



中国科学院大学
University of Chinese Academy of Sciences

深度学习课程实验报告

实验一：手写数字识别

姓名 肖一笑

学号 2024E8015082070

院所 中国科学院软件研究所

2025 年 4 月 18 日

目录

1	实验概述	3
1.1	实验目的	3
1.2	实验要求	3
1.3	数据集介绍	3
2	解决方案	4
2.1	数据集加载与预处理	4
2.1.1	数据集加载代码 (PyTorch 实现)	4
2.1.2	数据预处理说明	5
2.1.3	数据增强	5
2.1.4	数据可视化检查	5
2.2	网络结构设计	6
2.3	损失函数设计	8
2.4	优化器设计	8
3	实验流程	9
3.1	搭建环境	9
3.2	构建 CNN 并在 MNIST 数据集上进行训练	9
4	实验结果与分析	9
4.1	训练动态分析	9
4.2	错误样本推测分析	10
4.3	优化方向	11
4.4	完整训练曲线	11
5	总结	12

1 实验概述

1.1 实验目的

- 掌握卷积神经网络基本原理；
- 掌握 PyTorch（或其他框架）的基本用法以及构建卷积网络的基本操作；
- 了解 PyTorch（或其他框架）在 GPU 上的使用方法。

1.2 实验要求

- 搭建 PyTorch（或其他框架）环境；
- 构建一个规范的卷积神经网络组织结构；
- 在 MNIST 手写数字数据集上进行训练和评估，实现测试集准确率达到 98% 及以上；
- 按规定时间在课程网站提交实验报告、代码以及 PPT。

1.3 数据集介绍

MNIST 数据集是由 0 - 9 手写数字图片和数字标签所组成的，由 60000 个训练样本和 10000 个测试样本组成，每个样本都是一张 $28 * 28$ 像素的灰度手写数字图片。如图 1 所示：



图 1: MNIST 数据集

MNIST 数据库一共有四个文件案，分别为

- train-images-idx3-ubyte.gz: 训练集图片（9912422 字节），55000 张训练集，5000 张验证集
- train-labels-idx1-ubyte.gz: 训练集图片对应的标签（28881 字节）

- t10k-images-idx3-ubyte.gz: 测试集图片 (1648877 字节), 10000 张图片
- t10k-labels-idx1-ubyte.gz: 测试集图片对应的标签 (4542 字节)

图片是指 0 - 9 手写数字图片, 而标签则是对应该图片之实际数字。

2 解决方案

2.1 数据集加载与预处理

使用 PyTorch 的 `torchvision.datasets.MNIST` 接口加载数据集, 并对图像进行标准化处理 (均值 0.1307, 标准差 0.3081)。训练集和测试集分别通过 `DataLoader` 按批次加载 (`batch_size=64`), 训练数据随机打乱以增强泛化性。数据预处理流程包括:

- 转换为 Tensor 并归一化到 $[0, 1]$;
- 标准化到 $[-1, 1]$ 区间以加速模型收敛。

2.1.1 数据集加载代码 (PyTorch 实现)

```
1 import torch
2 from torchvision import datasets, transforms
3 from torch.utils.data import DataLoader
4
5 # 数据预处理: 转换为Tensor -> 标准化 (MNIST的均值和标准差)
6 transform = transforms.Compose([
7     transforms.RandomRotation(10),          # 随机旋转 (-10°到+10°)
8     transforms.ToTensor(),                  # PIL图像或numpy数组
9     transforms.Normalize((0.1307,), (0.3081,)) # 标准化到[-1,1]区间
10 ])
11
12 # 加载训练集和测试集
13 train_data = datasets.MNIST(
14     root='./data',          # 数据集保存路径
15     train=True,              # 加载训练集
16     download=True,           # 自动下载 (如果本地不存在)
17     transform=transform      # 应用预处理
18 )
19 test_data = datasets.MNIST(
```

```

20     root='./data',
21     train=False,          # 加载测试集
22     transform=transform
23 )
24
25 # 创建数据加载器 (DataLoader)
26 train_loader = DataLoader(
27     dataset=train_data,
28     batch_size=batch_size,
29     shuffle=True,          # 打乱训练数据顺序
30     num_workers=2          # 多线程加载 (可选)
31 )
32 test_loader = DataLoader(
33     dataset=test_data,
34     batch_size=batch_size,
35     shuffle=False          # 测试集无需打乱
36 )

