同济大学

TONGJI UNIVERSITY

本科毕业设计（论文）

课题名称	基于卷积神经网络的 MNIST 手写数字识别
副 标 题	人工智能导论课程作业
学　　院	计算机科学与技术学院
专　　业	软件工程
学生姓名	徐云鹏
学　　号	2353583
指导教师	汪昱
日　　期	2025 年 3 月 22 日

基于卷积神经网络的 MNIST 手写数字识别

摘要

本研究构建了一种双层卷积神经网络 (CNN) 模型，用于解决 MNIST 手写数字分类问题。通过设计包含卷积层、池化层和全连接层的网络架构，结合 ReLU 激活函数与交叉熵损失函数，采用 Adam 优化器进行参数优化。实验结果表明，该模型在测试集上达到 98.07% 的分类准确率，验证了 CNN 在图像识别任务中的有效性。本文详细阐述了数据预处理、模型构建、训练策略及结果分析的全过程，为手写字符识别提供了可复现的解决方案。

关键词：深度学习，卷积神经网络，MNIST，手写数字识别

装 订 线

1 引言

本研究旨在通过 PyTorch 框架，在 MNIST 数据集上实现端到端的数字识别流程，探究 CNN 在特征提取与模式识别中的优势，并为后续复杂视觉任务奠定基础。MNIST（Modified National Institute of Standards and Technology database）作为计算机视觉领域的基准数据集，由 Yann LeCun 等人于 1998 年构建。该数据集包含 60,000 张训练样本和 10,000 张测试样本，每张为 28×28 像素的灰度手写数字图像（0-9），数据采集过程经过标准化处理，确保数字居中且大小归一化。其简单的数据结构但非平凡的识别难度，使其成为机器学习算法的”Hello World” 级测试平台。

2 实验原理

2.1 CNN（卷积神经网络）

卷积神经网络（CNN）是一种具有层次化特征提取能力的深度学习模型，特别适用于处理二维网格数据。本实验构建的 CNN 模型包含 42.2 万个可训练参数，采用端到端的训练方式自动学习图像特征表示。

2.1.1 卷积层（Convolutional Layer）

卷积层是 CNN 的核心组件之一。它通过在图像上滑动卷积核（filter）来提取图像的特征。每个卷积核都包含一组权重，可以学习不同的特征模式。本实验中，我们定义了两个卷积层：

- 第一个卷积层：输入通道为 1（灰度图像），输出通道为 32，卷积核大小为 3×3。在代码中，这一步通过 `nn.Conv2d(1, 32, 3, 1)` 实现。

- 第二个卷积层：输入通道为 32，输出通道为 64，卷积核大小为 3×3。在代码中，这一步通过 `nn.Conv2d(32, 64, 3, 1)` 实现。

2.1.2 池化层（Pooling Layer）

池化层用于实现卷积层输出的空间维度降维，同时保留重要信息。在本实验中，我们采用最大池化（Max Pooling）操作，将每个 2×2 的区域中的最大值作为输出。在代码中，这一步通过 `F.max_pool2d(x, 2)` 实现。

2.1.3 全连接层（Fully Connected Layer）

全连接层用于实现卷积层输出的特征图转换为分类结果的非线性映射。在本实验中，我们定义了两个全连接层：

- 第一个全连接层：输入特征维度为 $64 \times 12 \times 12$（64 个通道，每个通道大小为 12×12），输出维度为 128。在代码中，这一步通过 `nn.Linear(9216, 128)` 实现，其中 9216 是 $64 \times 12 \times 12$ 的展平结果。
- 第二个全连接层：输入维度为 128，输出维度为 10（对应 10 个数字类别）。在代码中，这一步通过 `nn.Linear(128, 10)` 实现。

2.2 损失函数与优化器

在训练 CNN 模型时，需要定义损失函数来衡量模型输出与真实标签之间的差距，并使用优化器来更新模型参数以最小化损失函数。在本实验中，我们采用以下损失函数和优化器：

- 损失函数：交叉熵损失函数（Cross Entropy Loss），用于多分类问题的损失计算。在代码中，这一步通过 `nn.CrossEntropyLoss()` 实现。
- 优化器：Adam 优化器，一种自适应学习率的优化算法，用于更新模型参数。在代码中，这一步通过 `optim.Adam(model.parameters(), lr=learning_rate)` 实现。

