华中科技大学

课程实验报告

课程名称： 计算机视觉

实验名称： 实验二

院　　系：计算机科学与技术学院

专业班级：本硕博 2301 班

学　　号：U202315763

姓　　名：王家乐

2025 年 10 月 14 日

1. 任务要求

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**，输出为 1，否则为 0。从 MNIST 数据集的训练集中选取 10%作为训练图片，从 MNIST 数据集的测试集中选取 10%作为测试图片。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。

2. 实验设计

2.1 实验环境

项	参数
操作系统	Windows11
CPU	12th Gen Intel(R) Core(TM) i7-12700H
RAM	16GB
编程语言	Python3.10.18
平台	VScode & Jupyter Notebook

2.2 相关依赖

依赖项	版本
torch	2.8.0
pandas	2.3.1
numpy	2.2.6
matplotlib	3.10.5
torchvision	0.21.0
scikit-learn	1.7.2

2.3 配置信息

写一个 Cfg 类来管理本实验的相关配置，包括：数据集路径、batch 大小、训练轮数、学习率、权重衰退系数、log 保存路径、模型保存路径等。

```
class Cfg:
    data_root = './data'            # MNIST download/cache dir
```

```
    train_fraction = 0.10          # use 10% of the train split
    test_fraction  = 0.10          # use 10% of the test split
    train_num_pairs = 10000        # number of training pairs
    test_num_pairs  = 2000         # number of test pairs
    batch_size = 64
    epochs = 50
    lr = 1e-3
    weight_decay = 1e-4
    num_workers = 2
    save_dir = './output'
    model_name = 'siamese_mnist_same_digit.pth'
    epoch_log = 'epoch_log.csv'
```

2.4 数据集加载和处理

使用 torchvision.datasets.MNIST 加载数据集，按题目要求选取训练集 10%与测试集 10%。标准 MNIST 规模为 60000/10000，因此本实验实际使用 6000 张训练图片与 1000 张测试图片。使用 transforms 将像素值归一化并进行标准化。

```
# Transforms 预处理
transform = transforms.Compose([
    transforms.ToTensor(), # 归一化[0,1], shape (1,28,28)
    transforms.Normalize((0.1307,), (0.3081,)),
])

# 加载 MNIST 数据集
train_dataset = datasets.MNIST(Cfg.data_root, train=True,  download=True,
transform=transform)
test_dataset  = datasets.MNIST(Cfg.data_root, train=False, download=True,
transform=transform)
print(f"原始训练集大小: {len(train_dataset)}")
print(f"原始测试集大小: {len(test_dataset)}")

# 抽取 10%的数据
def sample_dataset(dataset, sample_ratio=0.1):
    indices = list(range(len(dataset)))
    sampled_indices = random.sample(indices, int(len(dataset) * sample_ratio))
    return Subset(dataset, sampled_indices)
train_subset = sample_dataset(train_dataset, Cfg.train_fraction)
test_subset = sample_dataset(test_dataset, Cfg.test_fraction)
print(f"采样后训练集大小: {len(train_subset)}")
print(f"采样后测试集大小: {len(test_subset)}")
```

