

# 1 引言

## 1.1 研究背景与研究意义

神经网络<sup>[1]</sup>，作为人工智能领域的先进模型工具<sup>[2]</sup>，其历史起源可以追溯至 20 世纪 40 年代。1943 年，美国的心理学家 Warren McCulloch 与数学家 Walter Pitts 联合提出将神经元定义为人工智能算法的核心逻辑单元，从而奠定了神经网络模型理论研究的基础<sup>[3]</sup>。然而，由于当时的计算能力所限，神经网络在早期的发展并未受到广泛的关注。随着计算硬件能力的显著提升以及反向传播算法的引入<sup>[4]</sup>，神经网络在 2010 年左右终于迎来了其辉煌时期，成为人工智能研究中的一个关键分支领域。现如今，神经网络在计算机视觉<sup>[5]</sup>、自然语言处理<sup>[6]</sup>、公共卫生<sup>[7]</sup>、交通安全<sup>[8]</sup>等众多应用场合展现了卓越的性能和成果。

在人工智能研究领域，卷积神经网络<sup>[9]</sup>和 Transformer<sup>[10]</sup>被认为是推进计算机视觉、自然语言处理等多个子领域发展的关键技术。卷积神经网络通过模拟生物的视觉感知机制，利用卷积操作处理视觉信息，借助其参数共享和高效局部特征提取的特性，已在众多视觉任务上展现出显著成效<sup>[11]</sup>。Transformer 采用自注意力机制，能够捕捉输入特征间复杂的位置关系，其性能已在多个领域超越传统方法：Transformer 不仅在语言理解和生成任务上确立了新的标准<sup>[12]</sup>，还成功地迁移到了计算机视觉等其他领域<sup>[13]</sup>，显示了其广泛的适用性。随着深度学习技术的进步，模型设计正朝着更大的规模和更复杂的结构方向发展，无论是卷积神经网络的深度与宽度的扩张<sup>[14]</sup>，还是基于 Transformer 构建的大模型通用智能时代<sup>[15]</sup>，都反映着对解决日益复杂问题的能力持续增强。

神经网络在自动驾驶<sup>[16]</sup>等对识别精度和识别速度同时具有高要求的领域中，面临着较大的部署挑战。固定规模的神经网络设计（涵盖不同的版本，如“迷你版”、“基础版”或“巨大版”）通常不能恰当地适应各种数据集的特定需求，导致其在性能或计算效率上出现不必要的牺牲，上述现象反映了灵活设计模型、提升其适应能力的需求。同时，随着神经网络规模的持续增长以及它们对计算资源的需求不断增大，相关领域探索了如轻量化神经网络设计<sup>[17]</sup>、网络剪枝<sup>[18]</sup>和网络量化<sup>[19]</sup>等方法来应对这些挑战。但是，这些模型压缩策略往往需要在模型性能上做出一定的牺牲，限制了其在需要同时满足高性能

和高效率需求的应用场景中的实用性。因此，研究和开发既能提供高性能又能快速识别，且能根据具体数据集需求进行个性化调整规模的神经网络解决方案，已成为领域内的一项重要任务。

本研究将探索一种能够根据特定数据集需求，从小到大动态扩展模型规模的方法，实现在特定数据集场景中的性能与效率的最佳平衡，促进神经网络模型的灵活性、适应性和应用范围的进一步拓展。本研究将提供如下意义：

(1) 模型规模优化：固定的模型规模通常无法满足不同数据集对性能和效率的综合要求。较大模型在资源受限的设备上会带来沉重负担，而较小模型可能性能不足。本研究将探索模型规模的适度扩张，其中将采用结构重参数化技术来实现模拟的卷积神经网络宽度增长，来最大化地控制网络的规模。本研究将实现更低规模的模型，提高模型规模与数据集需求的匹配程度，优化后的模型能够在保障所需性能的前提下，大幅降低其运行时的能源和内存需求，同时大幅提高其推理速度。

(2) 推理速度与精度的理想平衡：在深度神经网络的设计中，如何权衡推理速度与准确度始终是一项艰巨的任务——在自动驾驶、实时监控等领域，快速且准确的推理对于确保系统的安全性和效率至关重要。本研究将有效实现的精度与速度的理想平衡，使神经网络能为各类应用提供准确、快速的解决方案。这将为神经网络在广泛领域的应用及最终用户体验的提升打下基础。

(3) 应用领域拓宽与适应性提升：通过本研究将实现的在特定数据集条件下对神经网络模型进行自动规模扩张，将最大化地拓展神经网络在涵盖计算机视觉和自然语言处理等不同领域的应用范围。本研究还将进一步拓宽神经网络部署环境的适应性：通过本研究对模型性能和模型效率的高度优化，最终部署的神经网络将呈现出规模小、层次浅、宽度低、结构简洁等特点，展现出更高的推理速度和更低的部署运行的成本，使得无论是高性能计算设备还是常用的消费级计算设备，都能够高效运行本研究的自增长模型。

## 1.2 国内外研究现状

深度学习领域的模型设计正在朝着规模更大、结构更复杂的方向发展。在计算机视觉子领域中，模型设计从 VGG<sup>[20]</sup>、ResNet<sup>[21]</sup> 等传统卷积神经网络结构逐步向更大规模的 Transformer 模型<sup>[10]</sup> 如 Vision Transformer<sup>[22]</sup> 等。自然语言处理子领域更明显地反映了这种趋势，从最初的循环神经网络<sup>[23]</sup> 到 Transformer 结构的兴起，再到 BERT<sup>[24]</sup> 引领的预训练范式，甚至还出现了万亿级参数的超大规模语言模型。然而，这种趋向于更大模型规模的发展，

也带来了模型部署的挑战，尤其是在资源有限的环境中，模型的高效运行存在明显的挑战，亟需有效的解决方案来平衡模型的性能与效率。

现有的模型压缩领域包括若干研究方向。神经网络剪枝<sup>[18]</sup>通过删除不重要的连接（非结构化/半结构化剪枝）或神经元（结构化剪枝）来减少模型的参数数量和计算需求，从而提高模型的运行效率和减少内存消耗。神经网络量化<sup>[19]</sup>通过将浮点数参数（通常为FP32）转换为低位宽的数值表示，如FP16、INT8甚至更低，来降低模型的存储和计算复杂度。结构重参数化<sup>[25]</sup>关注于模型的训练时和推理时的结构转换，采用在训练阶段使用复杂、能力强的网络结构，在推理时转换为简单、效率高的结构，从而在不牺牲模型性能的情况下提升推理效率。神经架构搜索<sup>[26]</sup>旨在自动化设计高效的网络结构，通过探索最优的网络结构配置，以适应不同的数据集和硬件条件，近年来成为提升模型性能与效率的热门研究方向。

模型自增长<sup>[27]</sup>可以归类为神经架构搜索，但却不依赖耗时耗能的具体搜索过程，它通过在神经网络的训练过程中自动添加网络层和连接，以适应各异的数据集需求。对模型自增长的研究目前处于初级阶段，其研究重点通常聚焦于增长策略的设计，目前已经出现基于随机深度增长<sup>[27]</sup>、宽度增长前后保持计算等价性<sup>[28]</sup>、使用现有网络模块的动量来初始化新网络模块<sup>[29]</sup>等思想的增长策略，并取得了一定的实验成效和应用效果，但在增长策略的全面性、深度宽度增长匹配、更低规模网络搜索和Transformer架构适用性等方面存在着显著的短板。注意到模型自增长能够在保证数据集适应性的条件下，减少手动调整架构的需求和降低资源消耗这一强大优势，本研究将模型自增长作为研究对象，旨在寻找性能效率更优的数据集自适应模型。此外，本研究还将探索模型自增长在实际优化加速方面的效果，以验证其广泛应用的潜力。

### 1.3 本文主要工作和贡献

本研究提出了一种数据集自适应的模型自增长方法，通过在训练中逐步增长神经网络模型的规模，得到针对特定数据集，达到性能和效率更优平衡的模型。具体地，本研究以为模型重要表达位置添加功能多样化的特征提取结构为思路，解决了如何决定模型增长的频率、在模型的什么位置进行增长、如何选择新增长的网络模块、如何分配宽度和深度的增长、新增长的网络模块应如何初始化，以及增长后的模型应采用什么样的优化器进行优化等一系列问题。

本研究的主要工作和贡献包括：

(1) 构建模型自增长训练框架：本研究提出了一种新的训练框架，该框架允许神经网络在训练中逐步增长至适合特定数据集的规模。相比直接训练固定规模的模型，这种方法在提升训练速度、降低训练难度（包括正则化、学习率、优化器设计的简化）以及增强模型数据集适应性方面具有显著优势。

(2) 探索特定数据集下的模型性能和效率的更优权衡：在对模型进行自增长式训练的基础上，本研究继续探索通过结构重参数化为卷积神经网络提供更优的性能和效率的平衡：通过为层添加可重参数化的分支，本研究实现了模拟的网络宽度增长，这种方法避免了直接增加网络宽度可能导致的效率降低，同时确保了性能的有效提升。

(3) 拓展模型自增长训练方法的架构适应性：本文提出的自增长训练方法能够扩展至基于 Transformers 的多种主流架构，有效解决了 Transformer 架构难以训练的问题，同时为其提供了性能与效率间有效的权衡解决方案。这证明了本方法的广泛适应性，能够满足领域内多种常用模型的需求。

