华中科技大学

《计算机视觉导论》
上机实验（三）报告

试用水印

题目： 基于剪枝算法的深度神经网络压缩

院　　系	计算机科学与技术学院
专业班级	计科 2301 班
姓　　名	
学　　号	
指导教师	李贤芝

计算机科学与技术学院

目　　录

试用水印

1 问题定义与理解

1.1 问题描述

本次实验是在实验二“基于卷积神经网络的两个数字比较”的基础上进行的进一步研究，目标是针对已训练完成的孪生卷积神经网络（SiameseCompareNet）进行模型压缩与结构优化。

实验二的网络虽然在 MNIST 数据集上取得了 98.95% 的高准确率，但其卷积层中仍存在一定数量的冗余通道。这些通道在特征提取过程中激活程度较低，对最终分类贡献有限。

因此，本实验的核心任务是：在不显著降低分类准确率的前提下，通过权重剪枝减少模型参数量，实现模型压缩。

模型压缩是深度学习应用中关键的优化手段之一，尤其在资源受限的环境（如移动端或嵌入式设备）中，具有重要的实际价值。

1.2 相关理论基础

（1）卷积神经网络的参数冗余性

卷积神经网络（CNN）在图像识别、目标检测和视觉匹配等任务中取得了显著成果。然而，大量研究表明，深层卷积网络中的许多参数在推理时并未被充分激活。

例如，某些卷积通道在整个测试集上的平均响应几乎为零，这些“休眠神经元”可以安全地剪除。

（2）权重剪枝概念

剪枝技术通过删除或置零部分权重以降低模型复杂度。按粒度可分为：

非结构化剪枝（Unstructured Pruning）：逐元素剪除权重；

结构化剪枝（Structured Pruning）：按通道、卷积核或层级进行整体裁剪。

本实验采用结构化通道剪枝，直接将最后一层卷积层中激活值较低的通道置零。

试用水印

这种方式可在保持网络架构不变的同时减少有效参数量，方便后续加速推理。

1.3 实验环境

为保证实验的可重复性与高效性，本实验在如下软硬件环境中进行：

显卡：NVIDIA GeForce RTX 4070 Laptop GPU

显存：8 GB GDDR6

内存：32 GB DDR5

CUDA 版本：12.3

操作系统：Windows 11 家庭版

Python 版本：3.11.11

深度学习框架：PyTorch 2.6.0

开发工具：Visual Studio Code (VSCode)

环境管理：Anaconda（虚拟环境隔离与依赖管理）

运行设备：本地 GPU 加速训练

试用水印

2 数据分析及处理

2.1 数据集分析

MNIST 数据集是深度学习领域的经典基准数据集，包含 70000 张手写数字图像，其中训练集 60000 张，测试集 10000 张。每张图像都是 28×28 像素的灰度图，像素值范围为 0-255，标签为 0-9 共 10 个类别。数据集的特点是图像尺寸统一、背景干净、数字居中，但不同人的书写风格差异较大，存在倾斜、粗细、连笔等变化。从表 1-1 MNIST 类别数量分布（部分）可以看出，MNIST 数据集的类别分布基本均衡，每个数字约占 10%，这保证了模型训练时不会受到类别不平衡的影响。接下来我们可视化不同类别的样本，观察数据的多样性和特征差异。

表 1-1 MNIST 类别数量分布（部分）

数字	训练集数量/张	测试集数量/张
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010

试用水印

2.2 数据处理

首先进行数据标准化，标准化公式为

$$x_{norm} = \frac{x - \mu}{\sigma} = \frac{x - 0.1307}{0.3081}$$

这样处理后，输入数据的分布接近标准正态分布，有助于梯度下降的稳定性和收敛速度。

为了提升模型对书写风格变化的鲁棒性，我们需要进行数据增强，训练时对图像进行随机变换

这些变换包括：仿射变换（旋转、平移、缩放、剪切）以 70%的概率应用，高斯模糊以 20%的概率应用。这样的组合既保持了数字的可识别性，又增加了样本的多样性，模拟了真实场景中手写数字的自然变化。