继承 Dataset 类构建 MNISTPairsDataset 类，根据 Cfg 中的 train_num_pairs 和 test_num_pairs 从采样后的训练集和测试集随机生成比例为 1:1 的正负样本（正样本为相同数字的配对、负样本为不同数字的配对），并为正样本打上 Label=1 的标签，为负样本打上 Label=0 的标签。进而创建供 pytorch 使用的数据加载器。

```
# 创建配对数据集
class MNISTPairsDataset(Dataset):
    def __init__(self, subset, num_pairs):
        self.subset = subset
        self.num_pairs = num_pairs
        self.labels = [label for _, label in subset]
        self.images = [image for image, _ in subset]
        # 生成相同数字的索引映射
        label_to_indices = {}
        for idx, label in enumerate(self.labels):
            if label not in label_to_indices:
                label_to_indices[label] = []
            label_to_indices[label].append(idx)
        # 创建配对
        self.pairs = []
        self.pair_labels = []
        for _ in range(num_pairs // 2):
            # 正样本对
            label = random.choice(list(label_to_indices.keys()))
            if len(label_to_indices[label]) >= 2:
                idx1, idx2 = random.sample(label_to_indices[label], 2)
                self.pairs.append((idx1, idx2))
                self.pair_labels.append(1)
            # 负样本对
            label1, label2 = random.sample(list(label_to_indices.keys()), 2)
            idx1 = random.choice(label_to_indices[label1])
            idx2 = random.choice(label_to_indices[label2])
            self.pairs.append((idx1, idx2))
            self.pair_labels.append(0)

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

    def __getitem__(self, idx):
        idx1, idx2 = self.pairs[idx]
        x1 = self.images[idx1]
        x2 = self.images[idx2]
```

```
        y = self.pair_labels[idx]
        return x1, x2, torch.tensor(y, dtype=torch.float32)

# 创建训练和测试配对数据集
train_pairs_dataset = MNISTPairsDataset(train_subset, num_pairs=Cfg.train_num_pairs)
test_pairs_dataset = MNISTPairsDataset(test_subset, num_pairs=Cfg.test_num_pairs)
print(f"训练配对数据集大小: {len(train_pairs_dataset)}")
print(f"测试配对数据集大小: {len(test_pairs_dataset)}")

# 检查正负样本比例
train_labels = [label for _, _, label in train_pairs_dataset]
test_labels = [label for _, _, label in test_pairs_dataset]
print(f"训练集正样本比例: {sum(train_labels) / len(train_labels):.3f}")
print(f"测试集正样本比例: {sum(test_labels) / len(test_labels):.3f}")

# 数据加载器
train_loader = DataLoader(train_pairs_dataset, batch_size=Cfg.batch_size,
     shuffle=True, num_workers=Cfg.num_workers)
test_loader = DataLoader(test_pairs_dataset, batch_size=Cfg.batch_size,
     shuffle=False, num_workers=Cfg.num_workers)
```

终端输出的数据集相关信息如下：

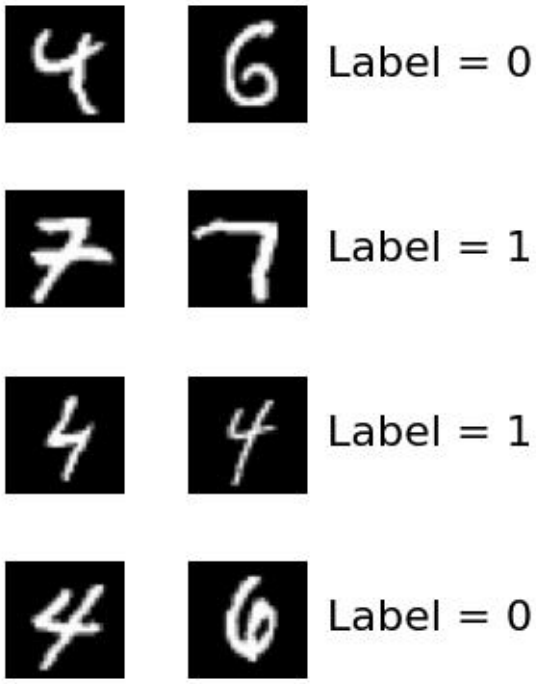
```
原始训练集大小: 60000
原始测试集大小: 10000
采样后训练集大小: 6000
采样后测试集大小: 1000
训练配对数据集大小: 10000
测试配对数据集大小: 2000
训练集正样本比例: 0.500
测试集正样本比例: 0.500
```

接下来使用 matplotlib 可视化部分配对样本。

```
def show_pairs(ds: MNISTPairsDataset, n=4):
    # 创建三列：左图、右图和标签
    fig, axes = plt.subplots(nrows=n, ncols=3, figsize=(3, 1*n))
    for i in range(n):
        x1, x2, y = ds[random.randint(0, len(ds)-1)]
        y = int(y.item())
        axes[i,0].imshow(x1.squeeze(0), cmap='gray')
        axes[i,0].axis('off')
        axes[i,1].imshow(x2.squeeze(0), cmap='gray')
        axes[i,1].axis('off')
        axes[i,2].text(0.5,0.5,f'Label = {y}',fontsize=16,ha='center',va='center')
        axes[i,2].axis('off')
    plt.tight_layout()
```

```
    plt.show()

show_pairs(train_pairs_dataset, n=4)
```

