华中科技大学

# 《计算机视觉导论》
# 上机实验（二）报告

## 题目： 基于卷积神经网络的两个数字比较



| | |
|---|---|
| 院　　系 | 计算机科学与技术学院 |
| 专业班级 | 计科 2301 班 |
| 姓　　名 | |
| 学　　号 | |
| 指导教师 | 李贤芝 |

计算机科学与技术学院

# 目　　录

# 1 问题定义与理解

## 1.1 问题描述

本次实验需要解决一个图像对相似性判断的二分类问题。与传统的图像分类任务不同，我们输入的不再是单张图片来判断其类别，而是同时输入两张独立的 MNIST 手写数字图片，要求设计的卷积神经网络（CNN）判断这两张图片所代表的数字类别是否相同。

这个判断的结果被编码为一个二元标签：如果两张图片所展示的是同一个数字（例如，两张图片都是 '7'，尽管它们是不同的书写样本），则输出标签为 1；反之，如果它们代表不同的数字（例如，一张是 '4'，另一张是 '9'），则输出标签为 0。

这实际上将传统的 10 个类别的分类问题（判断数字是 0 到 9 中的哪一个）转换成了一个更具挑战性的相似性学习问题。模型需要学会提取出足以区分数字类别的本质特征，并基于这些特征来判断任意两张手写数字图片在语义上是否一致。

提交的实验资料包中含有一个 viz_dataset.py 程序,用于展示数据集，如图 1-1 所示。

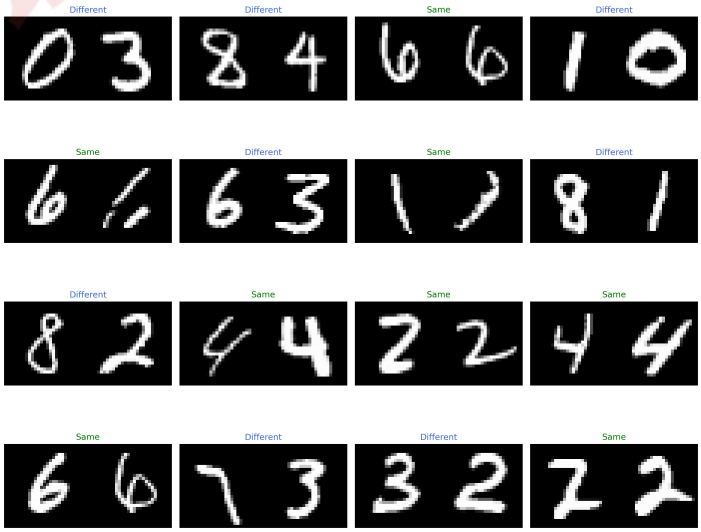


图 1-1 数据集示意图

## 1.2 相关理论基础

**卷积神经网络**(Convolutional Neural Networks, CNN)是一种深度学习算法，主要用于图像识别和处理领域。CNN 的设计模仿了人类视觉系统的工作机制，能够通过层次化的方式对图像中的局部特征和全局模式进行提取与分析，其应用已扩展到视频分析、自然语言处理等多个领域。

CNN 的核心思想在于通过**局部感知野**(Local Receptive Fields)和**权值共享**(Shared Weights)实现特征提取和数据的高效表示，其在图像处理任务中具有天然的优势，能够自动学习手写数字的关键视觉模式。

实验中设计的**特征编码器**（feature_extractor）采用了多层卷积、批量归一化（BatchNorm）和 ReLU 激活函数的组合。卷积层负责捕捉图像的局部特征，如笔画和边缘；批量归一化则确保了训练过程的稳定性和加速收敛；ReLU 引入非线性能力，使模型能够学习更复杂的映射。在卷积层之间交替使用的**最大池化**（MaxPool2d）操作，使模型对输入图像的微小平移和形变具有鲁棒性。最终，通过**自适应平均池化**（AdaptiveAvgPool2d）将多维特征图压缩为一个固定长度的特征嵌入向量，维度为 128。这个嵌入向量便是数字图像的“指纹”，是进行相似性度量的基础。

为了有效地解决图像对的相似性判断问题，通过参考相关资料，实验最终采用了**孪生神经网络**的架构思想。孪生网络由两个结构和参数完全相同的子网络构成，它们共享同一套权重。这种权值共享是孪生网络的核心，它强制两个输入 I1，I2 遵循相同的特征映射关系，从而保证了特征提取的公平性和一致性，使得模型学习到的嵌入空间能够直接用于衡量相似度：语义相似的输入在嵌入空间中距离近，不相似的输入则距离远。

