MNIST 手写体字符识别实验报告

一、问题描述

MNIST 数据集是一个手写数字数据集，所有图像都是 28*28 大小的黑白手写数字，每一张图像都有一个 0-9 的数字作为标签，表明该手写数字实际是多少。

神经网络是机器学习方法的一种，可以在有标签的数据集上自动学习模型的参数，可以被用来做很多任务，比如图片分类任务，即输入是一张图片，输出图片对应标签。请设计并训练一个神经网络模型，完成对 MNIST 数据集的图片分类任务。

本项目基于 MNIST 数据集训练神经网络以实现对手写数字的识别。本项目还探究了学习率（lr）、训练轮次（num_epoch）、批次大小（batch_size）和模型结构对最终模型表现的影响。

二、实验过程

1、运行环境配置过程

为了完成 MNIST 手写体字符识别任务，我们首先配置了必要的运行环境，具体步骤如下：

1. **创建 Conda 环境**

   使用 Conda 创建一个独立的虚拟环境，以便隔离运行环境并避免与系统其他软件的依赖冲突。创建 Python 3.10 环境的命令如下：

2. conda create -n mnist_env python=3.10 -y
3. conda activate mnist_env
4. **安装必要的 Python 包**

   在激活的 Conda 环境中，安装实验所需的 Python 库，包括以下核心库：

   - torch 和 torchvision：用于深度学习模型的构建和数据处理。
   - numpy 和 pandas：用于数据的科学计算和管理。
   - matplotlib 和 seaborn：用于数据的可视化。
   - scikit-learn：用于数据预处理和模型评估。

安装命令如下：

```
pip install torch torchvision numpy pandas matplotlib scikit-learn seaborn
```

5. **配置 GPU 加速（可选）**

如果运行环境中提供了 NVIDIA GPU 并希望启用加速，可以配置 CUDA 和 cuDNN。具体步骤如下：

- 确保系统安装了支持 CUDA 12.1 的 NVIDIA 驱动。
- 安装对应版本的 CUDA 工具包（12.1）和 cuDNN（9.6.0）。
- 安装支持 GPU 的 PyTorch 版本：
- pip install torch torchvision --index-url https://download.pytorch.org/whl/cu121

6. **验证安装**

配置完成后，可以通过以下命令验证 PyTorch 是否正确配置 GPU 加速：

```
import torch
print(torch.cuda.is_available())  # 如果返回 True，表示 GPU 可用
print(torch.cuda.get_device_name(0))  # 输出 GPU 名称
```

以上步骤完成后，运行环境已成功配置，可以顺利进行 MNIST 手写体字符识别实验。

2、代码编写、模型训练

在本实验中，为了实现 MNIST 手写体字符的分类任务，我们进行了代码编写和模型训练，涵盖数据加载、网络设计、模型训练、参数调优等多个环节。以下是各部分的详细描述。

1. 数据处理

我们使用 PyTorch 提供的 torchvision 库加载了 MNIST 数据集，同时对数据进行了标准化和尺寸调整。关键步骤如下：

- **数据增强与标准化**：通过 transforms 定义了数据增强与标准化流程，将图片调整为 28x28 大小，归一化至均值为 0.5，标准差为 0.5 的分布。
- **加载数据**：分别加载训练集和测试集，使用 DataLoader 设置了批量加载（batch size 默认 32），并启用了打乱顺序的选项以提升训练效果。
- **数据可视化**：展示了前 10 张训练集图片及其对应标签，确保数据加载正确。

2. 模型定义

我们设计了多种全连接网络和卷积神经网络（CNN）以比较模型性能：

1. **全连接网络（FCN）结构**
   - SimpleNN: 包含两层全连接层，适用于基础模型测试。
   - MediumNN: 增加了一层隐藏层，以增强模型的学习能力。

- ComplexNN: 包含四层隐藏层，用于探索复杂模型在数据上的表现。

2. **卷积神经网络（CNN）** 我们设计了一个三层卷积网络，包含以下关键模块：
   - 三层卷积层：分别具有 32、64、128 个卷积核，使用 3×3 卷积核和 ReLU 激活函数。
   - 两层池化层：最大池化操作（2×2）。
   - 全连接层：将特征映射展平后通过两层全连接层最终输出分类结果。