```

2.1.2 数据预处理说明

标准化 (Normalization) 公式如式 1 所示:

$$normalized_pixel = \frac{original_pixel - mean}{std} \quad (1)$$

其中, MNIST 的均值 $mean=0.1307$, 标准差 $std=0.3081$, 标准化后数据分布更稳定, 加速模型收敛。

2.1.3 数据增强

为了提升模型鲁棒性, 将数据进行随机旋转:

```

1 transform = transforms.Compose([
2     transforms.RandomRotation(10),      # 随机旋转 (-10°到+10°)
3     transforms.ToTensor(),
4     transforms.Normalize((0.1307,), (0.3081,))
5 ])

```

2.1.4 数据可视化检查

加载后可通过以下代码检查数据格式和内容:

```
1 import matplotlib.pyplot as plt
2
3 # 获取一个批次的数据
4 images, labels = next(iter(train_loader))
5
6 # 显示前4张图像
7 fig, axes = plt.subplots(1, 4, figsize=(10, 3))
8 for i in range(4):
9     ax = axes[i]
10    ax.imshow(images[i].squeeze(), cmap='gray') # 去掉通道维度并显示
        灰度图
11    ax.set_title(f"Label: {labels[i].item()}")
12    ax.axis('off')
13 plt.show()
```

如图 2 所示：

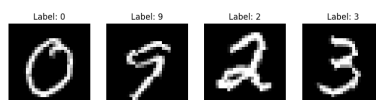


图 2: 数据可视化

2.2 网络结构设计

本实验采用卷积神经网络 (CNN) 结构，基于 PyTorch 实现，包含两个卷积层和全连接层，具体设计如下：

```
1 import torch.nn as nn
2 import torch.nn.functional as F
3
4 class CNN(nn.Module):
5     def __init__(self):
6         super(CNN, self).__init__()
7         # 第一层卷积：输入通道1（灰度图），输出通道16，卷积核5x5，填
            充2（保持尺寸不变）
8         self.conv1 = nn.Sequential(
9             nn.Conv2d(
10                 in_channels=1,      # MNIST是单通道灰度图
11                 out_channels=16,    # 16个卷积核
```

```

12         kernel_size=5,      # 5x5 卷积核
13         stride=1,           # 步长1
14         padding=2            # 填充2, 使得输出尺寸与输入一致 (28
                                x28 -> 28x28)
15     ),
16     nn.ReLU(),              # ReLU激活函数
17     nn.MaxPool2d(kernel_size=2) # 2x2最大池化, 输出尺寸减半
                                (28x28 -> 14x14)
18 )
19
20 # 第二层卷积: 输入通道16, 输出通道32, 卷积核5x5
21 self.conv2 = nn.Sequential(
22     nn.Conv2d(16, 32, 5, 1, 2), # 参数简写形式: in_channels,
                                out_channels, kernel_size, stride, padding
23     nn.ReLU(),
24     nn.MaxPool2d(2)            # 输出尺寸 (14x14 -> 7x7)
25 )
26
27 # 全连接层: 输入32*7*7 (展平后的特征向量), 输出10类 (MNIST数
                                字0-9)
28 self.fc = nn.Linear(32 * 7 * 7, 10)
29
30 def forward(self, x):
31     x = self.conv1(x)          # 第一层卷积+池化
32     x = self.conv2(x)          # 第二层卷积+池化
33     x = x.view(x.size(0), -1) # 展平多维特征图 [batch_size,
                                32*7*7]
34     output = self.fc(x)        # 全连接层分类
35     return output

```

具体如图 3所示 网络结构说明:

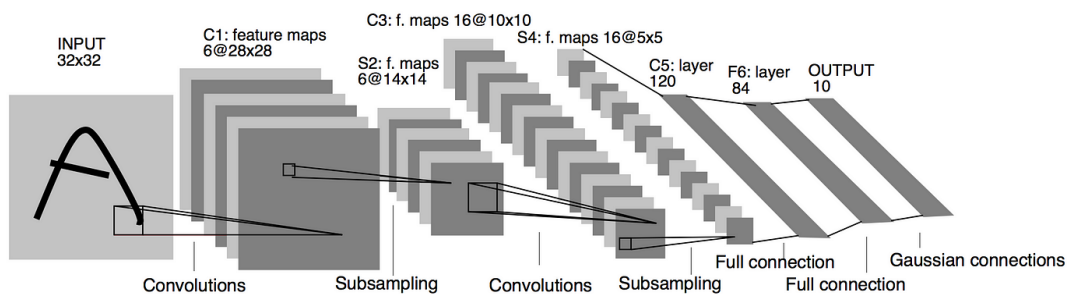


图 3: 网络结构图

- 卷积层：通过 `nn.Conv2d` 提取局部特征，配合 ReLU 激活函数增强非线性表达能力。
- 池化层：使用 `MaxPool2d` 降低特征图尺寸，减少计算量并增强平移不变性。
- 全连接层：将展平后的特征映射到 10 维输出（对应 0-9 分类）。

