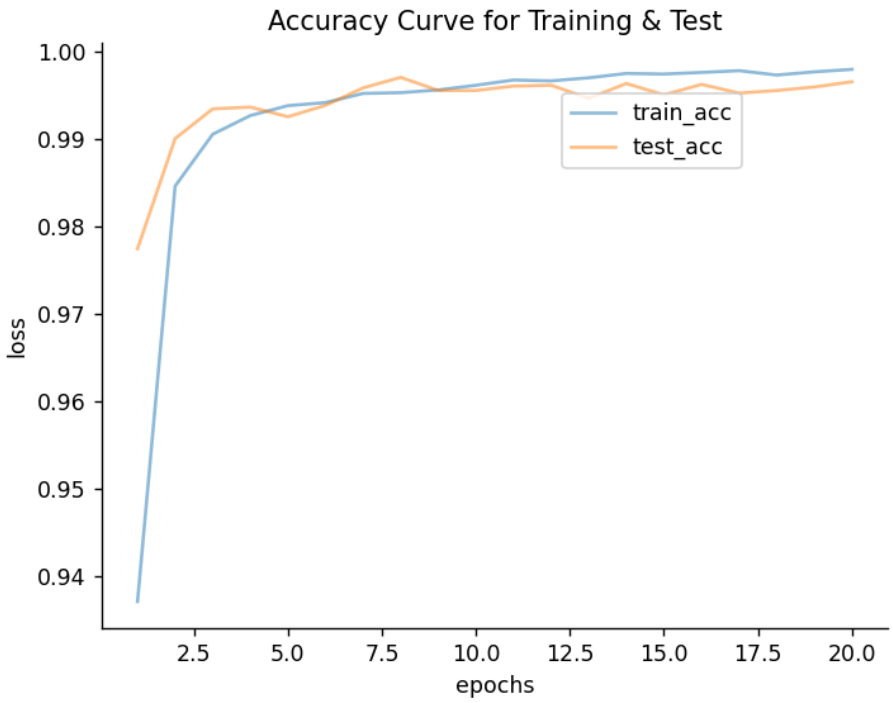

图 2 每一个 Epoch 的正确率

不同的损失函数设计对于网络性能有至关重要的影响。如果损失函数使用 BCELoss()，其测试绘图如图 3 所示，可见其测试集上和训练集上准确率均极低，仅为 15%水平。

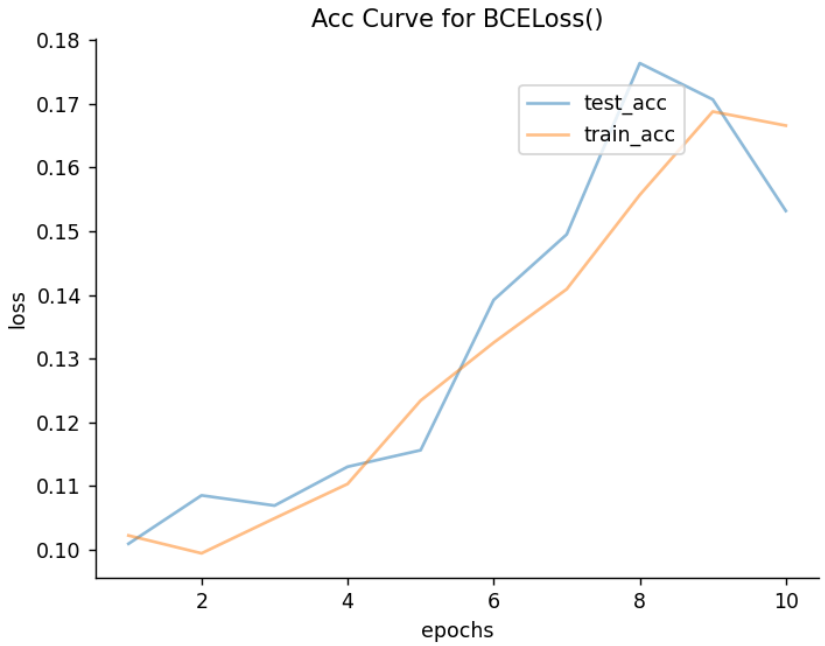

图 3 对于 BCELoss 损失函数的准确率

对于原先的孪生卷积网络，其在训练和测试集上的损失如图 4 所示。可见 Train_loss 逐渐下降，而 Test_loss 则有先下降后轻微上升的趋势，可能是由于图片对训练集和测试集完全分开而导致的泛化问题。

