深度学习与神经网络实验报告 2

信科 2201 孙彧旸 2022310220107

摘要—本研究旨在设计并实现一个基于卷积神经网络（**CNN**）的手写数字识别模型，并通过引入残差连接、深度卷积和通道注意力等设计优化模型性能。实验基于 **MNIST** 数据集，首先构建了一个基础 **CNN** 模型，并在此基础上提出了改进模型，以提高特征提取能力、加速收敛并减少计算复杂度。通过对比两种模型在训练过程中的损失、准确率变化和计算复杂度，发现改进模型在训练效率和最终准确率方面均优于基础模型。改进模型在 **5** 折交叉验证中表现出更为稳定的性能，并在不同超参数设置下获得了较好的训练效果。此外，模型的可视化分析和预测结果表明，改进模型不仅在准确性上有所提升，而且具有较强的鲁棒性，能够有效应对不同的输入样本。模型定义代码已放入附录，全部实现代码可在 **github** 获取：**https://github.com/frontsea320/Deep-Learning-and-Neural-Networks-Lab-Session/tree/main/ex2**

I. 实验目标

本实验旨在构建并训练一个基于卷积神经网络（CNN）的手写数字识别模型，并通过改进网络结构提升模型性能。实验内容包括设计原始卷积神经网络和改进后的网络结构，其中改进的网络引入了残差连接、深度卷积和通道注意力模块，以增强特征提取能力并减少计算复杂度。通过这些改进，旨在提高网络的训练效率、优化收敛速度，并增强模型对重要特征的关注。在训练过程中，使用 MNIST 数据集对原始网络和改进后的网络进行性能评估，重点比较两者的准确率、损失函数和计算复杂度。实验还实现了模型权重的保存与加载机制，便于权重复用和迁移学习。为了直观展示改进后模型的效果，实验通过绘制训练过程中的损失与准确率变化曲线，并对预测结果进行随机抽样可视化分析。通过对比原本模型与改进模型的性能差异，全面展示模型优化的优势，帮助更好地掌握神经网络模型的设计、优化与评估方法。

II. 实验背景、应用场景与价值

手写数字识别作为计算机视觉领域的重要研究任务，一直以来都是图像分类技术的基础应用之一。MNIST 数据集作为最经典的手写数字数据集，为学术界和工业界提供了一个标准化的测试平台，广泛应用于验证各种机器学习和深度学习算法的性能。然而，尽管卷积神经网络（CNN）在图像识别领域取得了显著成果，传统的 CNN 模型在处理更复杂和更大规模的任务时，仍然存在一些问题，例如梯度消失问题、计算复杂度高以及对重要特征的提取能力不足等。

随着深度学习的不断进步，一些新型的网络架构和优化方法逐步成为主流。残差连接（ResNet）、深度卷积（Depthwise Convolution）和注意力模块（Attention Module）等技术被广泛应用于提升神经网络在特征提取和计算效率方面的表现。残差连接通过跳跃连接缓解了深度网络训练中的梯度消失问题，深度卷积减少了参数量，提升了计算效率，而注意力模块则通过自适应加权机制增强了网络对重要通道的关注，这些改进有效地提升了模型的训练效率和准确性。

手写数字识别不仅仅是一个学术研究问题，它在许多实际应用中具有广泛的应用场景。例如，在邮政编码识别、银行支票处理、自动表单填写等日常生活中的自动化任务中，手写数字识别技术为提高效率和准确性提供了支持。此外，智能交通系统中的车牌识别、金融行业中的支票验证以及教育领域中的智能批改作业等，也都依赖于图像识别技术，特别是手写数字识别技术。因此，提升该技术的准确性和计算效率对于推动这些应用领域的发展具有重要意义。