输出的图像如下，由此可见，数据集已创建成功。



2.5 孪生（Siamese）卷积神经网络设计

网络架构的核心采用共享权重的 Siamese 卷积网络，包含以下部分：

1) ConvEncoder：两层卷积(ReLU 激活)与 2×2 最大池化提取 28×28 灰度图的表征；随后为 Flatten 与全连接层映射到嵌入空间。

2) 特征融合：对两个嵌入向量 z_1、z_2计算$|z_1-z_2|$与 $z_1 \odot z_2$并连接。

3) 判别头：构建全连接神经网络输出二分类的 logit 并使用 Sigmoid 激活输出[0,1]之间的概率，大于 0.5 认为是正样本，否则认为是负样本。

```
class ConvEncoder(nn.Module):
    def __init__(self, emb_dim=64):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),   # 1x28x28 -> 16x28x28
```

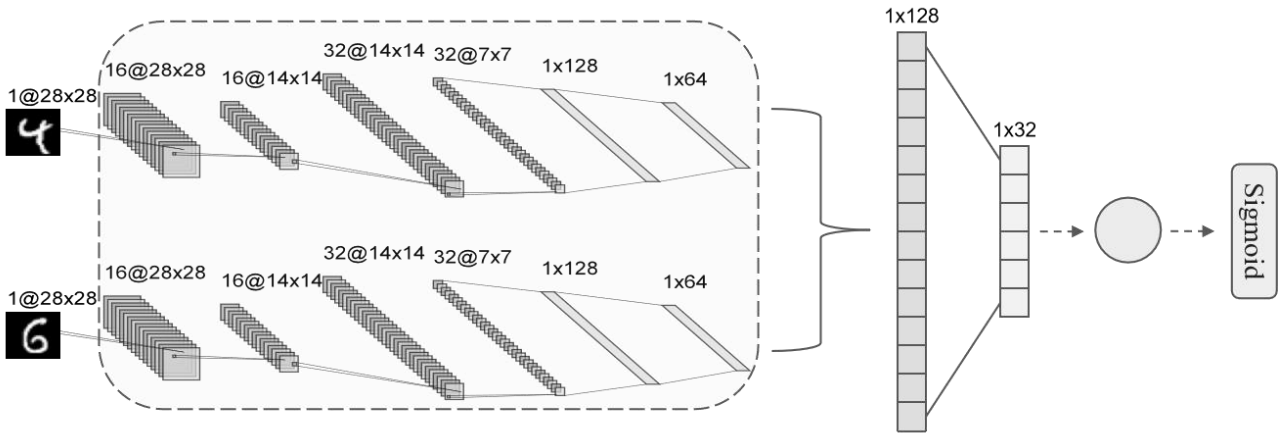
```
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),                                # -> 16x14x14
            nn.Conv2d(16, 32, kernel_size=3, padding=1),  # -> 32x14x14
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),                                # -> 32x7x7
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32*7*7, 128),
            nn.ReLU(inplace=True),
            nn.Dropout(0.2),
            nn.Linear(128, emb_dim),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.fc(x)
        return x

class SiameseSameDigit(nn.Module):
    def __init__(self, emb_dim=64):
        super().__init__()
        self.encoder = ConvEncoder(emb_dim=emb_dim)
        # 使用绝对差和元素级乘积组合特征
        self.head = nn.Sequential(
            nn.Linear(emb_dim*2, 32),
            nn.ReLU(inplace=True),
            nn.Linear(32, 1),  # logit
            nn.Sigmoid()  # 输出概率
        )

    def forward(self, x1, x2):
        z1 = self.encoder(x1)
        z2 = self.encoder(x2)
        feat = torch.cat([torch.abs(z1 - z2), z1 * z2], dim=1)
        logit = self.head(feat)
        return logit.squeeze(1)  # (B,)

model = SiameseSameDigit(emb_dim=64).to(device)
```

该模型的参数量为 218049，统计信息如下：

```
Layer Name                          Parameter Shape                 Param Count
=============================================================================
encoder.features.0.weight           [16, 1, 3, 3]                   144
encoder.features.0.bias             [16]                            16
```

```
encoder.features.3.weight          [32, 16, 3, 3]          4608
encoder.features.3.bias            [32]                    32
encoder.fc.1.weight                [128, 1568]             200704
encoder.fc.1.bias                  [128]                   128
encoder.fc.4.weight                [64, 128]               8192
encoder.fc.4.bias                  [64]                    64
head.0.weight                      [32, 128]               4096
head.0.bias                        [32]                    32
head.2.weight                      [1, 32]                 32
head.2.bias                        [1]                     1
================================================================================
Total Parameters: 218049
```