实验代码中的 SiameseCompareNet 利用这一架构，将两个输入图片 分别编码为嵌入向量 e1，e2。 随后，模型并未直接使用简单的距离度量，而是采用了一种复杂的特征融合策略，通过拼接嵌入向量的绝对差值和元素乘积构建一个 2×128 维的比较特征。这种融合特征同时捕捉了特征的距离信息和对应维度上的一致性信息，为后续的分类决策提供了更丰富的判别依据。最后，一个多层全连接网络作为分类器，接收这个融合特征，并输出一个 Logit 值，该值通过 BCEWithLogitsLoss 损失函数进行优化，实现端到端的相似性二分类判断。

## 1.3 实验环境

为保证实验的可重复性与高效性，本实验在如下软硬件环境中进行：

显卡：NVIDIA GeForce RTX 4070 Laptop GPU

显存：8 GB GDDR6

内存：32 GB DDR5

CUDA 版本：12.3

操作系统：Windows 11 家庭版

Python 版本：3.11.11

深度学习框架：PyTorch 2.6.0

开发工具：Visual Studio Code (VSCode)

环境管理：Anaconda（虚拟环境隔离与依赖管理）

运行设备：本地 GPU 加速训练

# 2 数据分析及处理

## 2.1 数据集分析

MNIST 数据集是深度学习领域的经典基准数据集，包含 70000 张手写数字图像，其中训练集 60000 张，测试集 10000 张。每张图像都是 28×28 像素的灰度图，像素值范围为 0-255，标签为 0-9 共 10 个类别。数据集的特点是图像尺寸统一、背景干净、数字居中，但不同人的书写风格差异较大，存在倾斜、粗细、连笔等变化。从表 1-1 MNIST 类别数量分布（部分）可以看出，MNIST 数据集的类别分布基本均衡，每个数字约占 10%，这保证了模型训练时不会受到类别不平衡的影响。接下来我们可视化不同类别的样本，观察数据的多样性和特征差异。

表 1-1 MNIST 类别数量分布（部分）

| 数字 | 训练集数量/张 | 测试集数量/张 |
|---|---|---|
| 0 | 5923 | 980 |
| 1 | 6742 | 1135 |
| 2 | 5958 | 1032 |
| 3 | 6131 | 1010 |



## 2.2 数据处理

首先进行数据标准化，参数如下：

```
MNIST_MEAN = 0.1307
MNIST_STD = 0.3081
```

这些参数是从整个 MNIST 数据集计算得到的全局统计量。标准化将像素值从[0, 255]映射到标准正态分布，加速模型收敛。标准化公式为：

$$x_{norm} = \frac{x - \mu}{\sigma} = \frac{x - 0.1307}{0.3081}$$

这样处理后，输入数据的分布接近标准正态分布，有助于梯度下降的稳定性和收敛速度。

为了提升模型对书写风格变化的鲁棒性，我们需要进行数据增强，训练时对

图像进行随机变换

```
     transforms.RandomApply([
          transforms.RandomAffine(
              degrees=12,              # 随机旋转±12 度
              translate=(0.12, 0.12),  # 随机平移±12%
              scale=(0.9, 1.1),        # 随机缩放 90%-110%
              shear=8                  # 随机剪切±8 度
          )
       ], p=0.7),  # 70%概率应用

       transforms.RandomApply([
          transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 1.0))
       ], p=0.2),  # 20%概率应用

       transforms.ToTensor(),
       transforms.Normalize((MNIST_MEAN,), (MNIST_STD,))
   ])
```

这些变换包括：仿射变换（旋转、平移、缩放、剪切）以 70%的概率应用，高斯模糊以 20%的概率应用。这样的组合既保持了数字的可识别性，又增加了样本的多样性，模拟了真实场景中手写数字的自然变化。

## 2.3 数据加载

为了模拟实际应用中标注数据稀缺的场景，实验只使用 10%的训练集（6000 张）和 10%的测试集（1000 张）。为了确保实验的可重复性，代码使用缓存机制固定数据划分。`load_or_create_indices` 函数首先检查缓存文件是否存在，如果存在则直接加载，确保每次运行都使用相同的数据划分。如果缓存不存在，则使用固定的随机种子生成新的随机索引并保存。这种设计保证了实验的完全可重复性。