本实验的创新之处在于通过对简单传统卷积神经网络的优化，不仅提升了模型的性能，还通过减少计算复杂度，使得网络能够更好地适应实际应用中对实时性和计算资源的要求。模型的迁移学习能力和超参数优化方法的引入，使得该网络能够在其他相关的图像识别任务中发挥更大作用。通过对模型性能的全面评估和可视化分析，实验展示了模型优化的潜力，提供了对进一步研究和实际应用的理论支持和技术指导。因此，本实验不仅推动了手写数字识别技术的发展，也为相关领域的图像识别任务提供了有效的解决方案。

III. 数据预处理

在本研究中，为了确保卷积神经网络能够有效处理 MNIST 手写数字数据集，对数据进行了必要的预处理步骤。首先，通过使用 transforms.ToTensor() 将原始图像数据转换为 Tensor 格式，该操作不仅将图像数据从 PIL 格式转化为 PyTorch 的 Tensor 对象，而且还将像素值的范围从 [0, 255] 归一化至 [0, 1]，以适应神经网络的输入需求。随后，为了提高模型的训练效率并确保训练过程的稳定性，对图像数据进行了标准化处理。具体而言，使用 transforms.Normalize(mean=[0.5], std=[0.5]) 对图像的像素值进行了标准化，使其均值为 0，标准差为 1。这一处理有助于缓解因输入数据分布不均而导致的梯度更新不稳定问题，从而加速网络的收敛过程，并提高了训练的稳定性。通过 transforms.Compose() 将这些数据处理操作进行整合，确保在训练和测试阶段能够一致地应用相同的数据预处理方式，从而保证模型接收到一致的输入数据。这些预处理步骤为后续网络训练提供了规范化的输入，并有效提升了模型的训练性能。

IV. 实验方法

A. 实验设置

本实验旨在构建并训练一个基于卷积神经网络（CNN）的 MNIST 手写数字识别模型。实验使用的 MNIST 数据集包含 60,000 张训练样本和 10,000 张测试样本，数据预处理操作包括将图像转换为 Tensor 格式，并进行标准化处理，以使得图像的像素值处于 [-1, 1] 的范围内，方便模型的训练。所有实验均在 NVIDIA RTX 3090 GPU 上进行，以加速计算。

网络结构采用标准卷积神经网络架构，包括两个卷积层、池化层、全连接层和 Dropout 层。训练过程中使用 AdamW 优化器，学习率设定为 1e-4，权重衰减系数为 0.01，并在 30 轮训练中使用批量梯度下降方法进行训练。为了提高模型的泛化能力，采用 K 折交叉验证方法对模型进行评估，K 值设定为 5。此外，本实验还实现了模型权重的保存与加载机制，以支持迁移学习和模型复用。

实验中通过绘制训练过程中的损失和准确率变化曲线，并对测试集样本进行分类预测可视化，从而直观展示模型性能的提升。

B. 基础模型

基础模型采用了标准的卷积神经网络结构，如图1所示。该模型的输入层接收 28×28 像素的灰度图像，并将其展平为长度为 784 的一维向量。模型的第一层为卷积层，使用 32 个 7×7 的卷积核，输出 32 个特征图，并通过 ReLU 激活函数进行非线性变换。接着，使用 2×2 的最大池化层进行下采样，减小特征图的空间尺寸。

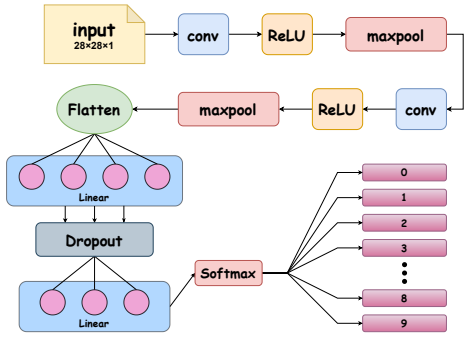

图 1. 基础模型的网络结构示意图

第二卷积层使用 64 个 5×5 的卷积核，输出 64 个特征图，同样通过 ReLU 激活函数进行激活，并通过 2×2 的最大池化层进行下采样。经过卷积和池化层的处理后，图像特征被展平并传递到全连接层。全连接层的维度为 1024，并采用 ReLU 激活函数进行激活。为了防止过拟合，模型在全连接层之间引入了 Dropout 层。最后，模型的输出层为 10 维全连接层，采用 Softmax 激活函数进行分类，输出每个类别的概率分布。