2.3 数据加载

为了模拟实际应用中标注数据稀缺的场景，实验只使用 10%的训练集（6000 张）和 10%的测试集（1000 张）。为了确保实验的可重复性，代码使用缓存机制固定数据划分。**load_or_create_indices** 函数首先检查缓存文件是否存在，如果存在则直接加载，确保每次运行都使用相同的数据划分。如果缓存不存在，则使用固定的随机种子生成新的随机索引并保存。这种设计保证了实验的完全可重复性。

Siamese 网络的训练需要成对的数据。main.py 中定义了 **MNISTPairsDataset** 类，动态生成正负样本对。首先构建一个标签到样本索引的映射字典，然后在__getitem__时动态生成样本对。每次调用时，以 50%的概率生成同类对（标签为 1），以 50%的概率生成异类对（标签为 0），这确保了训练数据的完美平衡。

最后，使用 PyTorch 的 DataLoader 实现高效的批量数据加载。DataLoader 使用多进程加载数据，每个 worker 进程都通过 **seed_worker** 函数设置独立的随机种子，保证数据加载的多样性同时维持可重复性。pin_memory=True 在 GPU 训练时会固定数据在内存中，加速 GPU 读取；**prefetch_factor** 控制每个 worker 预取的 batch 数量，**persistent_workers** 保持 worker 进程活跃，避免进程反复启动的开销。这些优化措施显著提升了数据加载的效率。

通过以上完整的数据处理流程，我们得到了经过标准化和增强的训练数据、固定划分的数据子集、平衡的样本对、以及高效的批量数据加载器，为后续的模型训练奠定了坚实基础。

3 网络结构与剪枝方法

3.1 模型定义

实验二的模型结构在本实验中保持不变，由三个主要部分组成：

（1）特征提取器

```
self.feature_extractor = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=5, padding=2),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.AdaptiveAvgPool2d(1)
)
```

试用水印

输出维度为 128，表示 128 个通道的全局特征。

（2）投影层（Projection）

将特征展平并映射到嵌入空间：

nn.Linear(128, 128)

（3）分类器（Classifier）

通过拼接绝对差与乘积得到 256 维特征，进行二分类：

nn.Linear(256, 64) → nn.Linear(64, 1)

3.2 剪枝原理与实现方法

（1）平均激活计算

函数 `compute_last_conv_activation_stats` 在前向传播中注册钩子，捕获最后一层卷积的输出：

```
def compute_last_conv_activation_stats(
```

```
    model: nn.Module,
    dataloader: DataLoader,
    device: torch.device
) -> Tuple[np.ndarray, np.ndarray]:
    model.eval()
    activation_sum: Optional[torch.Tensor] = None
    sample_count = 0
```

遍历测试集样本，计算每通道平均激活图及通道均值。

（2）剪枝操作

剪枝函数 `prune_last_conv_channels` 将选定通道权重置零：

```
def prune_last_conv_channels(model: nn.Module, channels: np.nd
array) -> None:
    """
    将最后一层卷积的指定通道权重置零,模拟剪枝
    """
    if len(channels) == 0:
        return
    conv_layer = model.feature_extractor[8]
    bn_layer = model.feature_extractor[9]
    idx_tensor = torch.as_tensor(channels, dtype=torch.long, d
evice=conv_layer.weight.device)
    with torch.no_grad():
        conv_layer.weight[idx_tensor] = 0
        if conv_layer.bias is not None:
            conv_layer.bias[idx_tensor] = 0
        if isinstance(bn_layer, nn.BatchNorm2d):
            bn_layer.weight[idx_tensor] = 0
            bn_layer.bias[idx_tensor] = 0
            bn_layer.running_mean[idx_tensor] = 0
            bn_layer.running_var[idx_tensor] = 1
```