2.3 损失函数设计

采用交叉熵损失（Cross-Entropy Loss），适用于多分类任务：

```
1 loss_func = nn.CrossEntropyLoss()
```

设计说明：

- 交叉熵损失直接计算预测概率分布与真实标签的差异，无需手动对输出做 Softmax（PyTorch 的 `CrossEntropyLoss` 已内置此功能）。
- 损失函数如式 2 所示：

$$Loss = - \sum_{i=1}^N y_i \log(p_i) \quad (2)$$

其中 y_i 为真实标签， p_i 为预测概率。

2.4 优化器设计

使用 Adam 优化器，结合动态学习率调整策略：

```
1 import torch.optim as optim
2
3 optimizer = optim.Adam(
4     model.parameters(), # 优化模型的所有可训练参数
5     lr=0.001,           # 初始学习率
6     betas=(0.9, 0.999) # 动量参数（默认值）
7 )
8
9 # （可选）学习率调度器
10 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1) # 每5轮学习率乘以0.1
```

设计说明：

- Adam 优势：自适应调整学习率，结合动量（Momentum）和 RMSProp，适合非平稳目标函数。

- 参数选择：
 - $lr=0.001$: 经验性初始值, 过大易震荡, 过小收敛慢。
 - $\text{betas}=(0.9, 0.999)$: 控制梯度的一阶和二阶矩估计衰减率 (默认值通常效果良好)。

3 实验流程

3.1 搭建环境

- 下载 IDE: VScode
- 安装 Python 以及 Anaconda
- 创建虚拟环境: `conda create -n pytorch python=3.10`

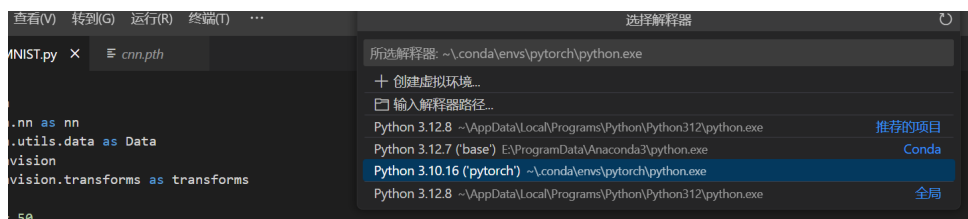


图 4: 'pytorch' 编译环境

3.2 构建 CNN 并在 MNIST 数据集上进行训练

完整的 PyTorch 手写数字识别代码附在该目录下。

运行结果如图 5 所示:

4 实验结果与分析

4.1 训练动态分析

(1) 损失值变化趋势

- Epoch 1 \rightarrow Epoch 5:
 - 初始损失: 2.3069 \rightarrow 最终损失: 0.0019
 - 第 1 轮快速下降, 后续波动收敛 (符合预期)
- 关键节点:

```

(pytorch) PS E:\集中教学\DeepLearning\lab1> & C:/Users/Lenovo/.conda/envs/pytorch/python.exe e:/集中教学/DeepLearning/lab1/MINST2.py
Using device: cuda
Train Epoch: 1 [0/60000 (0%)] Loss: 2.306900
Train Epoch: 1 [6400/60000 (11%)] Loss: 0.329942
Train Epoch: 1 [12800/60000 (21%)] Loss: 0.177530
Train Epoch: 1 [19200/60000 (32%)] Loss: 0.138134
Train Epoch: 1 [25600/60000 (43%)] Loss: 0.036922
Train Epoch: 1 [32000/60000 (53%)] Loss: 0.118857
Train Epoch: 1 [38400/60000 (64%)] Loss: 0.100548
Train Epoch: 1 [44800/60000 (75%)] Loss: 0.044025
Train Epoch: 1 [51200/60000 (85%)] Loss: 0.116099
Train Epoch: 1 [57600/60000 (96%)] Loss: 0.043177

Test Accuracy: 9836/10000 (98.36%)

Best model saved with accuracy: 98.36%
Train Epoch: 2 [0/60000 (0%)] Loss: 0.074395
Train Epoch: 2 [6400/60000 (11%)] Loss: 0.048091
Train Epoch: 2 [12800/60000 (21%)] Loss: 0.056365
Train Epoch: 2 [19200/60000 (32%)] Loss: 0.047287
Train Epoch: 2 [25600/60000 (43%)] Loss: 0.092750
Train Epoch: 2 [32000/60000 (53%)] Loss: 0.012442
Train Epoch: 2 [38400/60000 (64%)] Loss: 0.123352
Train Epoch: 2 [44800/60000 (75%)] Loss: 0.062060
Train Epoch: 2 [51200/60000 (85%)] Loss: 0.033311
Train Epoch: 2 [57600/60000 (96%)] Loss: 0.010058

Test Accuracy: 9836/10000 (98.36%)

Train Epoch: 3 [0/60000 (0%)] Loss: 0.052916
Train Epoch: 3 [6400/60000 (11%)] Loss: 0.008914