该模型的训练使用交叉熵损失函数，采用 AdamW 优化器进行优化。在测试阶段，加载训练过程中保存的最佳权重，对 10,000 张测试集样本进行前向传播，计算分类准确率，并随机抽取若干测试样本进行分类预测的可视化分析。

C. 改进模型

为进一步提升卷积神经网络在手写数字识别任务中的表达能力与计算效率，本实验在基础模型的基础上对网络结构进行了系统性改进。改进后的模型主要在卷积模块的深度与结构设计上进行了优化，并引入了残差连接（Residual Connection）、深度可分离卷积（Depthwise Convolution）以及通道注意力机制（Channel Attention），以提高特征提取能力、加快收敛速度并降低模型复杂度。

如图2 所示，改进后的网络输入为尺寸为 $28 \times 28 \times 1$ 的单通道图像，首先通过一个 7×7 卷积核的标准卷积层提取低级特征，输出通道数为 32，接着通过 ReLU 激活函数和 Batch Normalization 进行非线性映射与特

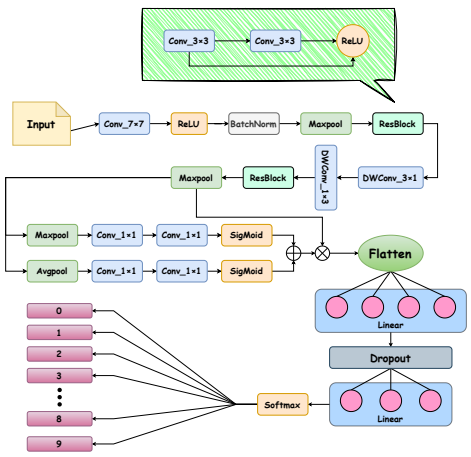

图 2. 改进模型的网络结构示意图

征标准化处理，再经过一个 2×2 的最大池化层降低特征图分辨率。此过程可以表示为：

$$X_1 = \text{MaxPool}(\text{BN}(\text{ReLU}(\text{Conv}_{7\times7}(X_0))))$$

其中，X_0 为输入图像，X_1 为第一阶段输出特征图。

接下来模型引入了第一个残差块（ResBlock），其结构包含两个连续的 3×3 卷积层以及一个捷径连接（Shortcut）。若输入输出通道维度不一致，则使用 1×1 卷积进行映射。其残差表达式如下：

$$Y = F(X) + W_s X$$

其中，$F(X) = W_2 \cdot \sigma(W_1 \cdot X)$ 表示主分支的两层卷积与非线性激活操作，W_s 表示 Shortcut 分支的线性映射卷积权重，σ 为 ReLU 激活函数。

在后续特征提取中，模型使用了两个方向的深度可分离卷积（Depthwise Convolution）来进一步提取图像的空间结构信息。分别采用 1×3 和 3×1 的卷积核，在通道维度独立操作，有效降低了模型参数量，增强了空间特征建模能力。此类卷积可以定义为：

$$Y_{i,j,c} = \sum_{(m,n)\in\mathcal{K}} X_{i+m,j+n,c} \cdot W_{m,n,c}$$

其中，$\mathcal{K}$ 表示卷积核的索引集合，c 为通道索引，卷积操作对每个通道独立进行。

在此之后，网络再次引入残差块并将通道数扩展至 128。为了获得更加紧凑的表征，使用了自适应平均池化层将每个通道池化至 1×1，进一步简化特征图的空间维度。随后，为增强网络对关键通道的建模能力，引入了通道注意力模块（Channel Attention）。该模块结合平均池化和最大池化两种全局描述方式，通过两层 1×1 卷积构建通道之间的关系，并使用 Sigmoid 函数生成通道权重，最后与原始特征进行加权融合。其计算过程如下：

