实验三：深度学习

实验安排

- **实验时间**：4学时

实验设计

- **设计思路**：本实验通过构建卷积神经网络（CNN）模型，并应用于手写数字识别任务，使学生深入理解CNN的基本原理，掌握其在图像分类中的实际应用，并培养学生的实践能力和问题解决能力。

实验目标

1. **理解CNN基本原理**：
   - 掌握卷积层、池化层、全连接层的作用和工作原理。
   - 理解CNN在图像特征提取和分类中的优势。
2. **实践CNN应用**：
   - 通过手写数字识别任务，实践CNN模型的构建、训练和评估。
   - 掌握数据增强、模型编译与训练等关键步骤。
3. **培养实践能力**：
   - 能够独立完成CNN模型的搭建和调试。
   - 能够对实验结果进行分析和优化。

实验环境

- **硬件**：
  - 显卡：NVIDIA GTX 1660或更高性能（确保支持CUDA加速）
  - 内存：至少8GB RAM
  - 存储：足够空间安装软件及存储数据集
- **软件**：
  - 操作系统：Windows 10/11或Linux（如Ubuntu 20.04）
  - 编程语言：Python 3.8+
  - 深度学习框架：TensorFlow 2.x（推荐2.4.0+）、 PyTorch（推荐1.7.0+）等
  - CUDA工具包：CUDA 11.x（与所选深度学习框架版本兼容）
  - 其他库：NumPy、Matplotlib等

实验题目

- **手写数字识别**：利用CNN模型对手写数字图像进行分类识别

实验数据

- **数据集**：MNIST手写数字数据集
  - 包含60,000个训练样本和10,000个测试样本。
  - 每个样本为28x28像素的灰度图像，标签为0-9的数字。

- **数据预处理**：
  - 归一化：将图像像素值归一化到0-1范围。
  - Reshape：将图像数据reshape为CNN模型输入要求的形状（如28x28x1）。
  - 数据增强（可选）：旋转、缩放、平移等，以提高模型泛化能力。
- **数据集获取**

1. **Kaggle**（需注册账号）：

- 链接：MNIST Dataset | Kaggle
- 说明：Kaggle提供了MNIST数据集的CSV格式版本，方便直接读取和处理。

2. **UCI Machine Learning Repository**：

- 链接：MNIST Dataset | UCI ML Repository
- 说明：UCI提供了MNIST数据集的原始格式下载，包括图像和标签文件。

3. **TensorFlow/Keras内置数据集**：

- 如果使用TensorFlow或Keras框架，可以直接通过代码加载MNIST数据集。
- 示例代码：

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# 加载MNIST数据集
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

4. **PyTorch内置数据集**：

- 如果使用PyTorch框架，同样可以通过代码直接加载MNIST数据集。
- 示例代码：

```
import torch
from torchvision import datasets, transforms

# 定义数据转换
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))]

# 加载MNIST数据集
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)
```

实验内容与步骤

1. **网络结构设计**：

- 选择TensorFlow或PyTorch等实现CNN模型。
- 构建包括卷积层、池化层、全连接层的网络结构。

  选择框架：自主选择TensorFlow或PyTorch等作为实现框架，例如：

- **TensorFlow实现**：

  卷积层：使用 `tf.keras.layers.Conv2D` ，设置卷积核大小（如3x3）、数量（如32个）和激活函数（如ReLU）。卷积层用于提取图像特征。
  池化层：使用 `tf.keras.layers.MaxPooling2D` 进行下采样，池化窗口大小通常为2x2。池化层用于降低特征图维度，减少计算量。
  全连接层：使用 `tf.keras.layers.Dense` 进行分类输出，输出节点数对应数字类别数（0-9，共10个）。全连接层用于将提取的特征映射到样本标记空间。

- **PyTorch实现思路**：

  卷积层：使用 `nn.Conv2d` ，设置输入通道数、输出通道数、卷积核大小等参数。
  池化层：使用 `nn.MaxPool2d` ，设置池化窗口大小和步长。
  全连接层：使用 `nn.Linear` ，根据卷积层和池化层的输出特征图大小，计算全连接层的输入节点数。

2. **数据增强**：

- **TensorFlow实现**：
  - 使用 `tf.keras.preprocessing.image.ImageDataGenerator` 进行数据增强，设置旋转范围、缩放范围等参数。
- **PyTorch实现思路**：
  - 使用 `torchvision.transforms` 进行数据增强，定义包括旋转、缩放、平移等在内的变换组合。

3. **模型编译与训练**：

- 使用训练集数据训练CNN模型。
- 监控训练过程中的损失值和准确率变化。

  例如：

- **TensorFlow实现**：

  模型编译：使用 `model.compile` 指定优化器（如Adam）、损失函数（如稀疏分类交叉熵 `sparse_categorical_crossentropy` ）和评估指标（如准确率 `accuracy` ）。
  模型训练：使用 `model.fit` 进行模型训练，设置合适的训练轮次（epochs）、批量大小（batch size）和验证集。监控训练过程中的损失值和准确率变化，及时调整模型参数或训练策略。

