基于LeNet5的MNIST手写数字识别

# 概述

## 任务

本实验旨在使用卷积神经网络（CNN）实现对MNIST手写数字数据集的分类。具体任务是设计、实现并训练一个基于LeNet5模型的神经网络，最终能够对测试集中的手写数字进行准确分类。

## 数据集

MNIST数据集包含60,000张训练图像和10,000张测试图像，每张图像是28x28像素的灰度图，标签为0到9的数字。

## 解决方案

选择使用经典的LeNet5卷积神经网络结构进行手写数字识别。解决方案包括以下步骤：

1. 数据读取与预处理
2. LeNet5网络结构设计
3. 损失函数和优化器设计
4. 模型训练与测试
5. 结果分析

# 解决方案

## 网络结构设计

设计的LeNet5模型包括两个卷积层、两个最大池化层、三个全连接层，并在卷积层和全连接层之间加入批量归一化和Dropout层。具体结构如下：

# 定义LeNet5模型结构  
class LeNet5(nn.Module):  
 def \_\_init\_\_(*self*):  
 super(LeNet5, *self*).\_\_init\_\_()  
 # 第一个卷积层，输入通道1（灰度图），输出通道6，卷积核大小5x5，边缘填充2  
 *self*.conv1 = nn.Conv2d(1, 6, 5, padding=2)  
 *self*.bn1 = nn.BatchNorm2d(6) # 批量归一化层，对第一个卷积层的输出进行归一化，以下类似  
 *self*.conv2 = nn.Conv2d(6, 16, 5)  
 *self*.bn2 = nn.BatchNorm2d(16)  
 # 第一个全连接层，输入特征数量16\*5\*5（16个特征图，每个特征图大小为5x5），输出特征数量120  
 *self*.fc1 = nn.Linear(16 \* 5 \* 5, 120)  
 *self*.bn3 = nn.BatchNorm1d(120)  
 *self*.fc2 = nn.Linear(120, 84)  
 *self*.bn4 = nn.BatchNorm1d(84)  
 *self*.fc3 = nn.Linear(84, 10)  
  
 # 定义模型的前向传播路径  
 def forward(*self*, x):  
 # 通过第一个卷积层后使用LeakyReLU激活函数，并应用最大池化  
 x = F.max\_pool2d(F.leaky\_relu(*self*.bn1(*self*.conv1(x)), negative\_slope=0.01), (2, 2))  
 x = F.dropout(x, p=0.3, training=*self*.training) # 应用dropout防止过拟合  
 x = F.max\_pool2d(F.leaky\_relu(*self*.bn2(*self*.conv2(x)), negative\_slope=0.01), (2, 2))  
 x = F.dropout(x, p=0.3, training=*self*.training)  
 x = x.view(-1, *self*.num\_flat\_features(x)) # 将特征图展平为一维向量  
 # 通过第一个全连接层后使用LeakyReLU激活函数  
 x = F.leaky\_relu(*self*.bn3(*self*.fc1(x)), negative\_slope=0.01)  
 x = F.dropout(x, p=0.3, training=*self*.training)  
 x = F.leaky\_relu(*self*.bn4(*self*.fc2(x)), negative\_slope=0.01)  
 x = F.dropout(x, p=0.3, training=*self*.training)  
 x = *self*.fc3(x) # 通过输出层  
 return x  
  
 # 计算展平后的特征数量  
 def num\_flat\_features(*self*, x):  
 size = x.size()[1:] # 除批量维度外的所有维度  
 num\_features = 1  
 for s in size:  
 num\_features \*= s # 计算展平后的特征总数量  
 return num\_features

## 损失函数设计

选择交叉熵损失函数（CrossEntropyLoss）作为损失函数，用于计算模型输出与真实标签之间的差异。交叉熵损失函数广泛应用于分类任务，特别是多分类问题。它衡量的是模型预测的概率分布与实际标签分布之间的差异。MNIST手写数字识别是一个典型的多分类问题（0-9，共10类），交叉熵损失函数在这种情况下非常合适。

criterion = nn.CrossEntropyLoss(size\_average=False) # 定义损失函数。

## 优化器设计

使用Adam优化器，学习率为0.001，并加入权重衰减（weight\_decay）来防止过拟合。

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.99), weight\_decay=1e-4)

## 模型训练与测试

训练过程中，在每个epoch结束后进行测试，并使用学习率调度器来逐步降低学习率。

# 加入学习率调度器  
scheduler = torch.optim.lr\_scheduler.StepLR(optimizer, step\_size=20, gamma=0.1)  
  
# 训练和测试循环  
ENDEPOCH = 99  
for epoch in range(0, ENDEPOCH+1):  
 print('start train:')  
 train(epoch)  
 if epoch == ENDEPOCH:  
 torch.save(model.state\_dict(), './model\_params.pkl') # 保存模型参数到.pkl文件  
 print('start test:')  
 test()  
 scheduler.step() # 在每个epoch结束后更新学习率