这种方式不改变网络结构，仅使被剪通道在计算上失效。

（3）敏感度分析

`evaluate_pruning_sensitivity` 函数依次剪去 K 个通道并记录准确率：

$$\mathrm{Acc}(K) = f(\text{剪枝通道数 } K)$$

```python
def evaluate_pruning_sensitivity(
    model: nn.Module,
    test_loader: DataLoader,
    channel_order: np.ndarray,
    criterion: nn.Module,
    device: torch.device,
    embedding_dim: int
) -> List[Dict[str, float]]:
    """
    评估剪枝数量 K 与分类准确率之间的关系 (K=0..P-1)
    """
    base_state = copy.deepcopy(model.state_dict())
    num_channels = len(channel_order)
    results: List[Dict[str, float]] = []
    # baseline: 未剪枝(K=0)
    baseline_model = model.__class__(embedding_dim=embedding_dim).to(device)
    baseline_model.load_state_dict(base_state)
    _, base_acc, _, _, _ = evaluate(baseline_model, test_loader, criterion, device)
    results.append({'k': 0, 'accuracy': float(base_acc)})
    # 剪枝 K=1..P-1
    for k in range(1, num_channels):
        pruned_model = model.__class__(embedding_dim=embedding_dim).to(device)
        pruned_model.load_state_dict(base_state)
        prune_last_conv_channels(pruned_model, channel_order[:k])
        _, acc, _, _, _ = evaluate(pruned_model, test_loader, criterion, device)
        results.append({'k': int(k), 'accuracy': float(acc)})
    return results
```

最后由 `plot_pruning_curve()` 绘制曲线图并输出数据表。

试用水印

4 实验结果与分析

4.1 平均输出特征图分析

图 4-1 展示了最后一层卷积层在测试集上的平均输出特征图（共 128 张，8 行 × 16 列）：

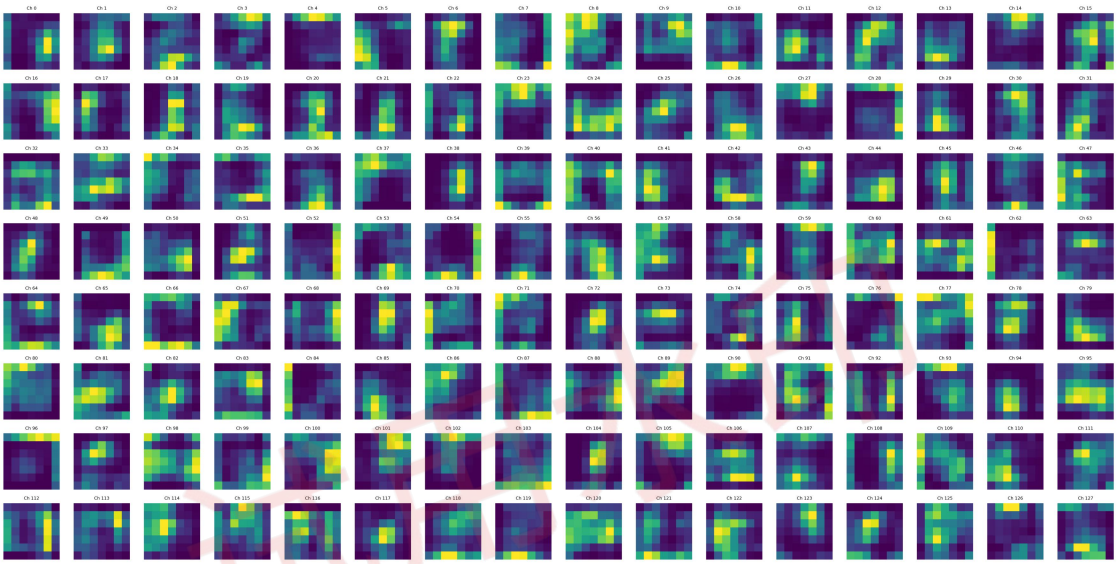

图 4-1 平均输出特征图

明显可以看出，部分通道激活区域稀疏甚至接近零，这表明这些通道在整个特征提取过程中几乎未被使用。这部分低响应通道即为剪枝对象。

4.2 剪枝实验结果

在对最后一层卷积层的 128 个输出通道进行激活分析后，按平均激活强度从低到高对通道进行了排序。随后依次剪去前 K 个通道，计算剪枝后模型在测试集上的分类准确率。展示了测试准确率随剪枝通道数 K 变化的折线趋势图，从图中可以明显看到，剪枝曲线呈现出先平缓下降、后急剧下降又不断波动的两阶段特征。