(4) 优化增长后模型的量化压缩效果：常规的模型量化方法以模型的过参数化为前提，探索模型的压缩空间。这导致了过参数化现象不明显的增长后模型的实际量化加速面临挑战，本研究进一步对常规模型量化方法进行优化，通过损失弥补和权重重新排布来提升量化后模型的性能，继而提升增长后模型在实际部署中的高效性。

## 1.4 论文组织结构

本文将分为五个主要部分。

第二章将对相关理论与工作进行回顾，包括神经网络架构、模型压缩和对相关工作的述评。

第三章将对数据集自适应的模型自增长方法进行介绍，其中包括模型自增长训练框架、卷积神经网络模型的自增长方案、Transformer 模型的自增长方案及相关实验验证。

第四章对自增长模型的优化量化压缩方案进行介绍，包括自增长模型的过参数化现象分析、基于误差弥补和权重重排的量化方法介绍和实际的量化加速测试。

最后，第五章将对本文的研究成果做出总结，并指出现有工作的不足之处以展望未来的工作方向。

## 2 相关理论与工作

### 2.1 神经网络架构

#### 2.1.1 卷积神经网络架构

卷积神经网络（Convolutional Neural Networks, CNN）的诞生和对生物视觉机制研究息息相关。卷积操作通过其强局部特征提取能力和参数共享机制高效地模拟视网膜对视觉信息的响应。现代卷积神经网络通过对卷积层、归一化层、激活层、下采样层和全连接层等结构的高效整合，大大推动了其在计算机视觉领域的应用。图 2-1 展示了卷积神经网络的主要模块构成。

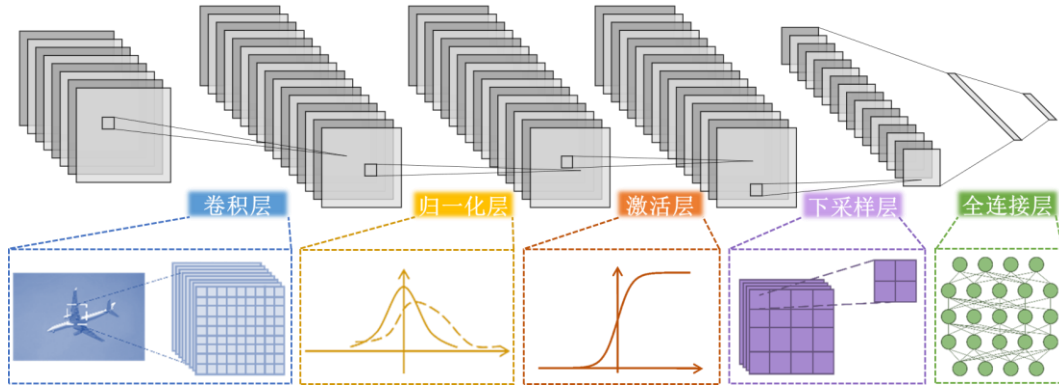
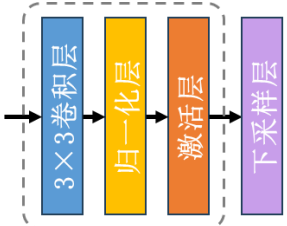
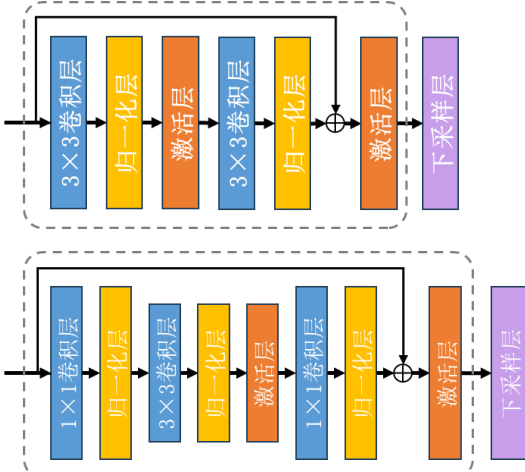
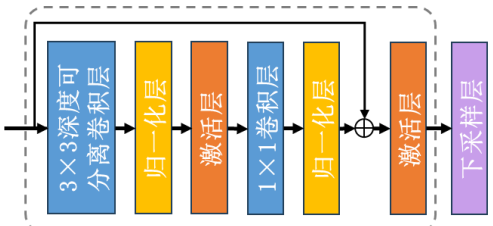


图 2-1 卷积神经网络的主要模块构成

卷积神经网络的主要特征提取器是卷积层，用来执行卷积滤波操作，具有参数共享、局部特征敏感等特点，能够有效帮助我们提取图像中的边缘、纹理、色彩等特征。卷积神经网络一般会采用批归一化（Batch Normalization）层<sup>[25]</sup>，其以批次为单位，不断将网络层的输入调整成正态分布，来减少网络训练中的梯度消失和梯度爆炸现象，帮助网络更快地收敛。激活层引入非线性，使得网络能够捕捉更加复杂的非线性特征来增强模型的特征提取能力，常见的激活函数包括 Sigmoid 函数<sup>[31]</sup>、ReLU 函数<sup>[32]</sup>、GELU 函数<sup>[33]</sup>、SiLU 函数<sup>[34]</sup>等。下采样层一般选用步长大于等于 2 的卷积层或池化层，这些层通过降低特征的空间维度（分辨率）来减少后续计算的负担，并保留更抽象的特征信息，池化层包括了最大池化层和平均池化层<sup>[35]</sup>。全连接层通常出现在网络的结尾，用于将前面层所提取到的语义信息进行整合，输出最后的分类或其他任务结果，完成智能识别过程。通过组合这些层，卷积神经网络能够有效地从视觉数据中学习复杂的特征，提升各种视觉识别任务的性能。

在卷积神经网络的发展中，一些经典的网络架构如 VGG<sup>[20]</sup>、ResNet<sup>[21]</sup>、MobileNet<sup>[36]</sup> 等，不仅在学术界引起了广泛关注，也在工业应用中展现了巨大的实用价值。这些网络通过先进的结构创新，有效地提升了其在视觉数据处理方面的性能和效率，不断刷新了各项计算机视觉任务的指标。表 2-1 展示了经典卷积神经网络的创新结构优化。

表 2-1 经典卷积神经网络的创新结构优化

| 网络        | 核心结构  | 特点  |
|-----------|---|---|
| VGG       |    | VGG 网络通过重复堆叠的小感受野(3×3)卷积层(包括归一化层和激活层)实现较大感受野，这种方法在增强网络拟合能力的同时保持了较低的数量。  |
| ResNet    |   | ResNet 通过引入捷径分支大大降低了深度神经网络的训练难度，允许构建前所未有的深度神经网络结构。ResNet 通过残差块引入的捷径分支，使得信号可以直接跨越数层，从而有效缓解梯度消失或爆炸的问题。进阶的瓶颈结构设计还能进一步提高神经网络的训练效率和推理效率。 |
| MobileNet |  | MobileNet 采用了深度可分离卷积(处理每个输入通道)技术和逐点卷积(整合特征通道)技术，减少了模型的大小和计算量，同时保持了良好的性能。  |

VGG 模型是由牛津大学提出的一种经典的、用于图像识别的卷积神经网络模型。这个模型通过重复使用核大小为 3×3 的卷积层和核大小为 2×2 的最大池化层，来替代更早期卷积神经网络的大核卷积结构，以此在保持高特征提取能力的情况下，进一步降低网络的参数量和计算量。VGG 模型有多个版本，例如包含 11 层的 VGG-11、以及更多层的 VGG-16 和 VGG-19 等。虽

然 VGG 模型的参数量和计算量依然较高，但其在图像识别领域取得了当时最优的成绩，为后续模型的发展奠定了基础。

ResNet 由微软研究院科学家何恺明提出，其作者意识到了深度神经网络模型的高训练难度问题：随着网络层次的加深，其实际表现却出现反常的下降现象。这种问题一般归因于在训练中出现了梯度消失或梯度爆炸等情况。ResNet 通过加入捷径分支，让输入不断在网络的深度方向上累加，理论上可以实现任意深度的神经网络，有效改善了深度神经网络的训练难题。ResNet 系列包括如 ResNet-50、ResNet-101 和 ResNet-152 等多个版本，较深的版本采用瓶颈结构设计，在减少参数量同时保持高性能：通过使用  $1 \times 1$  卷积层降低维度， $3 \times 3$  卷积层提取特征，再用  $1 \times 1$  卷积层扩张维度，做出了进一步的结构优化。

MobileNet 是谷歌设计的一款适用于移动设备的轻量级神经网络。MobileNet 主要涉及两种模块：一是深度可分离卷积（Depthwise Separable Convolution, DWConv）层，该层对每个输入通道独立对待，分别为其实施单通道卷积操作，相较于普通卷积层有着明显的参数量和计算量上的降低；二是逐点卷积（Pointwise Convolution, PWConv）层，用  $1 \times 1$  的卷积核整合通道信息。这样的设计既能大幅轻量化网络，又保证了较高的性能。MobileNet 系列已经发展出了多个版本，包括 MobileNet-V1<sup>[36]</sup>、V2<sup>[38]</sup> 和 V3<sup>[39]</sup>，后续版本通过引入 Squeeze-Excitation 模块<sup>[40]</sup>，通过学习通道和通道之间的相关性，对不同通道实施不同的尺度缩放，在模型结构和效率方面进行了进一步优化。