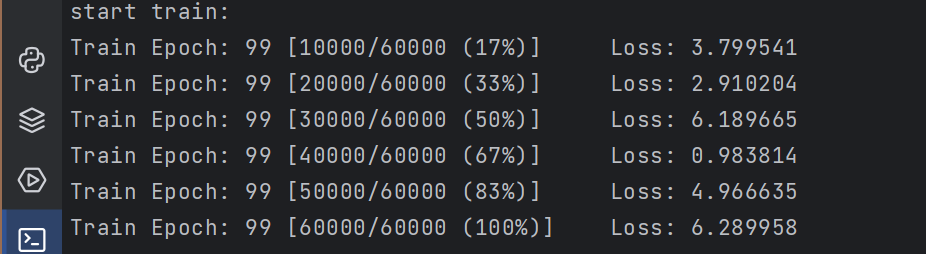
# 实验分析

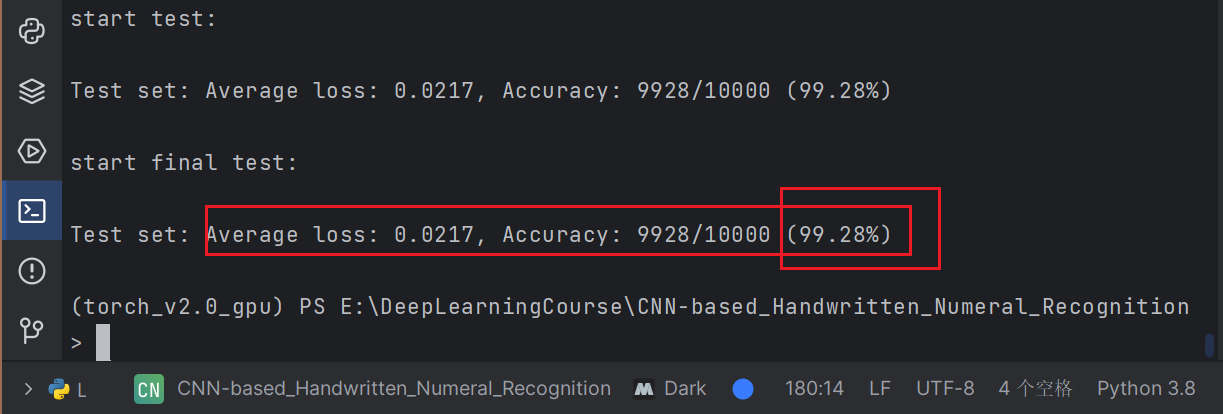
## 数据集介绍

MNIST数据集是一个大型的手写数字数据库，广泛用于训练和测试各种图像处理系统。它包含60,000个训练样本和10,000个测试样本。每张图像为28x28像素的灰度图，标签为对应的数字（0-9）。

## 实验结果与分析

在训练过程中记录了每个epoch的训练损失和测试准确率。以下是训练过程中的最终训练和测试结果：从实验结果可以看出， 我们的模型在测试集上的平均损失为0.0217，准确率达到了99.28%。





# 创新点

在经典的LeNet5结构基础上，引入以下改进：

* **批量归一化（Batch Normalization）**：在每个卷积层后添加了批量归一化层，这不仅加快了训练速度，还提高了模型的稳定性和准确性。批量归一化可以使得每一层的输入保持在一个合理的范围内，避免梯度消失或爆炸的问题。
* **Leaky ReLU激活函数**：Leaky ReLU能够有效避免ReLU可能引起的“神经元死亡”问题，使得训练过程更加平稳。
* **Dropout层**：在全连接层中加入了Dropout层，有效减少了过拟合现象，提高了模型的泛化能力。Dropout通过在训练过程中随机丢弃一部分神经元，迫使模型学到更加鲁棒的特征。
* **学习率调度器：**在训练过程中使用学习率调度器（StepLR），逐步降低学习率，以适应模型在不同训练阶段的需求，进一步提升了模型的性能。

# 总结

本次实验成功实现了基于LeNet5卷积神经网络的MNIST手写数字识别任务，并取得了优异的结果。以下是对本实验的详细总结：

1. 模型设计：

* 采用经典的LeNet5模型结构，包含两个卷积层、两个最大池化层和三个全连接层。
* 在每个卷积层后添加了批量归一化（Batch Normalization）层，以提高模型的训练速度和稳定性。
* 使用Leaky ReLU作为激活函数，避免了传统ReLU可能引起的“神经元死亡”问题。
* 在全连接层中加入了Dropout层，减少过拟合，提高模型的泛化能力。

1. 数据处理：

* 从MNIST数据集中读取训练和测试图像，并将其转换为适合卷积神经网络输入的格式。
* 使用DataLoader加载数据集，以支持批量处理、数据打乱和多线程加速。

1. 训练与优化：

* 选择交叉熵损失函数（CrossEntropyLoss），适合多分类问题并能有效指导模型的训练。
* 使用Adam优化器进行模型参数的优化，设置合理的学习率和权重衰减以防止过拟合。
* 加入学习率调度器（StepLR），在训练过程中动态调整学习率以进一步提高训练效果。

1. 实验结果：

经过100个epoch的训练，模型在测试集上达到了99.28%的准确率，平均损失为0.0217，表明模型具有良好的分类能力和稳定性。