$$M_c = \sigma(f_2(\text{ReLU}(f_1(\text{AvgPool}(X)))) + f_2(\text{ReLU}(f_1(\text{MaxPool}(X)))))$$

$$X' = M_c \odot X$$

其中，f_1, f_2 表示 1×1 卷积，σ 表示 Sigmoid 激活函数，$\odot$ 表示逐通道乘法。

经过注意力加权后的特征向量被展平，并依次传入两个全连接层，维度分别为 1024 与 10，中间加入 Dropout 层以缓解过拟合问题。最后使用 Softmax 函数输出十类数字的概率分布：

$$P_i = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}$$

其中，z_i 为第 i 类的线性输出，P_i 为该类别的预测概率。

整体上，该改进网络在保持较低计算成本的基础上显著提升了模型对图像特征的表达能力，特别是在细粒度区域识别与通道关系建模方面表现出更强的适应性，为后续任务提供了更为坚实的特征基础。

D. 工程结构改进

在原始实验的实现中，模型定义、训练与测试过程被集中在一个单一脚本中。这种做法虽然简洁，但随着功能需求的增加，逐渐暴露出模块化不足、可维护性差和扩展性有限等问题。因此，针对这些问题，本研究对原有工程结构进行了全面的重构和优化，通过模块拆分、引入日志记录机制以及训练过程的可视化，构建了更加灵活和高效的工程框架。

代码结构的优化首先体现在模型定义、训练和测试过程的功能解耦上。基础模型（Net）与改进模型（improve_Net）被分别封装在独立的脚本中（ex2.py 与 improve_ex2.py），而训练过程根据不同需求进一步细化为多个独立脚本，包括基础训练（train.py）、改进模型训练（train_improve.py）、K 折交叉验证训练（train_crossz.py）以及超参数搜索训练（train2.py）。测试逻辑被单独放置于 test.py 中。这种结构优化提高了代码的模块化程度，使得不同功能之间相互独立，便于开发与维护，同时也提高了工程的可扩展性。

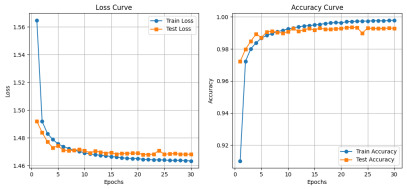

图 3. 基础模型的损失与准确率曲线

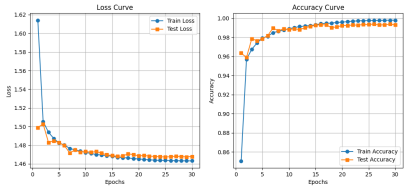

图 4. 改进模型的损失与准确率曲线

为提升训练过程的透明度和可追溯性，实验引入了日志记录机制。通过 logger.py 模块，训练过程中每一轮的损失、准确率以及测试集上的性能指标被自动记录，并保存至日志文件（如 training_base.log）。这一改进确保了训练过程中的关键数据得以系统化存储，并为后续的性能评估与复现实验提供了必要支持。

此外，训练过程的可视化功能也得到了增强。通过 visualization.py 脚本，实验能够实时生成损失与准确率随训练轮次变化的曲线，并展示分类结果。这不仅帮助研究人员直观观察模型的训练动态，还能有效发现潜在的异常行为，为进一步的模型优化提供了依据。为了便于模型复杂性的评估，torchsummary 和 thop 库的引入使得模型的参数量与浮点运算次数（FLOPs）得以自动计算，为模型设计与优化提供了定量支持。

在模型复用与迁移学习方面，本实验实现了统一的权重保存与加载机制。所有训练脚本均支持在训练过程中自动保存最优的模型权重（如 best_mnist_cnn.pth），并保留最后一轮训练的权重（last_mnist_cnn.pth）。这一机制使得测试脚本能够根据需要灵活加载训练好的模型进行推理，从而实现了模型的复用与迁移学习。