### 2.1.2 Transformer 架构

自注意力（Self-Attention）机制和 Transformer 架构的诞生，为人工智能领域带来了高性能、高自由度、多模态友好的全新基础架构<sup>[10]</sup>。

Transformer 模型最初由谷歌在 2017 年提出，以其先进的结构设计和强大的性能，改变了自然语言处理和计算机视觉的范式。区别于具备了较多先验信息的循环神经网络或卷积神经网络<sup>[41]</sup>，Transformer 采用了自注意力机制，不依赖任何先验信息，用于捕获特征内部任意距离的依赖关系。Transformer 架构极强的特征提取能力使得其可以在大规模数据上训练出卓越的性能，推动了包括机器翻译、文本生成、语言理解甚至计算机视觉等多个领域的性能突破。张量并行友好、流水线并行友好等特性进一步促进了 Transformer 模型的高训练效率，催生出大量超大规模的 Transformer 模型。

基于 Transformer 架构的若干变体已经在相关领域取得了较好的成绩，如 BERT<sup>[24]</sup>、GPT<sup>[42]</sup>、Vision Transformer<sup>[22]</sup> 等，证明了 Transformer 架构的强大潜力。

Transformer 架构涵盖了编码器-解码器范式、编码器范式和解码器范式，适应各种使用场景，但其结构基础是统一或相似的。图 2-2 展示了上述以 Transformer 为基础的编码器-解码器模型 (a)、编码器模型 (b) 和解码器模型 (c) 架构。

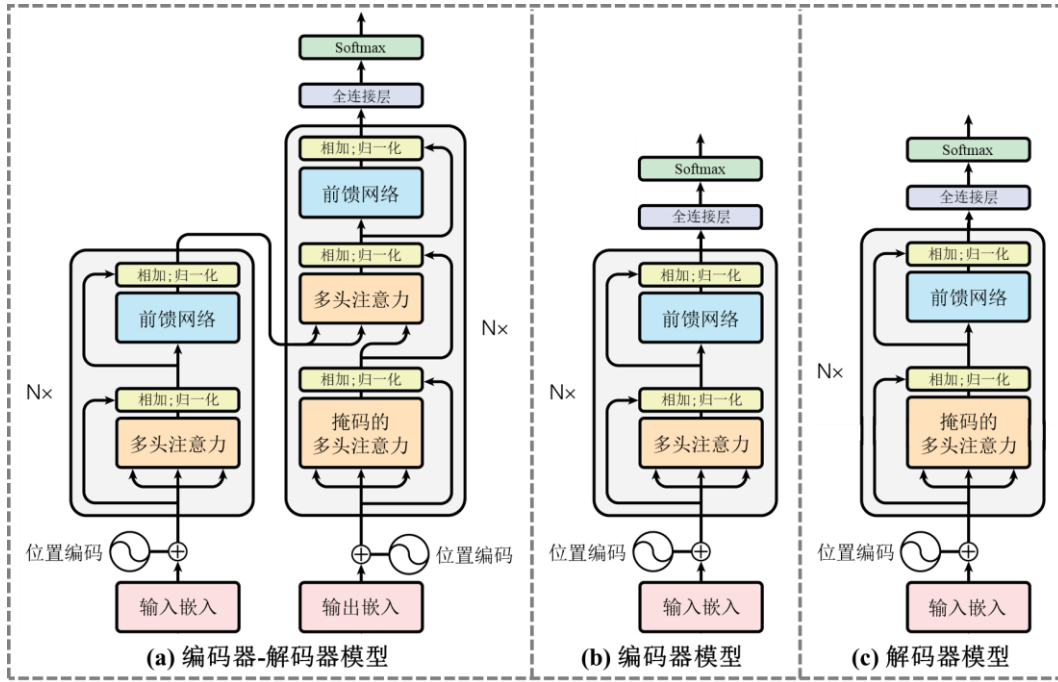


图 2-2 Transformer 模型架构

Transformer 首先将输入的令牌转换为向量，然后通过添加位置编码来指示序列中每个令牌的位置信息，位置编码会随着残差连接传递到网络的深层，确保模型能够理解特征的顺序。通过上述方式，任意特征都可以转换成统一的向量表示。

多头注意力 (Multi-head Attention, MHA) 模块和前馈网络 (Feed Forward Network, FFN) 是 Transformer 的核心。多头注意力模块通过将注意力机制“分头”独立进行，使模型能够同时从不同的表示子空间学习信息；前馈网络通过为模型引入额外的参数和非线性来提升模型的特征提取能力。具体而言，假设  $X_{l-1} \in \mathbb{R}^{n \times d}$  是 Transformer 模型第  $l$  层的输入，其中  $n$  代表序列长度， $d$  代表向量维度，那么该层的输出  $X_l$  可以表示为：

$$X = \text{Norm}(\text{MHA}(X_{l-1}) + X_{l-1}), \quad X_l = \text{Norm}(\text{FFN}(X) + X). \quad (2-1)$$

多头注意力模块聚合了  $h$  个头的输出映射，可以表示为  $W^O \cdot \text{Concat}$



( $\text{head}_1, \text{head}_2, \dots, \text{head}_h$ ), 其中 $W^O$ 表示输出映射矩阵, Concat表示聚合拼接操作, 具体单个头的计算可以表示为:

$$\text{head}_i = \text{Attention}([W^Q X]_i, [W^K X]_i, [W^V X]_i, M), \quad (2-2)$$

$$\text{Attention}(Q, K, V, M) = \text{softmax}\left(M \odot \frac{QK^T}{\sqrt{d}}\right)V. \quad (2-3)$$

其中,  $Q$ 、 $K$ 、 $V$ 分别表示查询 (Query) 序列、键 (Key) 序列和值 (Value) 序列, 其对应的映射权重分别为 $W^Q$ 、 $W^K$ 和 $W^V$ 。 $M$ 是用于对输入序列中的某些标记选择性忽略或赋予权重的掩码矩阵。前馈网络通过隐藏层扩展和收缩输入维度, 引入非线性来增强表征学习, 它由两个全连接层组成, 其权重分别表示为 $W^{FC1}$ 和 $W^{FC2}$ 。

Transformer 中的归一化一般指层归一化<sup>[43]</sup> (Layer Normalization), 其对网络中单个样本的所有激活项同步进行归一化处理, 以稳定训练过程, 促进模型的快速收敛。图 2-3 展示了多头注意力模块 (a)、前馈网络 (b) 和层归一化 (c) 的架构。

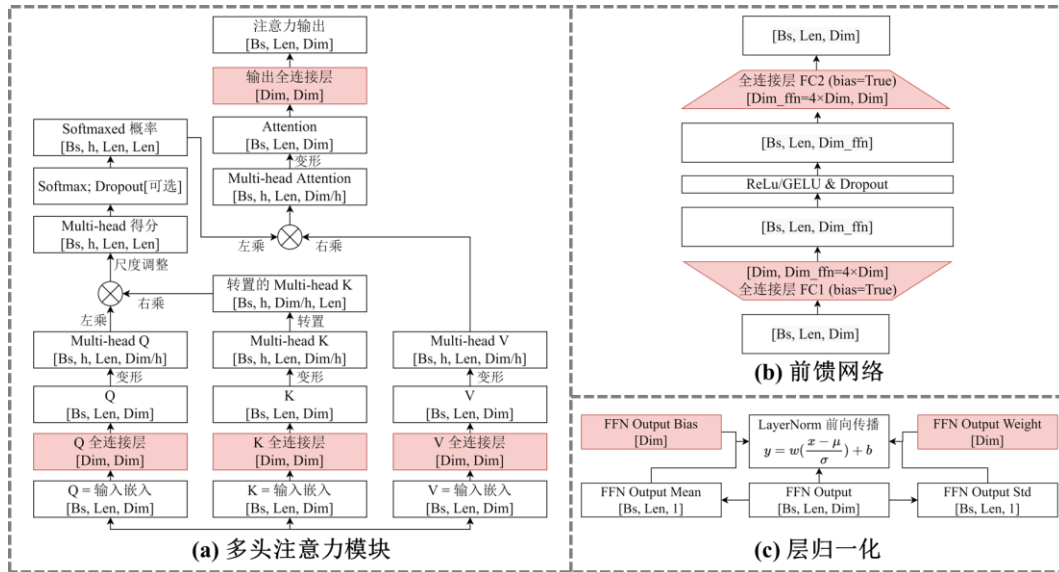


图 2-3 Transformer 架构的关键模块组成

BERT (Bidirectional Encoder Representations from Transformers) 是一种基于 Transformer 架构的, 适用于情感分类和上下文关系判断等任务的编码器模型。BERT 通过大规模的文本预训练来学习高度抽象的双向语义特征, 其预训练模型可以针对下游数据集进行微调, 以达到特定任务优化的效果。BERT 的预训练涵盖了两个主要任务: 一是掩码语言建模, 其通过随机隐藏句子中的一个或多个单词, 并预测这些空缺, 训练模型的上下文推断能力。二是下