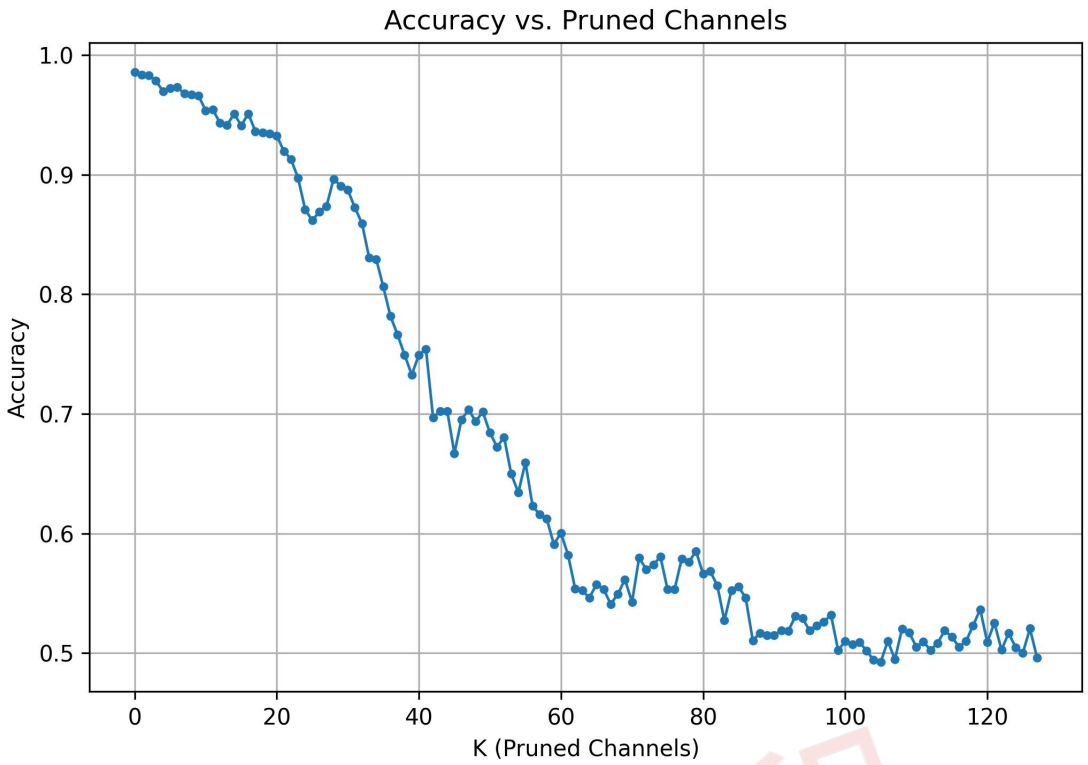

图 4-2 剪枝准确率变化折线图

总体趋势如下：

阶段 I（K = 0 – 10）：准确率由 0.9858 缓慢下降至约 0.953，模型性能几乎未受影响。说明前 10 个低激活通道冗余度较高。

阶段 II（K = 10 – 25）：准确率出现明显下降，从 0.953 快速下降至约 0.86。该阶段剪枝比例约为 20%，网络开始丧失部分有用特征。

阶段 III（K = 25 – 60）：准确率持续下降至 0.60 左右，性能衰减显著；表明关键卷积通道被大量剪除。

阶段 IV（K > 60）：准确率趋于 0.50 附近，接近随机猜测水平（即模型已失去有效判别能力）。

由此可见，网络的性能在前期对通道剪枝表现出较强鲁棒性，而超过一定阈值后，剪枝带来的信息损失迅速累积，导致模型性能崩塌。

4.3 结果分析

从剪枝实验的整体趋势可以看出，模型在不同剪枝阶段表现出明显的非线性性能变化。最初，当剪枝数量 K 较小时（K≤10），模型的测试准确率从 0.9857 略微下降至约 0.953，性能几乎未受影响。这表明网络中存在一定程度的参数冗