3. 模型训练

训练流程设计：

- **优化器与损失函数：**
  - 优化器使用 Adam，学习率初始值设为 0.001。
  - 损失函数采用交叉熵损失（CrossEntropyLoss）。
- **训练过程：**
  - 每个模型训练过程中，通过 train() 模式启用梯度计算，逐批次完成前向传播、损失计算、反向传播及参数更新。
  - 为防止过拟合，每隔若干步保存一次模型检查点（checkpoint），最多保存 10 个。
- **验证与测试：**
  - 训练完成后，在测试集上评估模型性能，计算准确率、精确率、召回率和 F1 分数。

4. 参数调整与实验对比

为了找到最佳模型及超参数组合，我们系统性地调整了以下参数：

- **超参数范围：**
  - 学习率：0.0001,0.001,0.01,0.1
  - 训练轮数：2,5,10,15
  - 批量大小：16,32,64,128
- **对比实验：**
  - 通过对不同模型（SimpleNN、MediumNN、ComplexNN）在不同超参数组合下的实验，记录测试集上的准确率及损失值。
  - 结果存储为 CSV 文件，并通过排序选择准确率最高的三个实验作为最优结果。

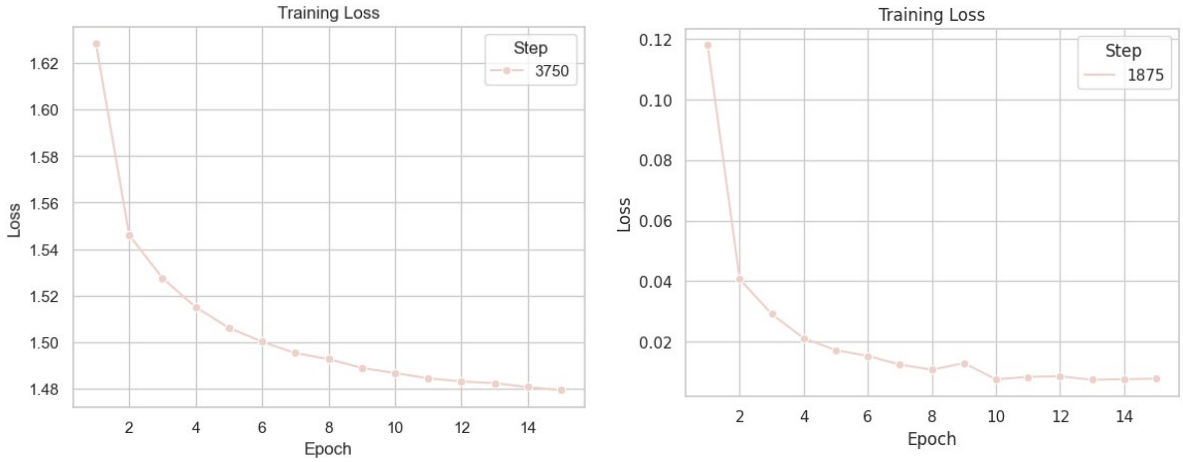
- 进一步选取性能最佳的模型与参数（MediumNN，学习率 0.0001，批量大小 16，训练轮数 15）进行训练与保存。

5. **最优模型测试与评估**

- **训练记录可视化：**
  - 使用 Seaborn 绘制训练损失随轮数变化的曲线，以验证模型收敛性。
- **单张图片预测：**
  - 加载保存的最佳模型，输入手写数字图片，对其类别进行预测并展示预测结果。