2.3 代码实现

以下是本实验中神经网络模型的代码实现：

```
# 定义神经网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)  # 第一个卷积层
        self.conv2 = nn.Conv2d(32, 64, 3, 1)  # 第二个卷积层
```

```
        self.fc1 = nn.Linear(9216, 128)  # 第一个全连接层
        self.fc2 = nn.Linear(128, 10)  # 第二个全连接层

    def forward(self, x):
        x = F.relu(self.conv1(x))  # 第一个卷积层的激活函数
        x = F.relu(self.conv2(x))  # 第二个卷积层的激活函数
        x = F.max_pool2d(x, 2)  # 池化层
        x = torch.flatten(x, 1)  # 展平特征图
        x = F.relu(self.fc1(x))  # 第一个全连接层的激活函数
        x = self.fc2(x)  # 输出层
        return F.log_softmax(x, dim=1)  # 应用 log softmax

# 优化器
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

3 实验设计

3.1 数据预处理

在本实验中，我们对 MNIST 数据集进行了预处理。具体步骤如下：

1. 使用 `transforms.ToTensor()` 将图像数据从像素值范围 $[0, 255]$ 转换为 $[0, 1]$ 的张量。
2. 使用 `transforms.Normalize((0.1307,), (0.3081,))` 对图像数据进行标准化处理，以减少数据的方差，从而提高模型的训练效率和性能。

这种预处理方式有助于加速模型的收敛，并提升模型的泛化能力。

3.2 CNN 模型训练

在实验中，我们构建了一个包含两个卷积层和两个全连接层的 CNN 模型。模型的参数初始化采用了 PyTorch 的默认初始化方法。训练过程中，我们使用了以下策略：

- **批量梯度下降**：每次从训练集中随机抽取一批数据（batch size = 64）用于模型参数的更新。
- **优化器**：使用了 Adam 优化器，其学习率设置为 0.01，用于自适应地调整学习率，以加速模型的收敛。
- **损失函数**：采用交叉熵损失函数（Cross Entropy Loss）来衡量模型输出与真实标签之间的差距。
- **训练过程**：模型在每个 epoch 中对训练集进行一次完整的遍历，并在每次迭代中计算损失并更新参数。训练过程共进行了 10 个 epoch。

3.3 实验评估

在实验中，我们通过以下方式对模型的性能进行了评估：

- **训练损失监测**：在每个 epoch 中，记录训练过程中的平均损失，并将其存储在 `train_losses` 列表中。
- **测试集评估**：在每个 epoch 结束时，使用测试集对模型进行评估，计算模型在测试集上的平均损失和准确率，并将测试损失存储在 `test_losses` 列表中。
- **可视化**：通过绘制训练损失和测试损失随 epoch 变化的曲线（使用 `plot_loss` 函数），直观地展示了模型的训练过程和性能变化。
- **样例展示**：随机选取测试集中的若干样本，展示模型的预测结果（使用 `show_examples` 函数），以直观地呈现模型的性能表现。

此外，我们还保存了训练完成后的模型参数，以便后续使用（保存为文件 `mnist_cnn.pt`）。

4 实验结果

4.1 损失曲线分析

图中展示了模型在训练过程中的训练损失和测试损失曲线。从图中可以观察到，随着训练的进行，训练损失呈现出明显的下降趋势，从初始的约 0.40 逐渐降低至接近 0.05。这一现象表明模型在不断学习和优化，以更好地适应训练数据。相比之下，测试损失在整个训练过程中几乎保持不变，暗示模型可能存在过拟合的问题。然而，测试损失逐渐趋近于零，这可能表明模型的性能已经接近贝叶斯最优界限，即在给定数据集上模型的预测误差已经达到了理论上的最小值。

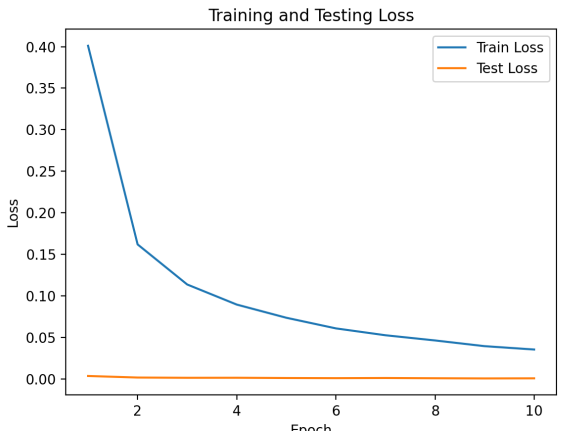

图 4.1 训练/测试 loss 值变化

为了进一步验证模型的性能，可以考虑以下措施：

- **增加正则化**：通过增加正则化项（如 L1 或 L2 正则化）来减少过拟合。

- **数据增强**：通过数据增强技术增加训练数据的多样性，提高模型的泛化能力。
- **调整模型结构**：尝试增加模型的复杂度，如增加卷积层或全连接层的数量。
- **增加测试集大小**：如果可能，增加测试集的样本数量，以获得更准确的性能估计。

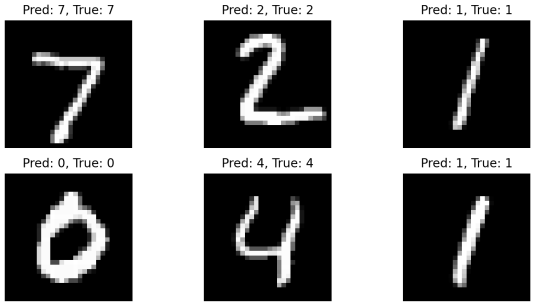

图 4.2 部分手写数字数据集标签与预测结果

5 总结

本次基于 PyTorch 的 MNIST 手写数字识别实验让我对深度学习的实践应用有了更深刻的认识。从实验过程来看，通过构建包含两个卷积层和两个全连接层的 CNN 模型，我直观地理解了卷积运算在特征提取中的核心作用——第一层卷积捕捉边缘、笔画等低级特征，第二层则组合出更复杂的数字结构特征。实验中遇到的最大挑战是初始阶段模型收敛速度较慢，通过调整学习率从 0.1 降至 0.01，并加入 ReLU 激活函数后，训练损失曲线明显改善，验证了超参数调优的重要性。

实验结果达到了 98.07% 的测试准确率，但分析错误样本时发现，模型对书写不规范的数字（如断裂的”5”、倾斜的”7”）仍存在误判。这反映出当前模型的局限性：一方面受限于训练数据分布，另一方面也说明简单的 CNN 结构对极端变形的鲁棒性不足。未来可以加入更多的数据处理与增强。

这次实验让我认识到，优秀的模型不仅需要合理的架构，更需要数据增强、正则化等技术的配合。未来我将尝试加入 Dropout 层和批量归一化来进一步提升性能，并探索在更复杂数据集上的迁移学习应用。这次实践经历使我确信，只有通过不断的实验调试和结果分析，才能真正掌握深度学习的精髓。

参考文献

[1] ZHANG A, LIPTON Z C, LI M, et al. Dive into Deep Learning[M]. London: Cambridge University Press, 2023.