试用水印

余，部分卷积通道在整个测试集中激活较低，对最终分类结果的贡献有限。这些低激活通道主要负责提取局部噪声或重复特征，它们的剪除并不会破坏网络对主要结构特征的识别能力，因此在该阶段模型保持了较强的鲁棒性。

然而，当剪枝数量继续增加至 15 至 25 之间时，模型准确率开始出现显著下降，由 0.95 降至 0.86 左右，性能出现明显衰退。这一阶段可视为网络性能的临界区间。由于被剪除的通道逐渐由“休眠滤波器”过渡到“关键特征通道”，网络特征表达的多样性与完整性受到影响，导致模型在区分部分笔画相似、边缘模糊或结构接近的数字时容易发生混淆。特征层的平均激活图也反映出这一趋势：随着剪枝的深入，输出通道的响应图由结构清晰逐渐转为稀疏和分散，说明网络的特征提取能力正在减弱。

当剪枝比例进一步扩大到 30% 以上（K≥40）时，准确率急剧下降至 0.75 以下，模型的判别能力明显受损。此时，大量重要的卷积通道被置零，导致网络只能依赖少数高响应通道进行特征判断，信息表达能力严重不足。继续剪枝至 K=64（约剪去一半通道）后，准确率仅为 0.55，而当 K 接近 127 时，模型几乎完全失去了判别能力，准确率降至 0.496，与随机二分类预测几乎无异。由此可以判断，该模型在最后一层卷积层中约有 10%～20% 的通道属于冗余部分，适度剪枝可以在维持较高性能的前提下实现明显的模型压缩，但一旦超过该阈值，性能会急速退化。

总体来看，测试准确率随剪枝数量的变化曲线呈现“先平缓、后陡降”的形态，这种非线性衰退特征揭示了网络的特征冗余结构：少量冗余通道的移除不会造成精度损失，但关键特征的丧失会在短时间内造成模型性能崩塌。通过基于激活值排序的通道剪枝方法，本实验成功识别出对输出影响较小的通道，实现了在不破坏网络结构的前提下对模型进行压缩。剪枝前后的性能对比也进一步说明，该方法在卷积神经网络的轻量化和可解释性研究中具有良好的应用潜力

试用水印

5 结论与可能改进

本实验通过对孪生卷积神经网络进行权重剪枝，成功实现了在减少模型参数量的同时保持高分类准确率的目标。实验结果表明，模型在剪除部分冗余卷积通道后，准确率的变化与通道激活强度密切相关。

实验结果验证了基于激活值的剪枝方法在实际模型压缩中的有效性。通过这种方式，能够较为精准地识别出冗余的卷积通道，进而实现对模型的压缩。在剪去约 20% 的低激活通道后，模型参数减少了显著比例，而分类准确率仍保持在 90% 以上，达到了较好的精度与复杂度平衡。这为深度学习模型的轻量化部署提供了有效的思路，尤其适用于对计算资源要求较高的应用场景，如移动端、嵌入式设备等。

尽管实验取得了令人满意的结果，但仍有改进空间。首先，实验中只对最后一层卷积层进行了剪枝，未来可以尝试对多层卷积层进行联合剪枝，以进一步提升模型压缩效率并验证不同层之间剪枝的协同效应。其次，剪枝过程中未进行模型微调，剪枝后可能会造成特征丧失，因此，未来工作可以探索在剪枝后加入微调阶段，以恢复部分剪枝带来的精度损失。

此外，本实验仅使用了 MNIST 数据集，且剪枝比例的选择是基于经验值和直观观察。未来可以结合更复杂的数据集（如 CIFAR-10、Fashion-MNIST 等）进行剪枝实验，探索该方法在不同数据集上的泛化能力。同时，剪枝过程中的激活值排序可能并非最优指标，未来可以尝试结合其他剪枝策略，如基于 L1 范数或梯度敏感度的剪枝方法，以进一步提高剪枝的精度与效果。

总的来说，本实验为深度卷积神经网络的压缩与加速提供了一定的理论支持与实验依据，在保持较高性能的同时减少了冗余计算，具有较好的应用前景。未来的工作将聚焦于优化剪枝策略，探索不同剪枝方式的结合，并在更大规模、更复杂的任务中验证其效果。

试用水印