Siamese 网络的训练需要成对的数据。main.py 中定义了 `MNISTPairsDataset` 类，动态生成正负样本对。首先构建一个标签到样本索引的映射字典，然后在__getitem__时动态生成样本对。每次调用时，以 50%的概率生成同类对（标签为 1），以 50%的概率生成异类对（标签为 0），这确保了训练数据的完美平衡。

```
train_pairs_dataset = MNISTPairsDataset(
    train_subset, CONFIG['pairs_per_epoch'], seed=CONFIG['seed']
)
test_pairs_dataset = MNISTPairsDataset(
    test_subset, max(CONFIG['pairs_per_epoch'] // 3, 1), seed=CONFIG['s
eed'] + 1
)
```

```
print(f"训练集样本对数量: {len(train_pairs_dataset)}")
print(f"测试集样本对数量: {len(test_pairs_dataset)}")
```

最后，使用 PyTorch 的 DataLoader 实现高效的批量数据加载。DataLoader 使用多进程加载数据，每个 worker 进程都通过 `seed_worker` 函数设置独立的随机种子，保证数据加载的多样性同时维持可重复性。`pin_memory=True` 在 GPU 训练时会固定数据在内存中，加速 GPU 读取；`prefetch_factor` 控制每个 worker 预取的 batch 数量，`persistent_workers` 保持 worker 进程活跃，避免进程反复启动的开销。这些优化措施显著提升了数据加载的效率。

通过以上完整的数据处理流程，我们得到了经过标准化和增强的训练数据、固定划分的数据子集、平衡的样本对、以及高效的批量数据加载器，为后续的模型训练奠定了坚实基础。

# 3 模型构建与训练

## 3.1 模型定义

模型的定义位于 main.py 的 SiameseCompareNet 类中，它由三个关键部分组成：共享权重的特征提取器、将特征图降维的投影层，以及负责最终相似性判断的分类器。

### 3.1.1 特征提取器

特征提取器采用三层卷积神经网络，负责将 28×28 的输入图像编码为抽象的特征表示：

```
self.feature_extractor = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=5, padding=2),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.AdaptiveAvgPool2d(1)
)
```

第一层卷积将 1 通道的灰度图像转换为 32 通道特征图，使用 5×5 卷积核捕获较大的感受野，通过 padding=2 保持尺寸为 28×28。后续的 BatchNorm 和 ReLU 激活提升训练稳定性和非线性表达能力。MaxPool2d 将特征图降采样至 14×14，在减少计算量的同时增强对位置变化的鲁棒性。

第二层和第三层采用类似的结构，通道数逐步增加到 64 和 128，卷积核缩小为 3×3 以捕获更精细的特征。最后通过 AdaptiveAvgPool2d(1)将任意尺寸的特征

图压缩为 1×1，得到 128 维的全局特征向量。这种设计保证了网络对不同尺度特征的有效提取。

### 3.1.2 投影层与分类器

特征提取后，投影层将 128 维卷积特征映射到嵌入空间：

```
self.projection = nn.Sequential(
    nn.Flatten(),
    nn.Linear(128, embedding_dim),
    nn.ReLU(inplace=True)
)
```

Flatten 操作将特征展平为一维向量，Linear 层投影到目标维度（本实验为 128 维），ReLU 激活进一步增强非线性。这一步将卷积特征转换为适合度量学习的嵌入向量。分类器接收两个嵌入向量的组合特征，输出相似度判断：

```
.Linear(256, 64),
    nn.ReLU(inplace=True),
    nn.Dropout(0.2),
    nn.Linear(64, 1)
)
```

输入维度为 embedding_dim * 2（本实验为 256 维），因为后续会拼接两个特征向量的组合信息。分类器采用三层全连接网络，通道数逐步降低为 256→64→1，中间插入 Dropout 层（丢弃率分别为 0.4 和 0.2）防止过拟合。最后输出 1 个标量值作为相似度分数。

### 3.1.3 前向传播

模型的前向传播包含特征编码和相似度计算两个阶段。首先定义编码函数：

```
def _encode(self, x: torch.Tensor) -> torch.Tensor:
    features = self.feature_extractor(x)
    embedding = self.projection(features)
    return embedding
```