3、实验数据与分析

1. 全连接神经网络与卷积神经网络的训练曲线分析

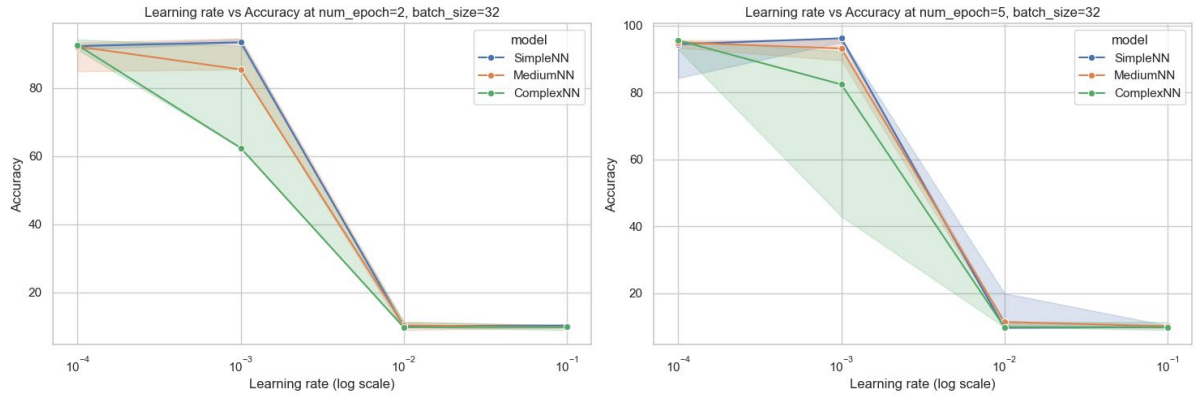


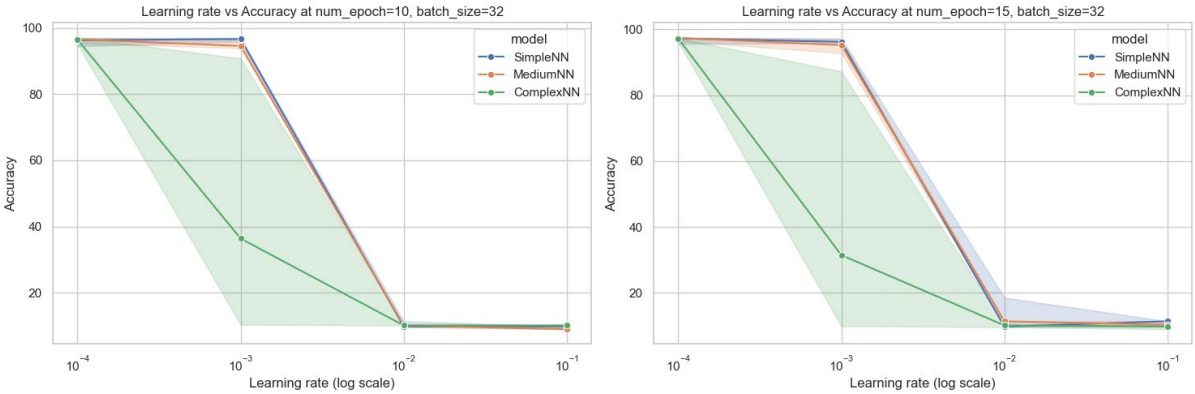
从全连接神经网络最优模型训练曲线（左图）可见，训练过程中，损失值随着轮次的增加持续下降，验证了模型的收敛性。初始阶段（Epoch 1），损失下降迅速，随后逐渐趋于平稳。最终训练损失趋近于 1.48。模型在设定的参数组合下（MediumNN，学习率 0.0001，批量大小 16）能够稳定收敛。

而卷积神经网络（CNN）（右图）的训练损失值相比全连接网络下降更快，最终训练损失显著更低。CNN 的复杂度提高使其对 MNIST 数据集更具表达能力，训练性能优于全连接网络。