一句预测，其通过判断两个句子是否连续，训练模型的句间关系理解能力。BERT 架构还演化出了多种改进版本，如 RoBERTa<sup>[44]</sup>（使用更大数据集和更长训练时间优化性能）和 ALBERT<sup>[45]</sup>（通过参数共享减小模型规模提高效率）等，进一步提升了该架构在自然语言处理领域的表现。

T5<sup>[46]</sup>（Text-to-text Transfer Transformer）是一个编码器-解码器模型，其把所有的任务都转换成了文本对文本（Text-to-text）范式。为了处理文本分类、摘要、翻译等多种任务，T5 首先需要在包含大量文本数据的 C4 语料库上进行大规模预训练：通过一个基于提示工程的掩码语言建模任务训练来充分 T5 的后文预测能力。接着在特定任务上微调，优化模型的下游任务表现，类似于 BERT 的预训练和微调范式。T5 通过这种统一框架大大简化了处理不同任务的方法，并在预训练阶段充分研究了不同的数据和模型规模对最终性能的影响。

GPT（Generative Pre-trained Transformer）是基于 Transformer 解码器架构的语言生成模型。它从大规模文本语料库中学习词语间的前后关系，深入地理解语言规律，完成预训练后可以通过微调适应各种任务，以文本生成为专长。GPT 模型的训练过程分为两步：首先是大规模的自监督预训练，模型学习预测文本中的下一个单词；然后在特定任务上进行微调，根据任务的需求调整参数来提高表现。目前最新的 GPT-4<sup>[47]</sup> 版本模型，极大地扩展了模型规模，增强了多模态能力，提高了数据处理的上下文长度，使得模型在语言理解、知识融合和应用适应性方面取得了显著进步，已经能够为广大的用户群体提供解决各种问题的能力。

最先将 Transformer 架构应用于计算机视觉领域的工作是谷歌大脑提出的 ViT（Vision Transformer）模型。ViT 首先将图像切割成固定大小的子图像，然后将这些子图像线性映射到序列空间并输入一个 Transformer 模型中进行特征提取。自注意力机制能够帮助 ViT 提取图像中任意两个区域之间的依赖关系，从而提升远距离视觉特征的提取能力。当拥有足够多的数据进行预训练的时候，ViT 的表现就会超过卷积神经网络，突破 Transformer 缺少归纳偏置的限制，并能够在下游任务中获得较好的迁移效果。ViT 同样演化出了多种改进版本，其中 DeiT<sup>[48]</sup>（Data-Efficient Image Transformers）通过知识蒸馏技术，显著提高了数据利用效率，使 ViT 模型仅需要较少的数据，就可以在图像分类中取得与卷积神经网络相似的性能。Swin Transformer<sup>[49]</sup> 引入了多层的、窗口尺寸可变的自注意力机制，用来有效捕获图像的局部和全局依赖，彻底刷新了多种计算机视觉任务的性能。

## 2.2 模型优化

### 2.2.1 神经网络剪枝

神经网络剪枝<sup>[50]</sup>是一种模型压缩和推理加速技术，通过修剪网络的结构来减少网络中的参数数量，降低模型的运算量和运行时资源消耗，同时尽可能少地牺牲模型的性能。神经网络剪枝主要分为结构化、非结构化和半结构化剪枝三大类别，如图 2-4 所示。

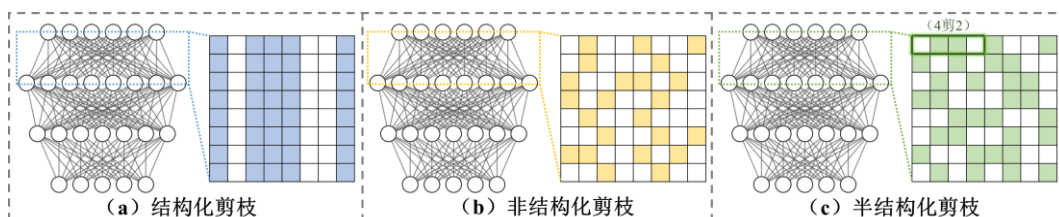


图 2-4 结构化、非结构化和半结构化剪枝

结构化剪枝主要修剪的是网络中的整体结构，例如卷积层的输入输出通道、全连接层的行或列等。这种方法的优势在于对硬件友好：实质性的网络结构简化能在广泛的硬件上实现加速，但对整体结构剪枝造成的性能损失也是不可忽视的。结构化剪枝由于其明显的加速效果，已经被广泛研究和应用。

中国人工智能科学家刘壮提出基于批归一化层的缩放因子指导的结构化剪枝方法<sup>[51]</sup>，通过引入通道级稀疏性，实现了在几乎不损失准确度的前提下对模型大小、运行时内存占用进行有效降低，这种方法直接适用于现代卷积神经网络架构，且在训练过程中引入的开销非常小。He 等<sup>[52]</sup>介绍了一种新的通道剪枝方法，通过基于岭回归的通道选择和最小二乘重构，迭代地实现对层的有效剪枝，该方法能够显著加速深层网络如 VGG、ResNet、Xception<sup>[53]</sup>等架构。DSP<sup>[54]</sup>是一种动态结构剪枝方法，自动优化每层内部通道组的最优剪枝粒度，与传统的通道剪枝相比，该方法在 GPU 上实现了更好的实际加速效果，并在不损失准确度的情况下显著减少了 ResNet-50 的计算复杂度。DepGraph<sup>[55]</sup>挑战了一种高难度的结构化剪枝任务——任意结构剪枝，提出了依赖图方法，通过显式建模层之间的依赖关系和综合分组耦合参数以实现剪枝，展示了在多种架构和任务上的广泛适用性和优秀性能。GOHSP<sup>[56]</sup>是一种图统一的异构结构 ViT 剪枝框架，其首次开发了一种基于图的排序方法，用于衡量注意力头的重要性，并将提取的重要性信息进一步整合到优化程序中，以在 ViT 模型上施加异构结构稀疏模式，GOHSP 在 ImageNet 数据集上，

对于 DeiT-Tiny 和 DeiT-Small 模型分别实现了 30% 和 35% 的剪枝率，并且准确率还有所提高。FisherPruning<sup>[57]</sup> 是一个快速的 Transformer 模型后训练剪枝框架，通过基于 Fisher 信息量的掩码搜索和掩码微调，解决了以往剪枝需要重新训练的问题，降低了推理成本。FisherPruning 应用于 BERT-base 和 DistilBERT<sup>[58]</sup> 时，实现了显著的计算量减少和推理加速，同时保持了准确率。

非结构化剪枝通过将权重矩阵中的一些元素设为零来实现稀疏化，这种方法可以极大地减少模型的参数数量，但由于优化后的稀疏权重不规则，需要特殊的硬件或软件支持来实现加速。目前，支持矩阵稀疏化计算的软硬件技术已经大量普及，非结构化剪枝已经成为模型压缩的一个重要方向。

OBD<sup>[59]</sup> 策略基于 Hessian 矩阵的概念，通过计算权重对最终损失的影响来决定权重的重要性，从而实现有效的稀疏化剪枝。这种方法早在神经网络的早期研究中就被提出，对于理解权重在神经网络中的作用和提升模型效率有着开创性的作用。人工智能科学家韩松提出了 Deep Compression 技术<sup>[60]</sup>，进一步推动了非结构化剪枝的研究，通过权重量化、非结构化剪枝和霍夫曼编码三个模型压缩 workflow，来显著减少模型的存储需求，同时尽可能保持模型的准确率，这一工作有效推动了在轻量级设备上部署神经网络模型的发展。SparseGPT<sup>[61]</sup> 作为近期的研究成果，通过在训练后引入基于 Hessian 矩阵的稀疏性和误差弥补策略，有效地减少了大型生成模型的参数量，同时保持了生成性能，展示了非结构化剪枝在处理大型模型中的潜力。

半结构化剪枝主要涵盖 N:M 稀疏化，这是一种介于结构化剪枝和非结构化剪枝之间的方法，它要求每 M 个参数中有 M-N 个是零，以便高效地将非零权重及对应的输入进行定位和计算，可以同时达到模型压缩和计算加速的效果（目前的硬件支持以 2:4 稀疏化为主<sup>[62]</sup>），这种固定模式的稀疏可以在一定程度上兼顾模型压缩的效率和硬件加速的可行性。

R-TOSS<sup>[63]</sup> 是一种半结构化的剪枝框架，用于实时物体检测，通过在 YOLO<sup>[64]</sup> 和 RetinaNet<sup>[65]</sup> 模型上的应用，展示了显著的压缩率、推理时间加速和能量消耗减少。该框架通过结合图形和优化方法，优化模型结构以适应计算资源限制的场景，展现了半结构化剪枝技术在实际应用中的潜力。Grimaldi 等<sup>[66]</sup> 聚焦于在嵌入式设备上高效处理深度神经网络的需求，提出一种通过微小的运行时修改就能利用半结构化激活稀疏性的解决方案，实现了在保持准确度的同时加速推理的目标。NxMTransformer<sup>[67]</sup> 通过 ADMM 方法<sup>[68]</sup> 优化预训练语言模型，引入 N:M 半结构化稀疏性，提高了在自然语言处理任务上的精度。