这个函数将输入图像通过共享的特征提取器和投影层，得到嵌入向量。关键的 forward 函数实现了完整的推理流程：

```
def forward(self, img1: torch.Tensor, img2: torch.Tensor) -> torch.Tens
or:
    embedding1 = self._encode(img1)
```

```
    embedding2 = self._encode(img2)
    features = torch.cat([torch.abs(embedding1 - embedding2), embedding
1 * embedding2], dim=1)
    logits = self.classifier(features)
    return logits.squeeze(dim=1)
```

两张输入图像分别通过相同的_encode 函数得到嵌入向量 embedding1 和 embedding2。这里的权重共享是 Siamese 网络的核心：无论输入的是哪张图像，都使用同一套参数进行特征提取。

特征融合采用双重策略：torch.abs(embedding1 - embedding2)计算特征的绝对差值，衡量两个向量的距离；embedding1 * embedding2 计算元素级乘积，衡量特征的相关性。将这两种信息在通道维度拼接后输入分类器，同时利用距离度量和相似性度量，增强了模型的判别能力。最后 squeeze 操作移除多余的维度，返回形状为[batch_size]的相似度分数。

## 3.2 训练过程设计

训练采用二元交叉熵损失函数，直接优化模型输出的概率分布与真实标签之间的差异，BCEWithLogitsLoss 内部包含 sigmoid 激活，相比先 sigmoid 再计算 BCE，这种实现在数值上更稳定，能有效避免极端值导致的梯度消失或溢出问题。

优化器选择 AdamW，它在 Adam 的基础上改进了权重衰减的实现方式：

```
optimizer = optim.AdamW(
    model.parameters(),
    lr=CONFIG['learning_rate'],
    weight_decay=CONFIG['weight_decay']
)
```

学习率设为 0.0008，权重衰减系数为 0.0001，通过 L2 正则化防止模型过拟合。AdamW 的自适应学习率机制能自动调整每个参数的更新步长，加速收敛。

学习率调度器采用 ReduceLROnPlateau 策略，根据验证集表现动态调整学习率：

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='min',
    factor=CONFIG['scheduler_factor'],
```

```
    patience=CONFIG['scheduler_patience'],
    min_lr=CONFIG['min_lr']
)
```

当测试损失连续 2 个 epoch 不下降时，学习率乘以 0.5，最低降至 0.00001。这种策略在训练后期能帮助模型跳出局部最优，进一步优化性能。

每个 epoch 的训练通过 train_epoch 函数完成。函数首先将模型设置为训练模式，这会启用 Dropout 和 BatchNorm 的训练行为。然后遍历所有 batch 进行训练。

数据移至 GPU 时使用 non_blocking=True 实现异步传输，与计算并行执行。zero_grad(set_to_none=True)比默认的清零方式更高效。如果启用混合精度训练，前向传播和反向传播需要在 autocast 上下文中执行：