- **PyTorch实现思路**：

  定义损失函数和优化器：使用 `nn.CrossEntropyLoss` 作为损失函数，使用 `optim.Adam` 或 `optim.SGD` 作为优化器。
  训练循环：编写训练循环，包括前向传播、计算损失、反向传播和参数更新等步骤。使用验证集监控模型的泛化能力。

4. **模型评估**：

- **评估指标**：使用准确率、精确率、召回率、F1分数等指标全面评估模型性能。
- **评估方法**：采用测试集评估、交叉验证等方法确保评估结果的客观性。
- **评估过程**：准备测试数据，运行模型进行预测，计算评估指标，绘制混淆矩阵。
- **结果分析**：比较不同模型或参数设置下的性能差异，分析错误样本，提出改进方向。

实验要求

1. **模型构建与实现**：

   - 独立完成CNN模型的构建，包括卷积层、池化层、全连接层的设置和连接。
   - 编写完整模型训练代码，实现数据加载、预处理、模型编译/定义及优化器设置、训练和评估等步骤。
   - 明确选择使用TensorFlow、PyTorch等框架实现，并说明选择理由。

2. **实验报告撰写**：

   - 报告包括详细的实验内容与实现步骤，如模型构建实现，训练评估等。
   - 包括如模型结构图、训练曲线、测试准确率、性能分析、框架对比和实验总结等。
   - 详细阐述模型评估过程，包括评估指标、评估方法、评估结果和结果分析等
   - 详细分析总结实验过程中的问题及解决方法，及思考、改进方向等

   举例如下：

   - **模型结构图**：使用绘图工具绘制CNN模型结构图，清晰展示各层的名称、参数、连接方式等。
   - **训练曲线**：绘制训练过程中的损失值和准确率变化曲线，包括训练集和验证集（如使用）的曲线。分析曲线的变化趋势，评估模型的训练效果。
   - **测试准确率**：报告模型在测试集上的准确率，并与其他基准模型（如简单的全连接网络）进行对比，分析CNN模型的优势。
   - **性能分析**：对比不同网络层数（如增加或减少卷积层数量）、不同超参数（如学习率、批量大小）对模型性能的影响。分析模型复杂度与性能之间的关系，提出改进模型性能的建议。
   - **评估过程**：评估指标、评估方法、评估结果和结果分析。
   - **框架对比**：对比TensorFlow和PyTorch在实现CNN模型时的异同，包括API设计、训练流程、社区支持等方面。
   - **实验总结**：总结实验过程中的收获和体会，分析实验中遇到的问题和解决方法，提出改进方向。

3. **代码规范与可复现性**：

   - 确保代码结构清晰、注释充分、易于理解。
   - 提供完整的代码文件和数据集下载链接（或说明数据集获取方式），以便他人复现实验结果。

实验评分标准（参考）

- **模型性能**（40分）：
  - 测试准确率（20分）：模型在测试集上的准确率越高，得分越高。
  - 训练曲线稳定性（10分）：训练过程中损失值和准确率曲线平稳下降和上升，无明显波动或过拟合现象。
  - 模型复杂度（10分）：模型结构合理，既不过于简单导致欠拟合，也不过于复杂导致过拟合。
- **代码质量**（30分）：
  - 代码结构清晰（10分）：代码组织合理，函数和类定义明确，易于阅读和维护。
  - 注释充分（10分）：关键代码段和复杂逻辑有详细注释，解释代码功能和实现思路。
  - 可复现性（10分）：提供完整的代码和数据集，确保他人能够顺利复现实验结果。

- **实验报告**（30分）：
  - 报告完整性（5分）：实验内容及步骤的完整性和正确性。
  - 报告质量（5分）：结构清晰，表述简洁明了，格式规范，内容正确。
  - 模型结构图（5分）：清晰、准确，展示各层的名称、参数和连接方式。
  - 性能分析（5分）：深入合理分析不同网络层数和超参数对性能的影响。
  - 框架对比（5分）：简要对比TensorFlow、PyTorch等框架的异同，分析到位。
  - 实验总结（5分）：总结全面深刻，能够反映实验过程中的收获和体会。

注意事项

- **实验环境**：确保实验环境满足硬件和软件要求，特别是GPU的支持和CUDA工具包的安装。
- **数据预处理**：注意数据的归一化和reshape操作，确保输入数据符合CNN模型的输入要求。
- **模型训练**：合理设置模型参数，如卷积核大小、数量、训练轮次和批量大小等，以避免过拟合或欠拟合。可以使用验证集来监控模型的泛化能力。
- **实验报告**：认真撰写实验报告，确保报告内容完整、准确、清晰。特别是性能分析部分，要深入分析不同网络层数和超参数对性能的影响，并提出合理的改进意见。
- **代码规范**：遵循良好的编程习惯，确保代码结构清晰、注释充分、易于理解。提供完整的代码和数据集，以便他人复现实验结果。
- **框架选择**：根据个人兴趣及分析自主选择合适的深度学习框架，并熟悉其基本API和用法。