### 2.2.2 神经网络量化

神经网络量化<sup>[69]</sup> 是一种减少深度学习模型存储和计算复杂度的技术, 通过减少表示模型权重和激活的比特数来实现。这一技术在嵌入式系统和移动设备上部署深度学习模型时尤其重要, 因为这些设备的计算资源和存储空间有限。量化不仅可以减少模型的存储大小, 还能在一些硬件上实现更优的模型推理效率。从高比特到低比特的量化过程如图 2-5 所示。

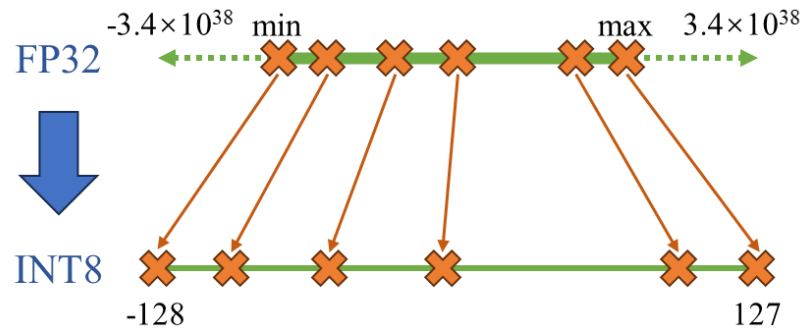


图 2-5 量化的映射过程

权重和激活是量化中两个主要的被量化对象。权重量化关注于模型的权重参数, 通过减少表示权重的比特数来减小模型大小和提高访存效率, 在不对激活进行量化的情况下, 模型的计算依然使用原始比特位, 无法提供计算加速效果。激活量化则是对模型的激活输出进行量化, 目的是提升运行时访存效率和加速推理计算。权重量化和激活量化通常结合在一起使用, 以同时实现模型压缩和推理加速。

量化可以分为对称量化 (Symmetric Quantization) 和非对称量化 (Asymmetric Quantization)。假设我们的量化目标是将浮点数量化到整数上, 对称量化会采用缩放因子 (Scale Factor)  $S$  作为唯一的量化参数, 对原始的浮点值  $V$  进行量化:  $Q = \text{round}\left(\frac{V}{S}\right)$ ,  $Q$  就是量化后的整数值。对称量化的量化和反量化的过程相对简单, 因为它仅有缩放因子这一个量化参数并且计算过程相对简洁。由于对称量化对正负值的量化精度是固定且相等的, 因此潜在的位宽浪费或不足可能会导致较高的信息损失。非对称量化相比对称量化灵活性更高, 它在使用缩放因子的基础上, 还采用了零点 (Zero-point)  $Z$  来实现浮点数到整数的映射:  $Q = \text{round}\left(\frac{V-Z}{S}\right)$ , 灵活的正负数表示范围缓解了位宽浪费和不足问题。非对称量化可以减少量化引入的误差, 适用于激活值范围不对称等情况。

ZeroQ<sup>[70]</sup> 是一种先进的量化方法，能够在不需要原始训练数据的情况下高精度地量化神经网络。ZeroQ 通过合成数据来模拟真实训练数据地分布，在卷积神经网络和 Transformer 模型上都有着较高的量化精度，适合涉及原始数据隐私的应用场景。AdaRound<sup>[71]</sup> 通过学习每个权重的最佳量化阈值（向上取整还是向下取整），有效降低了量化误差，并显著提高了量化后模型的性能。跨层权重均衡<sup>[72]</sup> 针对量化网络中的激活值分布不均的问题，通过基于计算等价性的权重缩放手段，重新将各层之间的激活调整至相对均匀，大幅提升了量化后卷积神经网络模型的精度。LSQ+<sup>[73]</sup> 是一种缩放因子可学习的对称量化策略，旨在提升低比特量化的性能。通过优化量化间隔和对称量化函数，LSQ+在多种神经网络架构上实现了较好的量化效果。Q-vit<sup>[74]</sup> 是一种针对 Vision Transformer 模型的量化方法，通过细分被量化对象的粒度以及实施针对 ViT 的有效微调，Q-ViT 能够在保持视觉处理性能的同时显著降低模型的计算复杂度和存储需求。GPTQ<sup>[75]</sup> 是一种通用的后训练量化框架，通过细粒度的权重调整和二阶量化误差最小化，有效保持了量化后模型的性能，对于处理自然语言处理任务及相关大型 Transformer 模型非常有效。

### 2.2.3 结构重参数化

结构重参数化<sup>[25]</sup>（Structural Re-parameterization）将模型拆分为推理时线性等价的一个训练时架构和一个推理时架构。其中，训练时架构通常被设计得比较复杂以获得更强的特征提取能力，例如采用多分支结构来提取特征。训练后的模型将经过一系列的图融合操作，被重参数化为一个相对简单的推理时架构，来保持模型推理时的高效率。

最经典的结构重参数化技巧是卷积层与批归一化层的融合。在训练过程中，批归一化层通过不断将层的激活约束至近似标准正态分布，来帮助模型收敛并提高其鲁棒性。具体而言，批归一化层的计算过程可以表示为： $y = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\gamma + \beta$ ，其中 $x$ 是卷积层的输出激活， $\mu$ 和 $\sigma^2$ 分别是特征通道对应的均值和方差， $\gamma$ 和 $\beta$ 是可学习的缩放因子和偏移项， $\epsilon$ 是为了数值稳定性而添加的极小常数。在模型推理阶段，可以将批归一化层的参数合并进卷积层的权重 $W$ 和偏置 $b$ 中，即得到新的卷积权重 $W' = W \cdot \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$ 和新的卷积偏置 $b' = \beta + \frac{\gamma(b-\mu)}{\sqrt{\sigma^2+\epsilon}}$ 。这种融合操作消除了推理阶段对批归一化层的依赖，减少了运算量和模型复杂度，而完全不会影响模型的输出和性能。

多分支并联待相加或多分支并联待拼接的卷积层，只要满足线性关系，就可以进行重参数化。多分支结构通常被用来增强网络的特征提取能力：通过控制不同分支的形状和尺寸差异，来实现多角度的特征提取功能，但此类结构的计算效率较低。为了提升这类结构的推理效率，可以采用重参数化技术来融合不同分支的参数，从而在不改变网络性能的前提下减少推理阶段的计算复杂度。假设各分支的卷积层分别为 $C_1, C_2, \dots, C_n$ ，其中每个卷积层 $C_i$ 对应的权重和偏置分别为 $W_i$ 和 $b_i$ 。在不失一般性的前提下，可以假设所有分支最终的输出尺寸是一致的，从而允许直接相加或拼接（不满足形状相同的情况可以提前使用零将所有分支的卷积核填充至最大宽高）。通过重参数化，可以计算出一个等效的单一卷积层 $C'$ ，其权重 $W'$ 和偏置 $b'$ 是原始各分支权重和偏置的加权和或拼接结果，即 $W' = \sum_{i=1}^n W_i$ （拼接情况： $W' = \text{cat}[W_1, W_2, \dots, W_n]$ ）以及 $b' = \sum_{i=1}^n b_i$ （拼接情况： $b' = \text{cat}[b_1, b_2, \dots, b_n]$ ）。这种方式通过单层卷积代替了多层卷积，对模型的结构进行了高度的简洁化，能够提高模型的推理速度。

结构重参数化的理论依据是先行操作要满足线性性质，这种性质使得操作能够满足交换律和结合律，根据卷积操作、批归一化操作、平均池化操作的线性性质，上述的结构重参数化范式可以轻松地扩展至分组卷积、点式卷积、深度可分离卷积等操作上，其中平均池化可以等价为权重恒等于感受像素数量分之一的卷积层。

在卷积神经网络的结构设计中，结构重参数化技术已经成为提升模型性能和推理效率的技巧之一。ACNet<sup>[25]</sup>（Adaptively Connected Neural Networks）采纳了一种可重参数化的卷积结构，包含不同尺寸（如 $1 \times 3$ 、 $3 \times 1$ 、 $3 \times 3$ 等尺寸）的并联待相加的卷积层，在不额外增加参数和计算复杂度的前提下，有效地增强了模型的特征提取能力和性能。DBB<sup>[76]</sup>（Diverse Branch Block）深入探索了各种结构的可重参数化潜力，采纳了一种更为复杂的并联待相加的重参数化结构，包括不同尺寸的卷积层、可混合串联卷积层以及平均池化层等模块，极大地强化了模型的特征提取能力。在训练结束后，可以将这些多样化的分支融合成单个卷积层来提高模型的效率。RepVGG<sup>[77]</sup>允许模型像ResNet一样训练，像VGG一样推理。在训练阶段，RepVGG为基本的卷积分支引入一个额外的带有批归一化层的捷径分支，以此增加网络的表达能力。训练完成后，RepVGG通过重参数化技术将这些分支合并为一个单一的卷积分支，实现了模型结构的简化和计算效率的提高。图 2-6 展示了上述三个经典工作的可重参数化特征提取模块。