该 Siamese 卷积网络结构如下图所示：



remark:虚线部分实际为同一网络，二者共享权重，这里为了画图方便将其分开

2.6 网络训练

训练模型，最大迭代次数 num_epochs 为 50、批大小 batch_size 为 64、学习率 learning_rate 为 0.001、权重衰退系数 weight_decay 为 0.0001。本实验选用二分类交叉熵 BCELoss 作为损失函数（若使用 BCEWithLogitsLoss，则不需要 Sigmoid 层），选用带动量的随机梯度下降 SGD 作为优化器（Adam 收敛更快，10 个 epochs 即可）。训练过程中记录每个 epoch 在训练集和测试集上的损失和准确率，并将其写入 output 文件夹下 Cfg.epoch_log 文件中，最后将模型保存至 Cfg.model_name 文件中。

```
def evaluate(model, loader, loss_fn):
    model.eval()
    total_loss = 0.0
    total_correct = 0
    total = 0
    with torch.no_grad():
        for x1, x2, y in loader:
            x1, x2, y = x1.to(device), x2.to(device), y.to(device)
            outputs = model(x1, x2)
            loss = loss_fn(outputs, y)
            total_loss += loss.item() * y.size(0)
            preds = (outputs >= 0.5).float()
            total_correct += (preds == y.view_as(preds)).sum().item()
            total += y.size(0)
    return total_loss/total, total_correct/total

def train(model, train_loader, test_loader, epochs, lr, wd):
    loss_fn = nn.BCELoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=wd)
    history = {
        'epoch': [],
        'train_loss': [],
        'train_acc': [],
        'test_loss': [],
        'test_acc': []
    }
    for epoch in range(1, epochs+1):
        model.train()
        epoch_loss = 0.0
        epoch_correct = 0
        epoch_total = 0

        for b, (x1, x2, y) in enumerate(train_loader):
            x1, x2, y = x1.to(device), x2.to(device), y.to(device)

            optimizer.zero_grad()
            outputs = model(x1, x2)
            loss = loss_fn(outputs, y)
            loss.backward()
            optimizer.step()

            with torch.no_grad():
                preds = (outputs >= 0.5).float()
                correct = (preds == y.view_as(preds)).sum().item()
```

```
            epoch_loss += loss.item() * y.size(0)
            epoch_correct += correct
            epoch_total += y.size(0)

        train_loss = epoch_loss/epoch_total
        train_acc  = epoch_correct/epoch_total
        test_loss, test_acc = evaluate(model, test_loader, loss_fn)
        history['epoch'].append(epoch)
        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['test_loss'].append(test_loss)
        history['test_acc'].append(test_acc)
        print(f"[Epoch {epoch:2d}] "+
              f"train_loss={train_loss:.4f} train_acc={train_acc:.4f} | "+
              f"test_loss={test_loss:.4f} test_acc={test_acc:.4f}")

    hist_df = pd.DataFrame(history)
    csv_path = os.path.join(Cfg.save_dir, Cfg.epoch_log)
    hist_df.to_csv(csv_path, index=False)
    print(f"Training log (epoch) saved to {csv_path}")
    model_path = os.path.join(Cfg.save_dir, Cfg.model_name)
    torch.save(model, model_path)
    print(f"Model saved to {model_path}")

    return history
```

为了更为直观地反映网络训练全过程的性能变化和分类效果，在训练结束后通过 matplotlib 库对训练集和测试集的损失、准确率进行可视化展示。

```
def plot_results(history):
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))
    axes[0].plot(history['train_loss'], label='Train Loss')
    axes[0].plot(history['test_loss'], label='Test Loss')
    axes[0].set_xlabel('Epoch')
    axes[0].set_ylabel('Loss')
    axes[0].set_title('Loss Curves')
    axes[0].legend()
    axes[1].plot(history['train_acc'], label='Train Acc')
    axes[1].plot(history['test_acc'], label='Test Acc')
    axes[1].set_xlabel('Epoch')
    axes[1].set_ylabel('Accuracy')
    axes[1].set_title('Accuracy Curves')
    axes[1].legend()
    plt.tight_layout()
    plt.show()
```