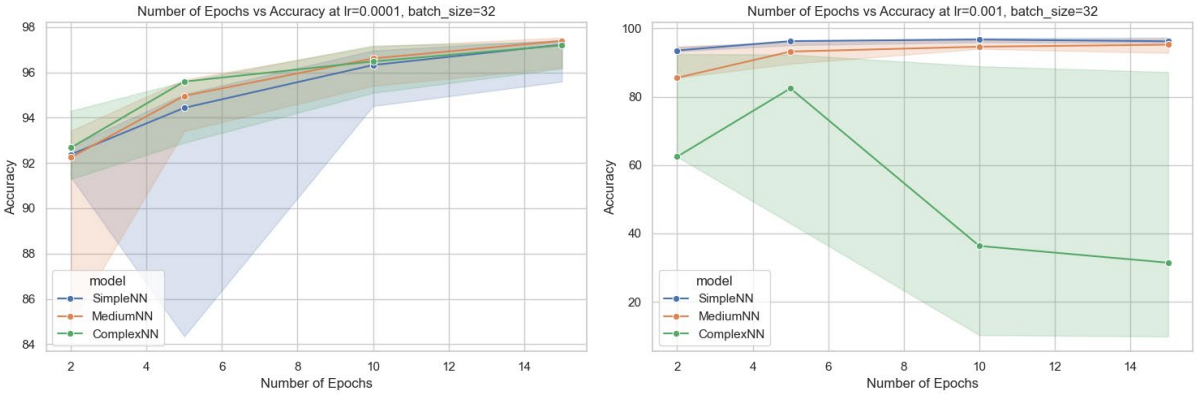
2. 学习率对模型性能的影响

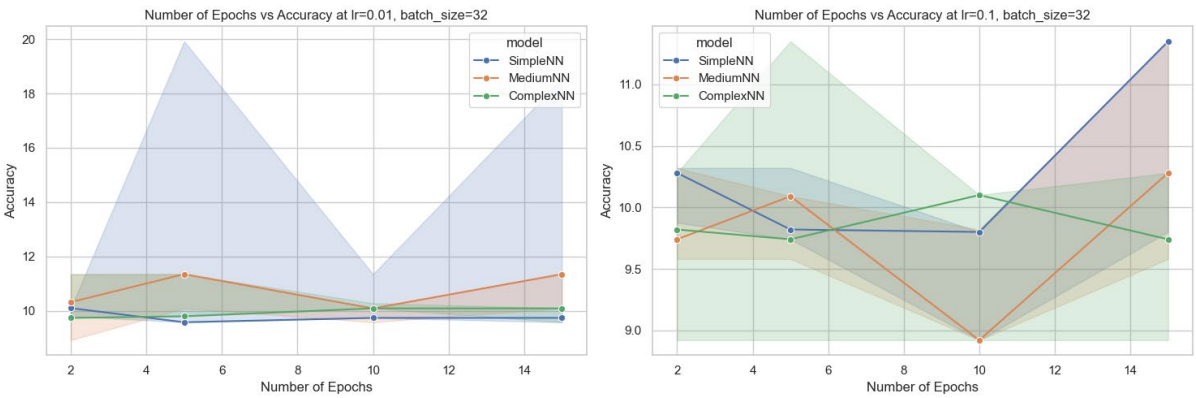
学习率对三种模型（SimpleNN、MediumNN、ComplexNN）的性能影响在不同学习率范围差异显著。在较低学习率（如 0.0001）下，所有模型均表现良好，准确率超过 95%。较高学习率（如 0.01 和 0.1）导致梯度更新震荡，模型难以收敛。因此学习率 0.0001 和 0.001 为较优选择，能够平衡训练效率与最终准确率。