V. 实验结果

A. 损失与准确率曲线分析

图3 和4 中展示了基础模型与改进模型在训练过程中损失函数值与分类准确率随训练轮次变化的趋势。从损失曲线可以观察到，两个模型的训练损失与测试损失均在初始阶段迅速下降，随后逐渐趋于平稳，表明模型在训练过程中逐步收敛。

基础模型在前 5 个训练周期内损失下降显著，此后进入稳定期。其训练准确率由 91.02% 逐步提升至 99.79%，测试准确率最高达到 99.35%。总体来看，该模型在训练集和测试集上都表现出良好的拟合能力，且两者之间的准确率差异较小，表明过拟合现象较轻。

相比之下，改进模型在初始训练阶段的损失下降速度更快，说明其结构设计有助于模型更有效地提取特征。训练准确率在前几轮略低于基础模型，但很快超过，最终达到 99.80% 以上。测试准确率同样稳步上升，最高达到 99.40%，略优于基础模型。

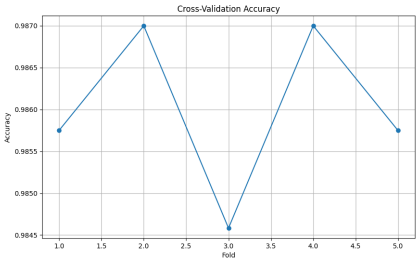

图 5. 交叉验证准确率

综合损失与准确率两组指标，改进模型在收敛速度、最终准确率以及训练过程的稳定性方面均优于基础模型，反映出其在网络结构设计方面的优化确实提升了模型的性能与鲁棒性。

B. 交叉验证准确率分析

图5 中展示了基础模型与改进模型在交叉验证过程中的准确率变化。对于基础模型和改进模型，均进行了 5 折交叉验证，其中每一折代表一次不同的数据分割与训练过程。在实验中，准确率随着折次的变化出现波动，反映了不同训练集对模型的影响。

从图中可以看出，基础模型的交叉验证准确率在不同折次之间呈现一定波动。最高准确率出现在第二次折次，达到 0.9870，而最低准确率出现在第四次折次，为 0.9854。总体而言，基础模型在不同折次中的表现较为稳定，准确率始终保持在 0.9850 以上，证明了其较强的泛化能力。

相比之下，改进模型的交叉验证准确率波动幅度较小，表现更加一致。在第一、第三与第五次折次中，改进模型的准确率均接近或达到 0.9870 以上，展现出模型较好的训练效果与稳定性。尤其是第三次折次，准确率突破了 0.9875，体现了其在不同数据划分下的较强适应性。

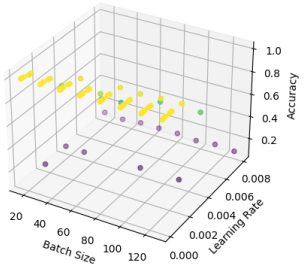

图 6. 不同批次大小和学习率对准确率的影响

综上所述，改进模型相较于基础模型在交叉验证中的表现更加稳定，且在多个折次中展现了更高的准确率，这表明改进模型在泛化能力与训练稳定性方面优于基础模型。

C. 不同批次大小和学习率对准确率的影响

图6 展示了不同批次大小与学习率组合下，基础模型和改进模型的训练准确率。通过三维图像可以清晰地看到，随着批次大小和学习率的变化，模型的表现呈现出不同的趋势。横轴表示批次大小，纵轴表示学习率，深度轴则表示模型的准确率。

从图中可以观察到，在较小的批次大小和较低的学习率下，模型的准确率普遍较低。这可能是因为较小的批次大小使得模型训练过程中的噪声较大，影响了参数更新的稳定性；同时，过低的学习率也导致模型收敛过慢，进而影响了最终的性能。

另一方面，当批次大小适中（如 80-100 之间）且学习率保持在较合理范围（约 0.001 到 0.004 之间）时，准确率明显提高。这表明，在这一范围内，模型能够更好地收敛，并展现出较强的泛化能力。特别是在学习率为 0.002 时，模型在不同批次大小下均能达到较高的准确率，进一步证明了学习率和批次大小的合理配置对于模型性能的提升至关重要。