3. 实验结果与分析

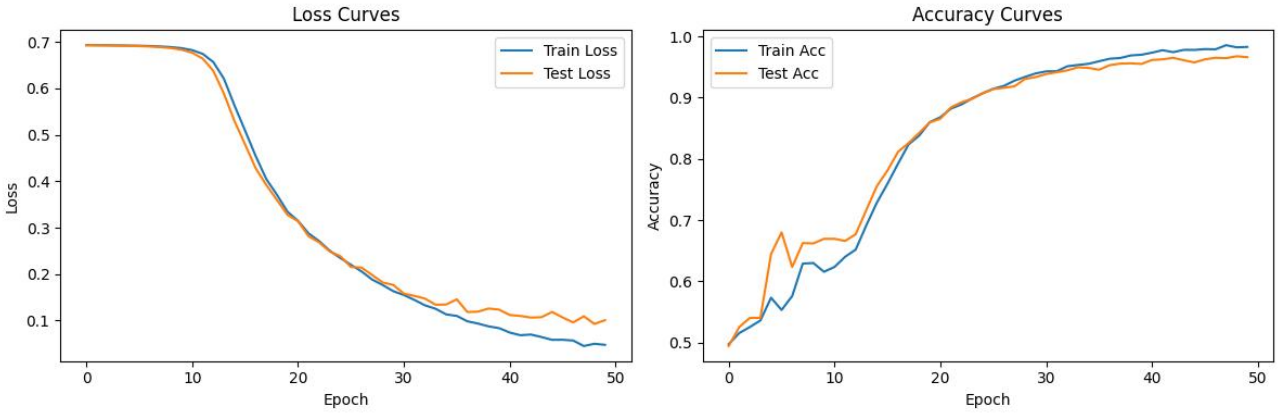
3.1 训练过程

最终模型在训练集上的准确率达到 98%以上，在测试集上的准确率达到 96%以上，训练过程每个 epoch 的 log 如下：

```
[Epoch  1] train_loss=0.6931 train_acc=0.4975 | test_loss=0.6930 test_acc=0.4945
[Epoch  2] train_loss=0.6929 train_acc=0.5157 | test_loss=0.6927 test_acc=0.5255
[Epoch  3] train_loss=0.6928 train_acc=0.5255 | test_loss=0.6924 test_acc=0.5405
[Epoch  4] train_loss=0.6925 train_acc=0.5363 | test_loss=0.6921 test_acc=0.5405
[Epoch  5] train_loss=0.6922 train_acc=0.5733 | test_loss=0.6916 test_acc=0.6445
[Epoch  6] train_loss=0.6919 train_acc=0.5534 | test_loss=0.6910 test_acc=0.6800
[Epoch  7] train_loss=0.6913 train_acc=0.5761 | test_loss=0.6902 test_acc=0.6235
[Epoch  8] train_loss=0.6904 train_acc=0.6291 | test_loss=0.6889 test_acc=0.6625
[Epoch  9] train_loss=0.6890 train_acc=0.6300 | test_loss=0.6868 test_acc=0.6620
[Epoch 10] train_loss=0.6867 train_acc=0.6157 | test_loss=0.6833 test_acc=0.6695
[Epoch 11] train_loss=0.6824 train_acc=0.6236 | test_loss=0.6769 test_acc=0.6695
[Epoch 12] train_loss=0.6743 train_acc=0.6402 | test_loss=0.6641 test_acc=0.6660
[Epoch 13] train_loss=0.6566 train_acc=0.6519 | test_loss=0.6376 test_acc=0.6770
[Epoch 14] train_loss=0.6207 train_acc=0.6914 | test_loss=0.5885 test_acc=0.7165
[Epoch 15] train_loss=0.5630 train_acc=0.7286 | test_loss=0.5293 test_acc=0.7555
[Epoch 16] train_loss=0.5090 train_acc=0.7593 | test_loss=0.4792 test_acc=0.7810
[Epoch 17] train_loss=0.4541 train_acc=0.7922 | test_loss=0.4280 test_acc=0.8115
[Epoch 18] train_loss=0.4047 train_acc=0.8237 | test_loss=0.3920 test_acc=0.8260
[Epoch 19] train_loss=0.3710 train_acc=0.8380 | test_loss=0.3599 test_acc=0.8425
[Epoch 20] train_loss=0.3345 train_acc=0.8599 | test_loss=0.3277 test_acc=0.8595
[Epoch 21] train_loss=0.3146 train_acc=0.8681 | test_loss=0.3138 test_acc=0.8655
[Epoch 22] train_loss=0.2876 train_acc=0.8821 | test_loss=0.2814 test_acc=0.8840
[Epoch 23] train_loss=0.2709 train_acc=0.8889 | test_loss=0.2682 test_acc=0.8925
[Epoch 24] train_loss=0.2502 train_acc=0.8988 | test_loss=0.2485 test_acc=0.8980
[Epoch 25] train_loss=0.2346 train_acc=0.9072 | test_loss=0.2386 test_acc=0.9070