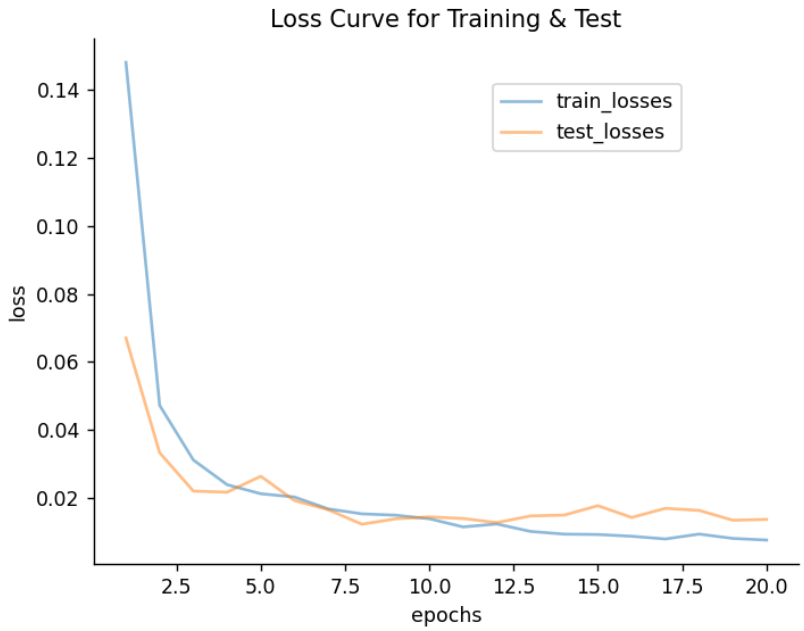

图 4 训练集和测试集的损失曲线

对于每个 mini-batch，其波动较大，但总体稳定在 0 附近。如图 5 所示，这是由于我们的采样是单点采样，未进行平滑处理，仅计算了对于每张图片所具有的 Loss 值，因此可见其有较多变化的点，每一个预测和正确结果不同的图片对都会产生较大的损失。

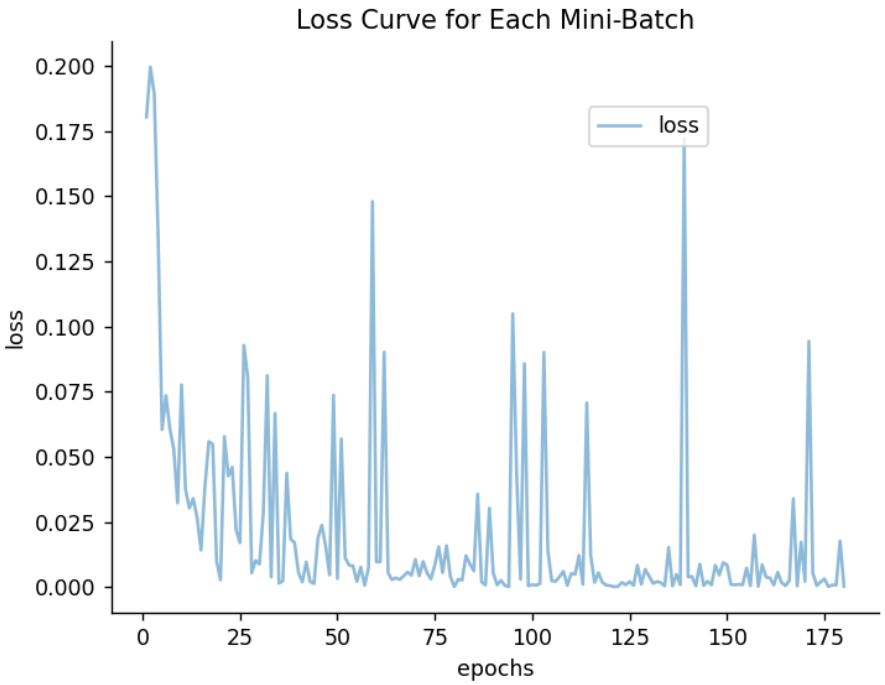

图 5 对于每个 mini-batch 的损失效果

1.4 讨论

我们使用孪生卷积神经网络测试了 MNIST 数据集上 10%数据作为训练集组的性能。其中，在卷积层内使用共享参数，通过相同的方式将图片对进行卷积后，通过全连接层，得到其嵌入向量。而后将嵌入向量做一次残差，得到其相差的欧式距离，以此对数字是否属于同一类进行学习。我们最后得到的效果较好，其正确率达到 99.70%。

为了进一步提高准确率，也可以更加优化网络架构，使用经典的 ResNet18 或 ResNet34 对于网络进行更换，并且仍然使用孪生网络参数共享的架构，使得网络可以对更多特征进行参考。