```
if amp_enabled and device.type == 'cuda':
    autocast_ctx = torch.amp.autocast(device_type='cuda', dtype=torch.f
loat16)
    with autocast_ctx:
        logits = model(imgs1, imgs2)
        loss = criterion(logits, targets)
    scaler.scale(loss).backward()
    if max_grad_norm is not None:
        scaler.unscale_(optimizer)
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_nor
m)
    scaler.step(optimizer)
    scaler.update()
```

autocast 会自动将部分操作转换为 float16 执行，loss 通过 scaler 缩放后反向传播，防止 float16 精度下的梯度下溢。如果设置了梯度裁剪阈值（本实验为 1.0），在参数更新前会限制梯度范数，防止梯度爆炸。

# 4 实验结果与分析

## 4.1 实验结果

实验在 GPU（CUDA）环境下运行，模型总参数量约为 192,449，训练过程中启用了数据增强（随机仿射变换与模糊处理），以提升模型的泛化性能。在训练结束时，模型的性能指标如表 4-1 实验结果数据所示。

表 4-1 实验结果数据

| 指标 | 训练集 | 测试集 |
|---|---|---|
| 最佳准确率（Best Acc） | 0.978 | 0.9895 |
| 最佳损失（Best Loss） | 0.063 | 0.029 |
| 最佳 AUC | 0.9994 | 0.9995 |
| 最佳轮次（Best Epoch） | - | 第 24 轮 |

从结果可见，模型在测试集上达到 98.95% 的准确率 与 0.9995 的 ROC-AUC 分数，说明其在判断图像对是否属于同一类别的任务中表现出极强的区分能力。

## 4.2 训练过程分析

### 4.2.1 损失和准确率曲线演化

根据训练历史文件中的统计数据，模型在 25 个 epoch 内损失稳步下降，准确率持续提升。图 4-1 演化图展示了训练与测试集的损失曲线与准确率曲线。

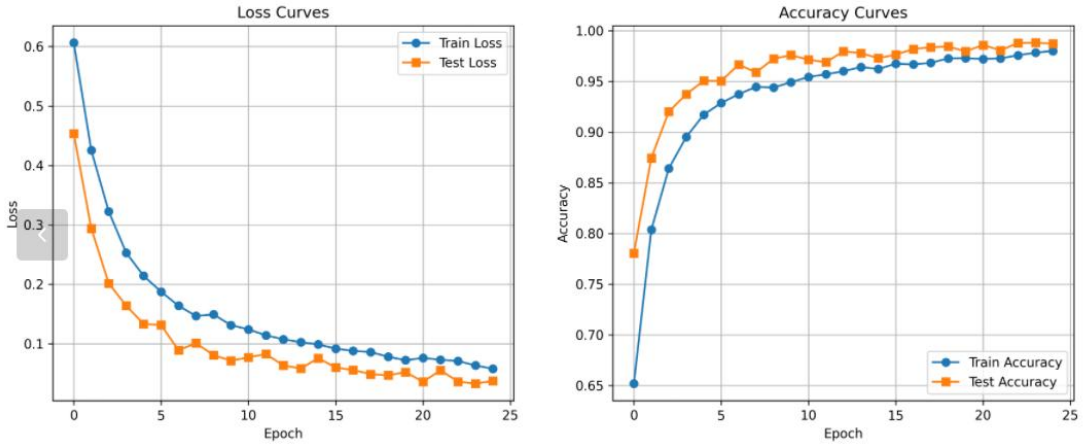


图 4-1 演化图

观察训练曲线（图 4-1）可以发现，模型在整个训练过程中表现出良好的收敛性。

**损失曲线** (Loss Curves)： 训练损失（Train Loss）和测试损失（Test Loss）在训练初期（前 10 个 Epoch）快速下降，随后趋于平稳。这表明模型迅速学会了识别特征和区分相似性。值得注意的是，测试损失的下降速度和最终值均优于训练损失（最佳测试损失为 0.0326，对应轮次的训练损失为 0.0631），这可能得益于我们在测试集上不使用 Dropout 层和更稳定的静态配对数据。

**准确率曲线** (Accuracy Curves)： 训练准确率和测试准确率均稳定上升，并在第 24 轮达到峰值 98.95%。虽然训练准确率略高于测试准确率（最终轮分别为 97.99% 和 98.70%），但两者差距微小，表明模型泛化能力优秀，并未出现明显的过拟合现象。

### 4.2.2 混淆矩阵分析

为了更直观地分析模型的预测效果，图 4-2 展示了测试集上的混淆矩阵。