[Epoch 26] train_loss=0.2200 train_acc=0.9144 | test_loss=0.2153 test_acc=0.9140
[Epoch 27] train_loss=0.2058 train_acc=0.9191 | test_loss=0.2141 test_acc=0.9160
[Epoch 28] train_loss=0.1883 train_acc=0.9277 | test_loss=0.1980 test_acc=0.9185
[Epoch 29] train_loss=0.1765 train_acc=0.9338 | test_loss=0.1819 test_acc=0.9305
[Epoch 30] train_loss=0.1632 train_acc=0.9395 | test_loss=0.1767 test_acc=0.9335
[Epoch 31] train_loss=0.1550 train_acc=0.9429 | test_loss=0.1579 test_acc=0.9385
[Epoch 32] train_loss=0.1443 train_acc=0.9431 | test_loss=0.1530 test_acc=0.9415
[Epoch 33] train_loss=0.1326 train_acc=0.9512 | test_loss=0.1472 test_acc=0.9445
[Epoch 34] train_loss=0.1253 train_acc=0.9533 | test_loss=0.1340 test_acc=0.9495
[Epoch 35] train_loss=0.1132 train_acc=0.9552 | test_loss=0.1346 test_acc=0.9485
```

```
[Epoch 36] train_loss=0.1099 train_acc=0.9594 | test_loss=0.1456 test_acc=0.9455
[Epoch 37] train_loss=0.0984 train_acc=0.9635 | test_loss=0.1183 test_acc=0.9530
[Epoch 38] train_loss=0.0935 train_acc=0.9647 | test_loss=0.1190 test_acc=0.9555
[Epoch 39] train_loss=0.0874 train_acc=0.9689 | test_loss=0.1259 test_acc=0.9560
[Epoch 40] train_loss=0.0835 train_acc=0.9700 | test_loss=0.1234 test_acc=0.9550
[Epoch 41] train_loss=0.0741 train_acc=0.9734 | test_loss=0.1119 test_acc=0.9615
[Epoch 42] train_loss=0.0684 train_acc=0.9774 | test_loss=0.1098 test_acc=0.9625
[Epoch 43] train_loss=0.0698 train_acc=0.9742 | test_loss=0.1063 test_acc=0.9650
[Epoch 44] train_loss=0.0645 train_acc=0.9780 | test_loss=0.1071 test_acc=0.9610
[Epoch 45] train_loss=0.0584 train_acc=0.9779 | test_loss=0.1185 test_acc=0.9575
[Epoch 46] train_loss=0.0585 train_acc=0.9792 | test_loss=0.1066 test_acc=0.9625
[Epoch 47] train_loss=0.0569 train_acc=0.9789 | test_loss=0.0959 test_acc=0.9650
[Epoch 48] train_loss=0.0450 train_acc=0.9856 | test_loss=0.1092 test_acc=0.9645
[Epoch 49] train_loss=0.0499 train_acc=0.9821 | test_loss=0.0927 test_acc=0.9675
[Epoch 50] train_loss=0.0476 train_acc=0.9828 | test_loss=0.1007 test_acc=0.9660
Training log (epoch) saved to ./output/epoch_log.csv
Model saved to ./output/siamese_mnist_same_digit.pth
```