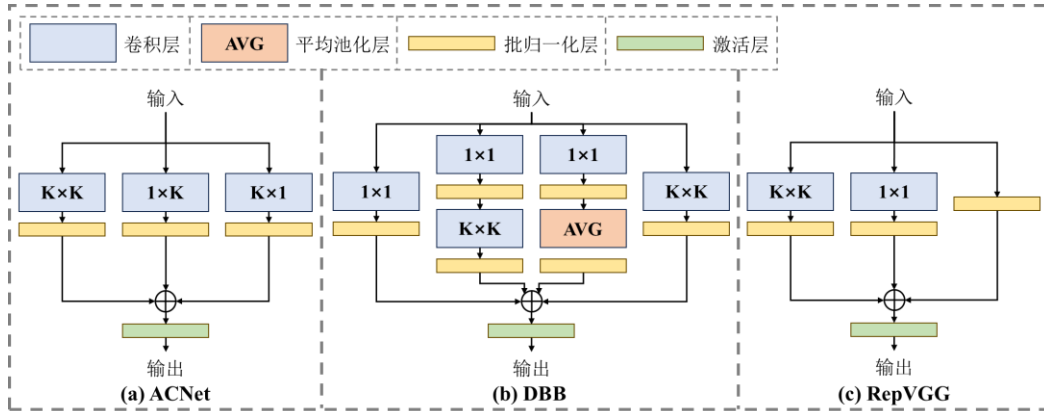


图 2-6 ACNet、DBB 和 RepVGG 的可重参数化特征提取模块

结构重参数化不仅可以用于设计高性能高效率的网络，还可以为模型优化提供更广阔的思路和方法。ResRep<sup>[78]</sup> 专注于卷积神经网络的剪枝问题，通过结构重组和带有梯度引导的微调来修剪冗余结构，以减少计算复杂度并提高推理速度。ResRep 通过精细的结构调整，使得剪枝后的网络在减轻计算负担的同时，仍能维持甚至提升原有的性能水平。RepMLP<sup>[79]</sup> 将多层前馈网络（Multi-layer Perceptron, MLP）的概念引入到卷积网络中，通过可重参数化双分支结构作为特征提取器，卷积分支用来提供强大的局部特征提取能力，多层感知机分支用来提供强大的全局特征提取能力，这种设计旨在结合卷积和 MLP 两种互补的强大优势，实现更为强大的特征提取机制。

## 2.2.4 神经网络架构搜索

神经架构搜索<sup>[26]</sup>（Neural Architecture Search, NAS）是一种自动化的方法，旨在发现性能更强、效率更高的模型结构设计模式，发现超越人类设计的高效模型架构。NAS 方法通常涉及搜索空间的定义、搜索策略（包括基于梯度、基于进化/遗传算法和基于强化学习等方法）以及性能评估策略。图 2-7 展示了神经架构搜索的基本流程。

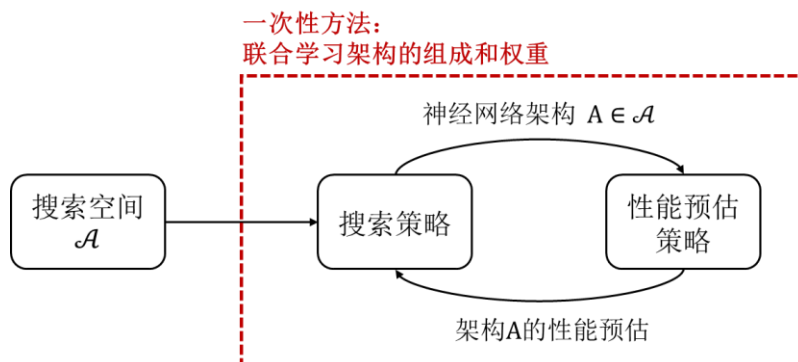


图 2-7 神经架构搜索的基本流程



神经架构搜索会不可避免地消耗大量的计算资源，但由于其能够减少设计网络所需的时间和降低对专业知识的依赖，NAS 技术仍然受到了广泛的关注。近几年神经架构搜索的研究取得了显著进展，EfficientNet<sup>[37]</sup> 是经典的通过神经架构搜索得到的网络架构，通过提出的一种复合模型缩放方法，平衡网络的宽度、深度和分辨率，显著提高了模型效率和准确性。DARTS<sup>[80]</sup> 是一种基于梯度的可微分 NAS 方法，通过将架构搜索空间放宽至连续空间，允许使用反向传播直接对架构进行优化。这种方法显著加快了搜索过程，降低了计算成本，成为后续研究的重要基础。Zoph 等<sup>[81]</sup> 通过强化学习方法进行架构搜索，利用奖励信号引导搜索过程，发掘性能更优的网络架构。这种方法利用了强化学习的探索和利用机制，推动了网络架构设计的自动化进程。MnasNet<sup>[82]</sup> 专注于移动设备的 NAS，引入了平台感知的目标函数，该函数不仅考虑模型的准确性，还同时考量模型的延迟或能效。通过这种多目标优化策略，MnasNet 能够在模型精度和运行效率之间找到最佳平衡，生成既快速又准确的模型，非常适合在计算能力和电力供应受限的移动设备上运行。Once for All<sup>[83]</sup> 是一种全新的思路，通过训练一个包含丰富网络设计选择的超网络，然后从中为不同的部署场景选择和优化子网络。这种方法极大地提高了模型的灵活性和部署效率，因为它允许从同一个超网络中为不同的硬件和应用需求定制模型，无需针对每个任务重新训练网络。AutoML-Zero<sup>[84]</sup> 以一种原始的方式使用进化算法，从最基本的数学运算开始，自动发现有效的机器学习算法。这种方法的创新之处在于它不是在预定义的网络架构空间中搜索，而是从零开始，探索新的算法结构，这对于发现非传统的、可能超出人类设计范围的机器学习方法具有重要意义。ENAS<sup>[85]</sup> 通过引入参数共享机制，在架构搜索过程中减少了重复训练模型的需要。这种方法显著降低了 NAS 的计算成本，使得在标准硬件上进行高效架构搜索成为可能。FBNet<sup>[86]</sup> 采用了一种可微分的 NAS 框架，考虑硬件约束进行卷积网络的设计。通过将硬件性能反馈整合到优化过程中，FBNet 可以直接针对特定硬件配置搜索最优的网络架构，这使得生成的网络既高效又具有高性能，特别适用于实时应用场景和资源受限的设备。

### 2.2.5 神经网络自增长

对神经网络进行自增长是本研究关注的重点，它隶属于神经架构搜索<sup>[26]</sup> 领域，但却不再需要大规模的计算资源消耗，反而可以一定程度上降低训练一个网络的成本。对神经网络自增长的探索目前仍处于初级阶段，杜克大学

最早提出了基于随机深度的模型自增长方法<sup>[27]</sup>，如图 2-8 所示。

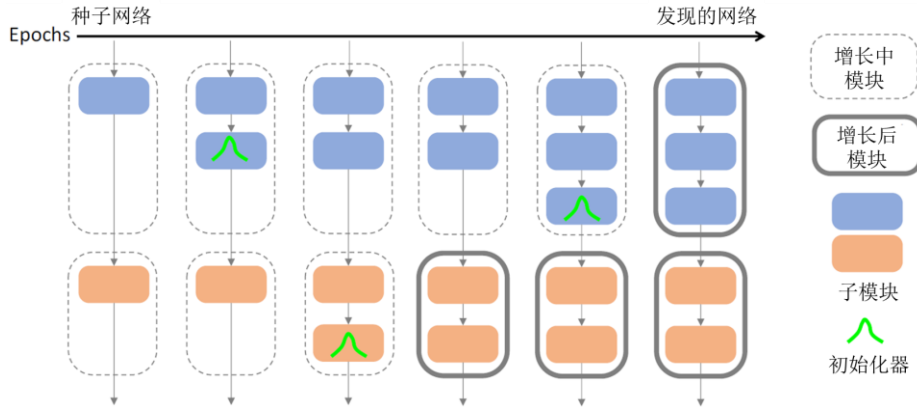


图 2-8 基于随机深度的模型自增长方法

基于随机深度的网络自增长方法预设了若干个待增长/增长中模块以及一个初始种子网络，每次随机或顺序挑选待增长模块进行结构复制并初始化，直到网络达到预设大小退出，达到了比从头训练预设大小的网络更优的性能，并有着更快的训练速度。MoGrow<sup>[29]</sup> 是一种自动化渐进学习方法，面向于应对视觉 ViT 模型训练的高效性需求，通过引入动量增长机制和自动增加训练负载的策略，实现了模型在训练过程中的无损加速，并通过对子网络架构优化问题的放松和利用弹性超网络一次性估计子网络性能，显著降低了搜索开销。SplitGrow<sup>[28]</sup> 通过动态稳定权重、激活和梯度缩放的参数化方案，以及学习率适应机制，有效地解决了因网络增长带来的训练不平衡问题，展示了与大型固定模型相比，该方法能在节省大量原始训练计算预算的同时，达到较为接近的准确性。MixtureGrowth<sup>[87]</sup> 通过重组已学习的参数，提出了一种新的网络增长方案，该方法在扩展网络大小时无需从头开始训练，而是利用已经预训练好的若干个小模型的模块进行组装式增长，既保留了已训练的参数对任务的强初始化能力，又通过新系数的灵活性为新增层权重提供了学习新知识的空间。GROWN<sup>[88]</sup> 是一种连续学习框架，该框架通过学习稀疏增长方法，仅在必要时动态增长模型，避免了传统结构基连续学习方法中的冗余全尺寸网络和复杂学习过程，展示了在多个数据集上与最先进方法相比，在准确率和模型大小方面的优越性能。Fernandes 等<sup>[89]</sup> 提出了一个两阶段算法 DNNDeepeningPruning，旨在医学成像诊断领域自动生成紧凑的深度神经网络架构，通过增加残差层块深化网络直至过拟合，然后进行剪枝以优化浮点操作数量，该方法将深度神经网络架构搜索和剪枝统一于单一框架下，展现了在两个医学成像数据集上的满意结果，提高了诊断支持的效率和准确性。