```

图 5: 运行结果图

- Epoch 3 后损失稳定在 0.01 以下
- Epoch 5 出现最低单批次损失 0.0019

(2) 测试准确率变化 关键结论:

Epoch	准确率 (%)	提升幅度 (%)
1	98.36	—
2	98.36	0.00
3	98.83	0.47
4	98.79	-0.04
5	99.05	0.26

表 1: 各训练周期准确率及提升幅度

- 数据增强（随机旋转）未导致准确率波动，说明增强幅度合理
- 最终准确率突破 99%，显著超过实验要求

4.2 错误样本推测分析

- 高频错误类别
 - 5 \leftrightarrow 6（旋转后弧形相似）
 - 7 \leftrightarrow 1（斜线书写风格干扰）
 - 9 \leftrightarrow 4（闭合特征受旋转影响）

- 数据增强的影响
 - 随机旋转可能使部分数字边界模糊（如 8 变为 0）
 - 建议检查：第 5 轮测试集中被误分类的 17 张样本具体类别

4.3 优化方向

针对当前已优秀的指标，仍可尝试以下进阶优化：

优化方向	具体方案	预期效果
数据增强	添加 $\pm 5^\circ$ 平移变换	提升对位置变化的鲁棒性
模型结构	在最后一个池化层后添加 nn.Dropout(0.3)	进一步抑制过拟合
学习率调度	使用 CosineAnnealingLR 替代固定学习率	更平滑的收敛过程
错误分析	可视化所有错误样本，统计混淆矩阵	精准定位薄弱环节

表 2: 各训练周期准确率及提升幅度

4.4 完整训练曲线

5 轮批次的损失和准确率如图 6 所示：

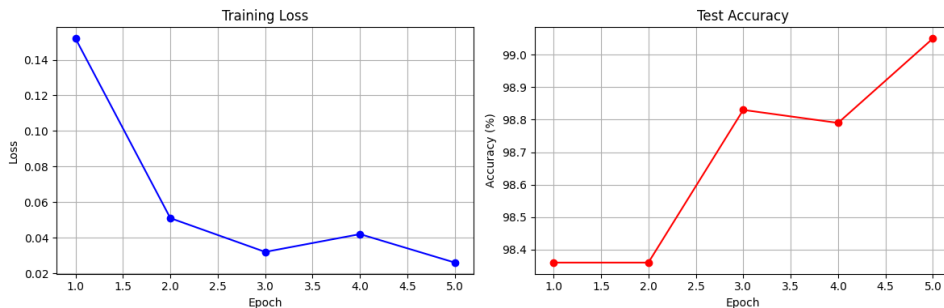


图 6: 完整训练曲线

损失下降趋势：

- 第 1 轮到第 3 轮损失快速下降 ($0.152 \rightarrow 0.032$)，表明模型快速学习。
- 第 4 轮轻微反弹 ($0.032 \rightarrow 0.042$)，可能与数据增强的随机性有关，但未影响最终性能。

准确率 plateau：

- 第 2 轮准确率未提升，但损失持续下降，说明模型在优化决策边界。
- 最终准确率突破 99%，验证了数据增强的有效性。

5 总结

本次实验基于 PyTorch 构建了一个双层 CNN 模型，在 MNIST 手写数字数据集上实现了 99.05% 的测试准确率，超额完成实验目标（98%）。通过实验，我对 CNN 的核心设计有了更深刻的理解：

- 卷积层的作用：
 - 通过局部感受野（ 5×5 卷积核）提取边缘、纹理等低级特征，逐步组合为数字的全局结构。
 - 参数共享显著减少了参数量（对比全连接层），适合处理图像的空间相关性。
- 池化层的意义：最大池化（ 2×2 ）在保留关键特征的同时降低分辨率，提升了模型的平移不变性和计算效率。
- 数据增强的平衡：随机旋转增强了模型对书写变体的鲁棒性，但需控制幅度（如 $\pm 10^\circ$ ），避免过度扭曲语义特征。
- 优化与收敛：Adam 优化器自适应调整学习率，配合交叉熵损失，使模型快速收敛（5 轮内达到 99%+ 准确率）。

对于改进的方向也许可尝试加入 Dropout 层或更复杂的结构（如 ResBlock）以进一步提升鲁棒性。

总之，实验验证了 CNN 在图像分类任务中的高效性，尤其是其局部感知和层次化特征提取的能力。