3. 不同训练轮次对准确率的影响

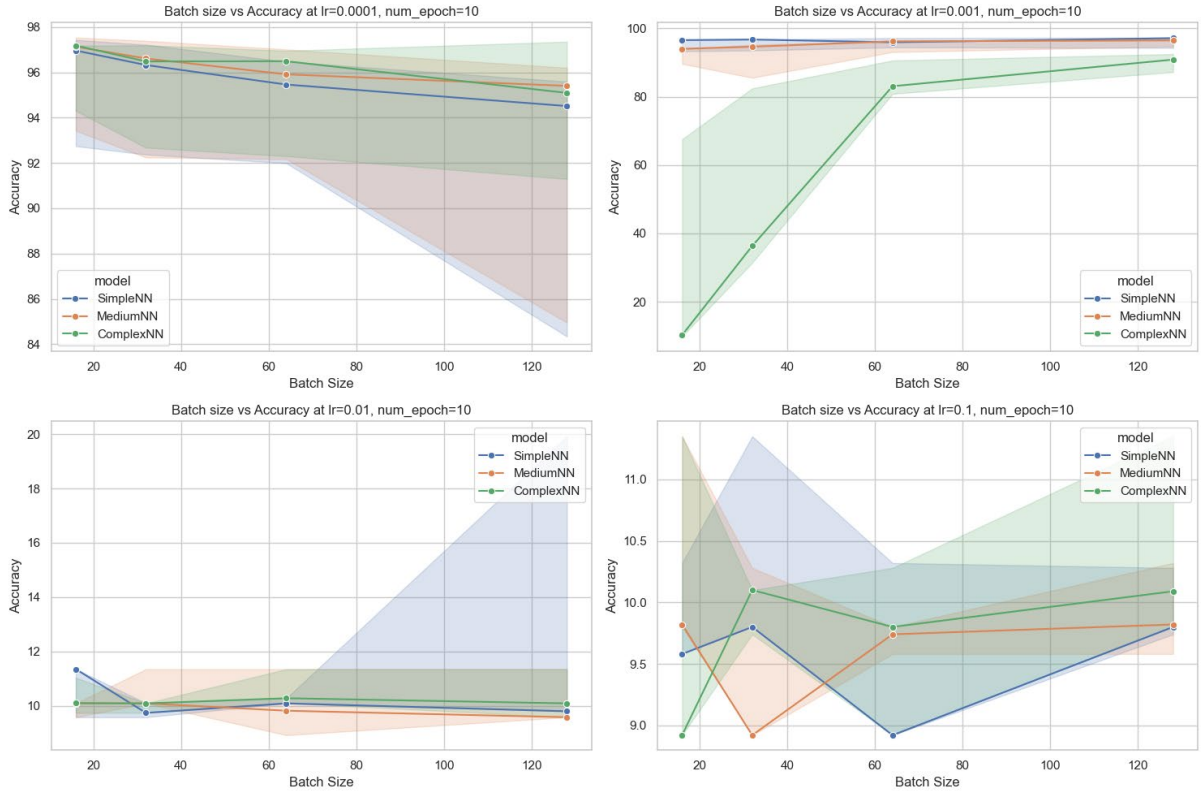




在学习率 0.001 和批量大小 32 的条件下，SimpleNN 和 MediumNN 随训练轮次增加，准确率稳定提高。ComplexNN 在训练轮次较小时准确率有所提升，但随着轮次继续增加，准确率出现下降，这表明模型发生了过拟合。在学习率 0.1 的情况下，增加

训练轮次并未有效提高模型性能，反而可能因过拟合或梯度爆炸导致性能下降。因此对于较复杂模型（如 ComplexNN），需设置足够的训练轮次以实现性能提升。

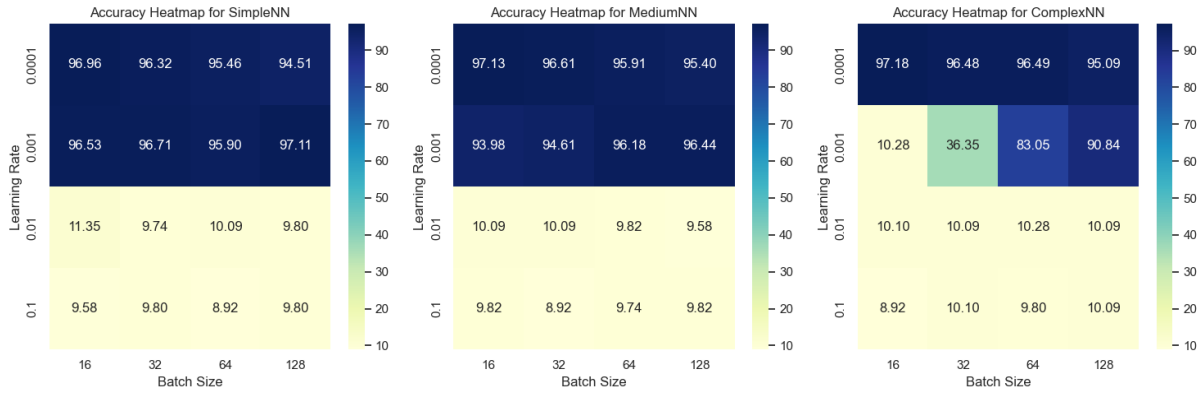
4. 批次大小对准确率的影响



在学习率较小（0.0001）条件下，批量大小较小时（16 和 32），准确率表现较高。批量大小增加（64 和 128）后，准确率略有下降，尤其对较复杂模型影响更显著。当学习率较大时（0.001），随批量大小增加，SimpleNN 和 MediumNN 的准确率略有下降，但 ComplexNN 的准确率却在稳定上升。