## 2.3 相关工作述评

本章在第一部分介绍了深度学习的主流架构，主要涉及卷积神经网络与 Transformer。随着对神经网络的持续研究和应用，先进模型正在朝着体积更庞大、性能更卓越的方向发展，这就使得神经网络开始面临资源受限、计算效率低下等挑战。为了优化神经网络的推理效率，各种模型压缩技术正在被广泛研究并应用。本章的第二部分详细介绍了神经网络剪枝、量化、结构重参数化等先进的模型优化方法。尽管某些模型压缩技术已经能够在几乎不牺牲模型性能的前提下提高网络的运行效率，但它们大多依赖于复杂的“预训练，压缩，微调”工作流程或需要承担昂贵的数据访问或搜索等成本。因此，探索更高效的模型优化方法，进一步提升神经网络在实际应用中的数据集合适应性效率，成为了本研究的关键目标。

2.2.4 介绍了神经网络自增长的相关工作。神经网络自增长与传统的模型压缩方法，例如剪枝，能够构成一组对偶策略：这两类方法虽然在工作流程上相反，但共享着构建尽可能高性能的紧凑网络的共同目标。具体来说，神经网络自增长通过“从小到大”的过程逐渐扩展网络结构，而神经网络剪枝则通过“从大到小”的过程精简网络结构。与其他模型压缩技术不同，神经网络自增长方法展现出更多的优势。它通过一个更直接、更简洁的工作流程来增强网络的能力，避免了需要事先训练一个庞大模型，并从这个庞大模型开始进行繁琐剪枝或复杂搜索来寻找最优架构的需求。

模型自增长的核心在于逐步、有目的地扩展网络结构，使其精确地适应特定数据集的需求，从而实现快速且高效的模型优化，找到性能更强效率更高的神经网络模型。然而，大多数模型自增长方法仅考虑了深度方向的增长，并且增长策略过于“随机化”。本研究将提出的方法则要深入考虑网络在增长过程中的结构选择、增长位置选择、增长后优化方案选择等增长策略，避免无效或过度增长带来的资源浪费，确保网络增长的每一步都朝着提高模型的性能和效率的方向推进。通过对宽度增长的精细控制与基于结构重参数化的优化，本研究所提方法将不仅能够提升模型的性能，还能够保证高效率和低资源消耗的运行。在当前追求高效与性能并重的研究大方向中，本研究相信这种自增长方法能为神经网络的发展开辟新的可能性。

### 3 数据集自适应的模型自增长方法

#### 3.1 引言

在人类大脑发育的过程中，脑容量和神经元的连接数量会随着年龄的增长而持续增加，直到发育成熟，这一过程展现了智慧生命体自然增长的智能发展模式<sup>[90]</sup>，图 3-1 以灰质体积（神经细胞的细胞体）和白质体积（神经细胞的突触）为例，展示了这种生物智慧增长模式。

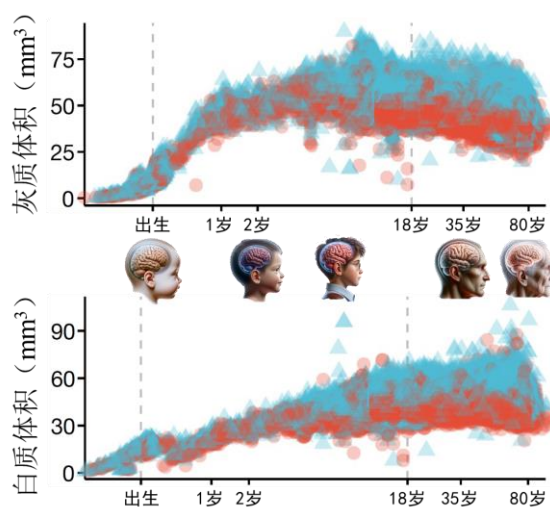


图 3-1 智慧生命体自然增长的智能发展模式

相比之下，现代人工神经网络的设计理念却截然不同，通常采取的做法是在训练之前固定网络的规模，而不是模拟智慧生物的成长过程。智慧生物的神经网络在成长过程中能够动态地调整和优化，展现出惊人的学习能力和适应性。相较而言，静态设计的人工神经网络在这些方面的潜力受到了限制：首先，从零开始训练一个固定规模（尤其是规模较大）的神经网络时，其损失面可能会呈现出更多的局部最小值和鞍点，这不仅会增加参数初始化的难度，还可能导致优化算法在这些不利点附近徘徊，进而使得收敛过程速度变慢或者完全停滞。其次，预训练模型通常提供若干个版本的固定规模网络，如 ResNet18、ResNet50 等，但这种固定规模的网络可能与特定数据集需要的网络规模存在矛盾，例如某个数据集可能更适合一个不存在的“ResNet37”版本，这种情况下，用户只能从头开始设计和训练网络，而且用户很难预先知道所需的确切网络规模。再次，流行的模型压缩方法虽旨在平衡准确度与速度，但在特定数据集中面临挑战：模型压缩不仅可能导致关键信息损失，影响性能表现，而且模型的压缩和微调可能依赖大量资源和时间，对资源受限

环境不利。与此同时，模型压缩的超参数优化过程复杂且耗时，其实施和优化还需深厚的专业知识，限制了其广泛应用。这些问题共同反映了固定规模模型训练方法在灵活性、效率和适应性方面的根本局限。

本研究旨在贴合智慧生物大脑的发展策略，探索一种自增长式的模型训练方法，让神经网络在训练的过程中自主进行结构调整。这种自增长式训练方法的目标在于使网络能够根据学习的需要动态地扩展其结构和规模，实现数据集自适应：网络规模需要根据特定数据集复杂度进行调整。这种基于模型自增长的结构优化机制，将能够避免优化过程中的不合理资源分配，提高模型的资源适应性，还能够有效探索模型性能和推理效率的理想平衡状态。

本研究聚焦于解决模型增长过程中的若干关键问题与难点，包括以什么样的频率对模型进行增长、在模型的什么位置进行增长、新增长哪些类型的网络模块、如何分配网络宽度和深度的增长、如何初始化新增长的网络模块、增长后的模型如何进行优化，以及如何确定增长的终止条件。基于以上关键问题，本章内容将详细介绍模型自增长训练框架（3.2 节），着重于针对卷积神经网络（3.3 节）和 Transformer（3.4 节）两种架构，来探讨包括模型自增长总体流程、模型自增长的决策机制、网络模块的选择和增长、初始化策略和优化器调整等影响网络自增长的因素。本研究还将通过实验验证与案例分析（3.5 节）来评估所提框架的有效性，包括数据集和实验配置的介绍、卷积神经网络和 Transformer 的数据集自适应性评估，以及与模型剪枝和其他自增长方法的对比。这些实验验证旨在验证自增长模型在不同数据集上的性能、效率和适应性，并证明其相对于现有方法的优势。最后，本研究将对本章的关键贡献做出归纳和总结（3.6 节）。

## 3.2 模型自增长训练框架

本节将探索一种全新的模型训练中动态结构调整方法，形成数据集自适应的模型自增长框架。不同于常见的固定模型规模的静态结构设计，本框架允许模型根据训练过程中的学习进展自适应地调整其结构，包括网络层的深度、各层的宽度以及相应模块的类型。核心目标是在适应特定数据集的前提下，实现模型性能和效率的理想平衡。

本框架深入考虑了增长策略的多个方面：如增长的频率以确保模型复杂度与数据集难度的平衡、增长的位置以强化模型对特定特征的学习能力、网络模块类型的选择以丰富模型的表达能力、以及如何在保持模型效率的同时分配网络的宽度和深度。此外，对新增长网络模块的初始化方法和后续优化

过程的调整也进行了深入研究,旨在确保模型增长后能够快速适应新的结构,以提高模型性能。最终,通过设定合理的增长终止条件,确保模型的增长能够适时退出,防止了无效增长和资源浪费。数据集自适应的模型自增长框架的流程如算法 3-1 所示。