可见，模型收敛到了较好的效果，绘制损失和准确率曲线如下：



训练过程呈现三个阶段：

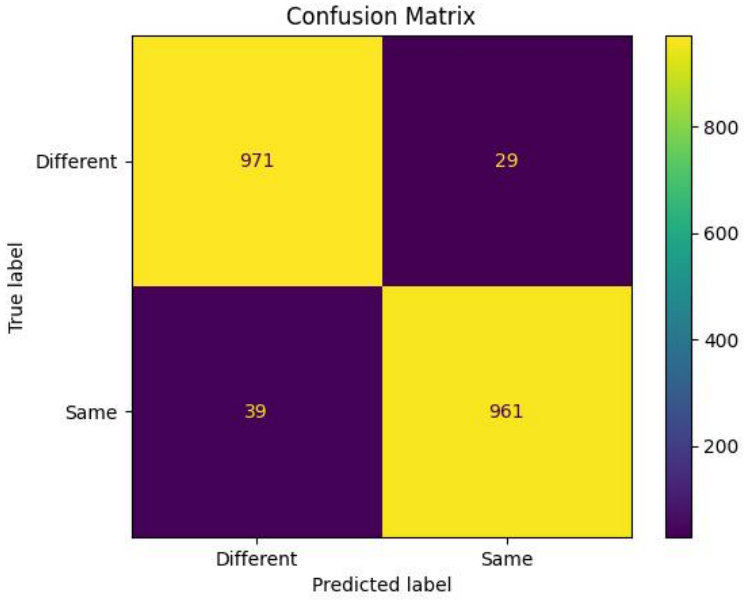
1) 冷启动/特征成形（Epoch 0–10）：Loss 较高；Acc 逐步上升。这是因为 lr=1e-3 + 轻度正则（Dropout、weight_decay）让早期更新较为保守，表现为“慢热”。

2) 快速收敛（Epoch 12–30）：Loss 明显陡降；Acc 快速上升。这是因为嵌入逐渐线性可分，融合特征（|z1−z2| + z1⊙z2）开始稳定向判别头提供有效信号。

3) 稳态收敛（Epoch 30–50）：Loss 缓慢下降并趋于稳定；Acc 逼近上限。未见明显过拟合现象。

3.2 混淆矩阵

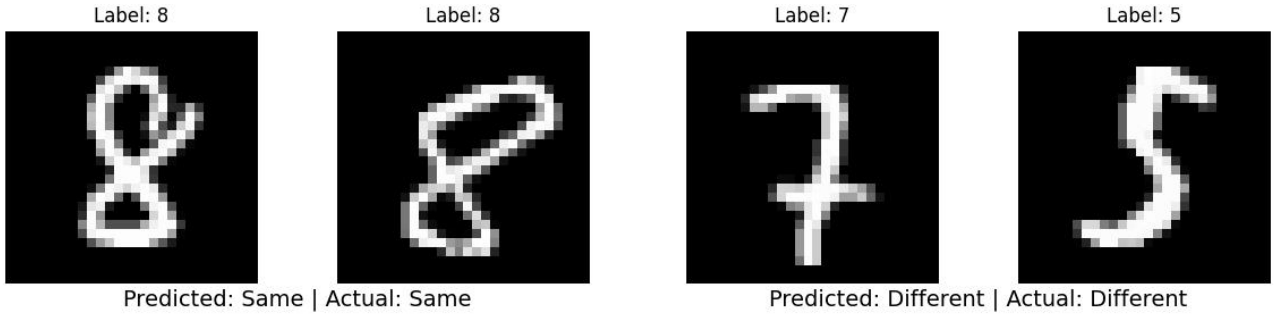
加载训练后的模型在测试集上预测，准确率为 96.6%，绘制混淆矩阵如下：



错误案例分析：对于笔画相似的不同数字（如'3'与'8'）或书写不规范的样本，Siamese 结构可能产生混淆。可考虑更强的编码器、更深的网络结构或数据增强提升稳健性。

3.3 模型预测

从原始的未处理的 MNIST 数据集中随机选择两张图片，根据他们的 Label 是否相同来确定是否为相同数字，再加载保存的模型进行预测，最终输出的值大于 0.5 则认为是相同数字，否则认为是不同数字。测试两组，结果如下：



4. 总结与体会

4.1 总结

本实验成功设计并实现了一个基于孪生（Siamese）结构的卷积神经网络，用于判断两张 MNIST 手写数字图片是否为同一数字。通过构建包含 10,000 对训练样本和 2,000 对测试样本的数据集，并采用共享权重的双分支编码器提取特征，结合差异特征与元素乘积进行融合，最终通过全连接层输出判别结果。实验结果表明，模型在测试集上达到了 96.6%的准确率，验证了所设计网络在数字图像对判别任务上的有效性。

4.2 体会

通过本次实验，我深入理解了孪生网络的结构特点及其在图像对匹配任务中的应用。实践中发现，特征融合方式（如绝对值差与点积结合）对模型性能具有关键影响，而适度的正则化策略（如 Dropout 与权重衰减）有助于提升模型泛化能力。然而，对于笔画相似或书写不规范的数字，模型仍存在一定的误判，未来可考虑引入更复杂的网络结构或数据增强方法以进一步提升鲁棒性。