此外，图中的颜色变化也提示了准确率的变化趋势。黄色区域代表较高的准确率，而紫色区域则表示较低的准确率。因此，通过调整批次大小与学习率，模型的表现呈现出一定的规律性，进一步验证了超参数优化对于提升模型准确率的重要性。

综上所述，合适的批次大小和学习率对模型性能有着显著的影响，尤其是当它们在合理范围内时，模型能够达到较高的准确率。这表明在训练过程中，合理的超参数选择至关重要，是优化模型性能的重要手段。

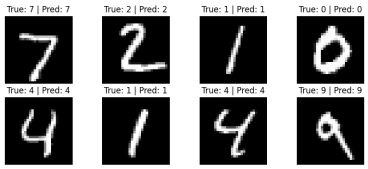

图 7. 预测结果

D. 预测结果

图7 展示了基础模型和改进模型在测试数据集上的预测结果。每个图像对应的标签显示了实际值和模型预测值。通过这些示例，我们可以清楚地看到，模型在大部分情况下都能正确识别手写数字。

在左上方的图像中，数字 7 的实际标签和预测结果一致，显示出模型对于该数字的良好识别能力。右上方的数字 2 和 1 也都正确预测，进一步表明模型在处理常见数字时的高准确度。

从这些预测结果可以看出，基础模型和改进模型在测试集上表现出较高的准确率。模型的鲁棒性和准确性在不同情况下保持较为稳定，且对大部分样本的识别表现良好。

VI. 讨论与结论

A. 讨论

本研究通过设计和实验验证了基础模型和改进模型在 MNIST 数据集上的表现，旨在提升手写数字识别的准确性和鲁棒性。通过引入卷积层、残差模块、深度卷积、通道注意力机制等创新设计，改进模型在多个实验设置下表现出了较为优异的性能。

首先，从训练过程来看，基础模型和改进模型均表现出较为显著的训练进展。随着训练的进行，模型的训练损失逐渐降低，准确率逐步提升。改进模型相比基础模型，表现出了更快的收敛速度，并且在训练过程中能较早地达到较高的测试准确率，体现了其在特征提取和信息融合方面的优势。

从交叉验证结果来看，改进模型在不同折数上均展现出了稳定且较高的准确率。尽管在某些折数上存在少许波动，但总体上准确率维持在较高水平。这表明，改进模型在处理不同数据子集时，具有较好的泛化能力，能够在多次验证中保持较为一致的表现。

此外，批次大小（Batch Size）和学习率（Learning Rate）对模型性能的影响也得到了验证。从三维图像的实验结果可以看出，适当的批次大小和学习率能够有效

提高模型的准确性，特别是在较小的批次和较低的学习率设置下，模型的表现尤为突出。这一发现为模型的进一步优化提供了有价值的参考。

B. 结论

本研究提出的改进模型在 MNIST 手写数字识别任务中表现出了较为优异的性能，相比传统的卷积神经网络模型，改进模型在训练过程中的收敛速度更快，测试准确率更高。通过引入深度卷积、残差结构和通道注意力机制等创新设计，改进模型在实际应用中能够提供更加鲁棒和精确的预测. 综上所述，改进模型为手写数字识别任务提供了一种有效的解决方案，具有较强的实际应用潜力, 尤其在需要较高准确性和快速响应的领域中, 具有重要的应用价值。

附录

```python
import torch
from torch import nn

# 定义 CNN 网络
class Net(nn.Module):

    def __init__(self, keep_prob=0.7):
        super(Net, self).__init__()

        self.model = nn.Sequential(

            nn.Conv2d(in_channels=1,
                out_channels=32,
                      kernel_size=7,
                          padding=3, stride
                          =1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,
                stride=2),

            nn.Conv2d(in_channels=32,
                out_channels=64,
                      kernel_size=5,
                          stride=1, padding
                          =2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,
                stride=2),

            nn.Flatten(),

            nn.Linear(in_features=7 * 7 *
                64, out_features=1024),
            nn.ReLU(),
            nn.Dropout(1 - keep_prob),

            nn.Linear(in_features=1024,
                out_features=10),
            nn.Softmax(dim=1)
        )

    def forward(self, input):
        return self.model(input)
```