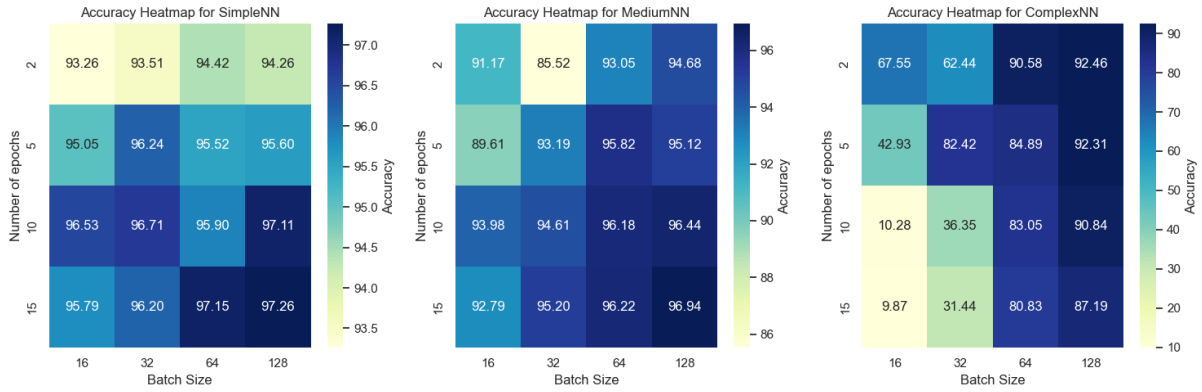
5. 热力图分析：学习率、批量大小与准确率的交互作用

5.1 学习率 vs 批量大小的影响（Epoch = 10）



SimpleNN 和 MediumNN 在低学习率和小批量大小的组合下（0.0001、16 和 32）表现最佳。ComplexNN 对学习率和批量大小更为敏感，其性能在低学习率（0.0001）和小批量大小（16 和 32）下达到峰值。

5.2 训练轮次 vs 批量大小的影响（Learning Rate = 0.001）



SimpleNN 和 MediumNN 随训练轮次和批量大小的合理搭配，准确率逐步提升。ComplexNN 需要更长训练轮次和中等批量大小组合才能充分表现。

当学习率很小时，每次参数更新的幅度都很小。小批量大小（例如 16 或 32）能够提供更多的梯度更新步骤。由于每次更新基于较少的数据，梯度中包含的噪声更大。这种噪声反而有助于模型跳出局部最优解，探索更广阔的参数空间。当批量大小较大（例如 128）时，每次梯度更新的依据是更多的样本，梯度变得更“光滑”且接近真实梯度，但是更新次数会减少。在较小的学习率下，这种更新可能会过于保守，模型更新方向的探索能力下降，容易陷入局部最优解，收敛速度也更慢。

当学习率较大时，每次参数更新的幅度也较大。 如果批量大小太小，梯度中的噪声可能会被放大，导致参数更新不稳定，难以收敛到极小值。较大的批量大小（例如 128）可以平滑梯度中的噪声，使得更新更加稳定。

三、实验过程中遇到的问题和解决方案

复杂模型过拟合

ComplexNN 在训练轮次较多（15）时，训练集损失显著低于测试集，且测试准确率波动较大，出现过拟合现象。选择较少的训练轮次（如 10）进行折中实验，降低过拟合风险。另外还可以采用正则化技术（如 Dropout）进一步缓解过拟合问题。增加数据增强操作，如随机旋转、平移等，来提高模型的泛化能力。

批量大小对性能的不一致性

在复杂模型（如 ComplexNN）的训练中，较大的批量大小（如 128）导致准确率显著下降，模型表现不稳定。通过对比小批量（16、32）与大批量（64、128）在不同模型中的表现，发现小批量在梯度更新的稳定性上更具优势。最终选择中等批量 32 作为复杂模型的推荐设置。

图片颜色通道问题，彩色图与灰度图的转换

在测试自定义图片时，图片预处理不匹配（如颜色通道问题），导致模型无法正确预测。解决方案是修改数据预处理流程，确保图片统一转换为灰度单通道格式，同时与训练数据一致。

四、总结反思

1.超参数调整的重要性，超参数对模型性能的影响显著。通过系统性调整超参数，可以显著提升模型表现。

2.模型复杂度与数据规模的匹配，较复杂的模型需要足够大的数据集才能充分发挥优势。对于小规模数据集，简单的模型能够避免过拟合。

3.卷积神经网络的优势，CNN 在处理图像数据时，能够捕获局部特征，表现优于全连接网络。模型架构设计应结合任务特性。