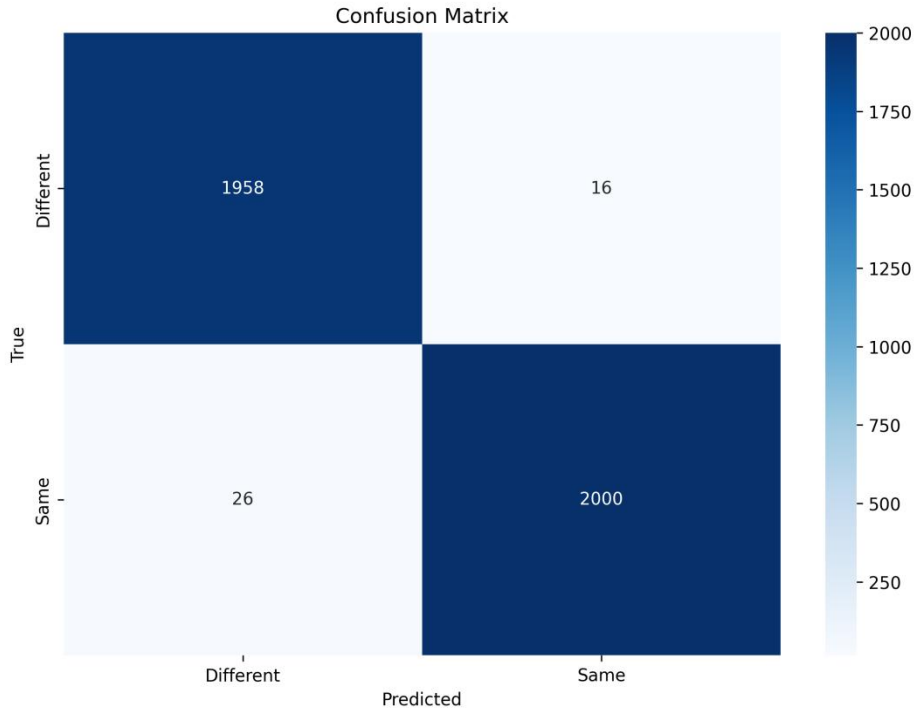


图 4-2 混淆矩阵

从混淆矩阵可见：真阳性（TP）和真阴性（TN）比例极高，错误预测主要集中在部分风格差异较大的数字（例如“1”与“7”、或“3”与“8”）；假阳性（FP）和假阴性（FN）极少，占比不足 1%。

这表明模型不仅在数值上准确率高，在结构上也学到了有效的相似度度量。

### 4.2.3 可视化测试分析

图 4-3 展示了测试集中部分样本对的预测结果（sample_predictions.png）。绿色标题表示预测正确，红色表示错误。


图 4-3 可视化预测结果

从可视化结果可以看出：对同一数字的不同书写形式（如两个“5”）模型能稳定输出高置信度的相似预测；对形状相近的不同数字（如“3”和“8”），模型仍能给出正确的“不相同”判断；少量误判多发生在笔画模糊、对称性强的样本之间，这说明模型对局部模糊或书写缺陷仍存在一定敏感性。

## 4.3 性能评估与结果讨论

总体来看，SiameseCompareNet 在 MNIST 相似度任务上表现出色：

**高精度与高 AUC**：测试准确率接近 99%，AUC 接近 1.0，说明模型几乎完美地区分了相同与不同数字对。

**收敛稳定**：训练过程平滑无震荡，验证集性能持续上升，表明优化器与学习率设置合理。

**泛化能力强**：测试损失持续低于训练损失，说明数据增强与正负样本均衡策略有效防止了过拟合。

**模型高效轻量**：参数量仅 19 万，训练速度快，适合小规模部署。

# 5 结论与可能改进

本次实验成功构建并训练了基于共享权重 CNN 编码器的孪生网络架构，用于 MNIST 数字图像对的相似性判断。模型在仅使用 10% MNIST 基础图片集的情况下，通过动态生成 12000 个训练样本对，最终在测试集上取得了 98.95% 的高准确率和 0.9994 的 ROC-AUC 值。这充分证明了孪生网络在处理相似性学习和度量学习任务上的高效性和卓越性能。

在实验过程中，卷积网络作为特征提取器成功捕捉到了数字的结构性差异，而双特征融合策略（$|f_1-f_2|$与 $f_1 \cdot f_2$的拼接）进一步增强了相似度度量的判别力。实验结果同时证明了混合精度训练（AMP）与 AdamW 优化器在中小型网络中的高效性，能够在保持精度的前提下降低训练时间与内存占用。

从训练曲线与混淆矩阵可以看出，模型在前 10 个 epoch 内快速收敛，并在后期稳定保持低误差，且几乎无明显过拟合现象。这说明数据增强（仿射变换与模糊）与样本对均衡生成机制（正负样本比例 1:1）对模型泛化性提升起到了关键作用。

实验存在的不足包括：模型主要依赖像素空间相似性，对复杂背景或旋转变化较大的情况可能性能下降；模型在极少数相似形状的数字（如“1”与“7”）上仍存在混淆；后续可考虑引入对比损失（Contrastive Loss）或 Triplet Loss，进一步提升对边界样本的区分能力。

另外，MNIST 数据集相对简单，且训练数据量较少。模型在复杂数据集（如 Fashion-MNIST 或 Omniglot）中的表现仍需验证。后续可考虑在更具挑战性的图像集上测试泛化性能。

未来的工作可从网络结构、损失函数及数据多样性等方面继续优化，使模型不仅在 MNIST 上取得高分，更能在实际复杂视觉场景中展现出稳健的相似性判断能力，从而为小样本视觉识别和嵌入学习提供坚实的技术基础。