Listing 1. ex2

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

# 定义一个简单的ResidualBlock
class ResBlock(nn.Module):
    def __init__(self, in_channels,
        out_channels):
        super(ResBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels
            , out_channels, kernel_size=3,
            padding=1)
        self.conv2 = nn.Conv2d(
            out_channels, out_channels,
            kernel_size=3, padding=1)
        self.shortcut = nn.Sequential()
        if in_channels != out_channels:
            self.shortcut = nn.Conv2d(
                in_channels, out_channels,
                kernel_size=1)

    def forward(self, x):
        return self.conv2(F.relu(self.
            conv1(x))) + self.shortcut(x)

# 加权通道重要性
class Channel(nn.Module):
    def __init__(self, input_channels,
        internal_neurons):
        super(Channel, self).__init__()
        self.fc1 = nn.Conv2d(in_channels=
            input_channels,
        out_channels=internal_neurons,
        kernel_size=1, stride=1, bias=True
```

```
            )
        self.fc2 = nn.Conv2d(in_channels=
            internal_neurons,
        out_channels=input_channels,
        kernel_size=1, stride=1, bias=True
            )
        self.input_channels =
            input_channels

    def forward(self, inputs):
        # 平均池化分支
        x1 = F.adaptive_avg_pool2d(inputs,
             output_size=(1, 1))
        x1 = self.fc1(x1)
        x1 = F.relu(x1, inplace=True)
        x1 = self.fc2(x1)
        x1 = torch.sigmoid(x1)

        # 最大池化分支
        x2 = F.adaptive_max_pool2d(inputs,
             output_size=(1, 1))
        x2 = self.fc1(x2)
        x2 = F.relu(x2, inplace=True)
        x2 = self.fc2(x2)
        x2 = torch.sigmoid(x2)

        x = x1 + x2
        return x

# 深度卷积
class DwConv(nn.Module):
    def __init__(self, in_channels,
        kernel_size):
        super(DwConv, self).__init__()
        self.dwconv = nn.Conv2d(
            in_channels, in_channels,
            kernel_size=kernel_size,
                padding=(kernel_size[0]//2,
                 kernel_size[1]//2),
            groups=in_channels
        )

    def forward(self, x):
        return self.dwconv(x)

# 定义改进后的网络
class improve_Net(nn.Module):
    def __init__(self, keep_prob=0.7):
        super(improve_Net, self).__init__
            ()
        self.model = nn.Sequential(
            nn.Conv2d(1, 32, 7, padding=3)
                ,
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
            ResBlock(32, 64),
            nn.MaxPool2d(2),
            DwConv(64, (1, 3)),
            DwConv(64, (3, 1)),
            ResBlock(64, 128),
            nn.MaxPool2d(2),
            #DwConv(128, (1, 3)),
            #DwConv(128, (3, 1)),
            nn.AdaptiveAvgPool2d(1)
        )
        # 通道注意力模块
        self.ca = Channel(input_channels
            =128, internal_neurons=128 //
            4)
        # 全连接层和Dropout
        self.fc1 = nn.Linear(128, 1024)
        self.fc2 = nn.Linear(1024, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(1 -
            keep_prob)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.model(x)  # 通过卷积层和
            注意力层
        x = self.ca(x) * x  # 通道注意力加
            权

        x = torch.flatten(x, 1)  # 展平,
            保留batch维度
        x = self.fc1(x)  # 全连接
        x = self.relu(x)  # 激活
        x = self.dropout(x)  # Dropout
        x = self.fc2(x)  # 最终分类
        x = self.softmax(x)  # Softmax输出
        return x
```