---

**算法 3-1 数据集自适应的模型自增长框架**


---

**输入:** 数据集 $\mathcal{X}$ ; 一个初始网络 $\mathcal{F}(W_0)$ ; [可选]人工定义的 StoppingStrategy;

**输出:** 一个训练完成的网络 $\mathcal{F}(W_k)$ , 其规模达到数据集自适应要求;

**初始化:** 最大增长中阶段迭代轮次  $N$ ; 增长后微调轮次  $M$ 。

```

1:   $n \leftarrow 0; k \leftarrow 0; m \leftarrow 0$ 
2:   $\text{optimizer} \leftarrow \text{getOptimizer}(\mathcal{F}(W_0), \text{"SGD"})$ 
3:   $\text{criterion} \leftarrow \text{getCriterion}(\mathcal{X})$ 
4:  while  $n < N$  do
5:       $\text{train}(\mathcal{F}(W_k), \mathcal{X}, \text{optimizer}, \text{criterion})$ 
6:      if  $\text{matchStoppingStrategy}(\mathcal{F}(W_k), \text{StoppingStrategy})$  then
7:          break // 退出进入微调阶段, StoppingStrategy 可为空
8:      end if
9:      if  $\text{matchWidthGrowingStrategy}() \text{ or } \text{matchDepthGrowingStrategy}()$  then
10:          $k \leftarrow k + 1$  // 记录增长次数
11:          $\text{growLocation} \leftarrow \text{findGrowLocation}(\mathcal{F}(W_k))$  // 寻找最优增长位置
12:          $\text{growModule} \leftarrow \text{findGrowModule}(\mathcal{F}(W_k), \text{growLocation})$ 
13:          $\mathcal{W} \leftarrow \text{grownModule.weight} \cup \text{grownModule.bias}$  // 归纳新增参数
14:          $\text{weightInitializer}(\mathcal{W})$  // 初始化新增模块权重
15:         // 将新增模块插入模型
16:          $\mathcal{F}(W_k) \leftarrow \text{insertModule}(\mathcal{F}(W_k), \text{growModule}, \text{growLocation}, \mathcal{W} \cup W_k)$ 
17:          $\text{newOptimizer} \leftarrow \text{getOptimizer}(\mathcal{F}(W_k), \text{"SGD"})$  // 构建新模型优化器
18:          $\text{optimizerInitializer}(\text{newOptimizer}, \text{optimizer})$  // 初始化新优化器状态
19:          $\text{optimizer} \leftarrow \text{newOptimizer}$ 
20:      end if
21:       $n \leftarrow n + 1$ 
22:  end while
23:  while  $m < M$  do
24:       $\text{train}(\mathcal{F}(W_k), \mathcal{X}, \text{optimizer}, \text{criterion})$  // 微调增长后模型
25:  end while

```

---

本框架以数据集 $\mathcal{X}$ 和一个初始的狭窄浅层网络 $\mathcal{F}(W_0)$ 为输入,其中 $\mathcal{F}(\cdot)$ 代表网络的前向计算方式, $W_0$ 为初始的网络参数。可选的输入还包括人工定义的停止策略 StoppingStrategy,用于用户自定义网络增长的终止条件。输出为一个训练完成的网络 $\mathcal{F}(W_k)$ ,其宽度和深度均达到数据集自适应要求,对应的参数为 $W_k$ 。初始化阶段,设置最大增长阶段的迭代轮次  $N$  和增长后微调阶段的迭代轮次  $M$ 。算法从  $k=0$  和  $m=0$  开始,采用随机梯度下降法 (SGD optimizer) 优化,并根据数据集 $\mathcal{X}$ 确定损失函数 criterion。

在增长阶段中，算法在迭代中训练模型，并在每次训练后评估是否达到了增长终止条件。如果达到，则终止增长，进入微调阶段。否则，算法将评估是否需要根据宽度增长策略或深度增长策略对模型进行增长。一旦明确了增长的必要性，算法将寻找最优增长位置 `growLocation` 和最适合的增长模块 `growModule`。新增模块的权重和偏置通过特定的初始化策略进行初始化，并将该模块插入到模型的相应位置。此外，为了适应模型结构的变化，每次结构变更都需要进行优化器替换，并从旧优化器继承和初始化优化状态。在完成所有增长迭代后，算法将进入微调阶段，对增长后的模型进行进一步的训练优化，以确保模型的训练达到最终收敛。

数据集自适应的模型自增长框架能够实现模型结构的动态调整，以最佳方式适应给定的数据集，同时优化模型性能和计算资源的使用。接下来的小节将分别基于卷积神经网络和 Transformer，对模型增长的决策机制、网络模块的选择和增长以及新增模块和优化器的初始化策略等展开介绍。

### 3.3 卷积神经网络模型的自增长方案

#### 3.3.1 卷积神经网络的自增长流程

卷积神经网络在处理图像等高维数据时，会对特征进行有效提取和压缩，以逐步降低其计算复杂度。因此，卷积神经网络是一般是非各向同性的，即其不同位置维度（模型宽度）不同，并在功能上有所差异。下采样层（通常是步长大于 1 的卷积层或池化层）在这一过程中发挥着关键作用，它们能够减少数据的空间维度（即宽度和高度），从而降低计算复杂度，增加模型的抽象语义理解能力。

以一个典型的输入图像尺寸 $[1, 3, 224, 224]$ 为例，其中，该尺寸依次代表批次大小为 1，通道数为 3（对应于 RGB 三通道），以及图像的宽度和高度均为 224 像素。在处理此类输入时，卷积神经网络通过一系列卷积层和下采样层，逐步提取图像中的特征，并降低特征维度，从而实现对图像内容的有效理解。首先，假设网络的第一层是一个卷积层，采用  $3 \times 3$  的卷积核，步长设为 1，输出通道设为 32，不使用 padding，经过此层处理后，图像的尺寸会变为 $[1, 32, 222, 222]$ ，特征通道数的增加为后续的特征提取奠定了基础。假设网络接着引入了一个下采样层，以最大池化层为例，假设该层的步长为 2，池化核大小为  $2 \times 2$ ，经过这一层后，图像的尺寸将进一步减少到 $[1, 32, 111, 111]$ 。在卷积神经网络的架构中，通常会集成多次（大约 2 至 4 次）的下采样过程，

旨在逐级抽象和提炼更为高阶的特征，以便有效地从输入数据中识别出复杂的模式和结构。卷积神经网络被下采样结构分割为若干个计算区块(Block)，在每个区块内部应用一致的基础维度，随着网络深度的增加，这些区块的基础维度相应增高，而处理的特征图分辨率则逐步降低。因此，在探索模型自增长策略时，要考虑到卷积神经网络的上述特性，本研究选择一个仅包含下采样结构的网络作为增长的起始模型，并在其结构中预留空的计算区块，为后续的增长留出可能。这种初始网络设计方法能够为模型提供灵活性，使其能根据数据集的复杂度和其特性动态调整网络结构，还有助于维持模型在增长过程中的稳定性。

基于对卷积神经网络非各向同性结构和自增长初始模型的讨论，本研究进一步探索将自增长框架应用于具体的卷积神经网络架构，包括 VGG<sup>[20]</sup>、ResNet<sup>[21]</sup>（含普通架构和 Bottleneck 架构）和 MobileNetV3<sup>[39]</sup>，旨在验证自增长策略在不同网络架构上的适用性和有效性。表 3-1 展示了上述架构在两次下采样情况下的推理时模块组成情况。

表 3-1 两次下采样时的自增长卷积神经网络模块组成

| 模块            | VGG                                       | ResNet   | ResNet-Bottleneck  | MobileNetV3  |
|---------------|---|--|--|--|
| Stem × 1      | [Conv(64, 3, 3, 3)]<br>BN(64)             | [Conv(64, 3, 3, 3)]<br>BN(64)  | [Conv(64, 3, 3, 3)]<br>BN(64)  | [Conv(32, 3, 3, 3)]<br>BN(32)  |
| Block1<br>× ? | [Conv(64, 64, 3, 3)]<br>BN(64)<br>ReLU    | [Conv(64, 64, 3, 3)]<br>BN(64)<br>Add + ReLU<br>Conv(64, 64, 3, 3)<br>BN(64)<br>Add + ReLU       | [Conv(16, 64, 1, 1)]<br>BN(16) + ReLU<br>Conv(16, 16, 3, 3)<br>BN(16) + ReLU<br>Conv(64, 16, 1, 1)<br>BN(64) + Add + ReLU    | [Conv(128, 32, 1, 1)]<br>BN(128) + SiLU<br>DWConv(128, 128, 3, 3)<br>BN(128) + SiLU + SE<br>Conv(32, 128, 1, 1)<br>BN(32) + Add + SiLU |
| Down1<br>× 1  | [BN(64)]<br>Conv(96, 64, 3, 3)            | [BN(64)]<br>Conv(96, 64, 3, 3)   | [BN(64)]<br>Conv(96, 64, 3, 3)   | [BN(32)]<br>Conv(64, 32, 3, 3)   |
| Block2<br>× ? | [Conv(96, 96, 3, 3)]<br>BN(96)<br>ReLU    | [Conv(96, 96, 3, 3)]<br>BN(96)<br>Add + ReLU<br>Conv(96, 96, 3, 3)<br>BN(96)<br>Add + ReLU       | [Conv(24, 96, 1, 1)]<br>BN(24) + ReLU<br>Conv(24, 24, 3, 3)<br>BN(24) + ReLU<br>Conv(96, 24, 1, 1)<br>BN(96) + Add + ReLU    | [Conv(256, 64, 1, 1)]<br>BN(256) + SiLU<br>DWConv(256, 256, 3, 3)<br>BN(256) + SiLU + SE<br>Conv(64, 256, 1, 1)<br>BN(64) + Add + SiLU |
| Down2<br>× 1  | [BN(96)]<br>Conv(128, 96, 3, 3)           | [BN(96)]<br>Conv(128, 96, 3, 3)  | [BN(96)]<br>Conv(128, 96, 3, 3)  | [BN(64)]<br>Conv(96, 64, 3, 3)   |
| Block3<br>× ? | [Conv(128, 128, 3, 3)]<br>BN(128)<br>ReLU | [Conv(128, 128, 3, 3)]<br>BN(128)<br>Add + ReLU<br>Conv(128, 128, 3, 3)<br>BN(128)<br>Add + ReLU | [Conv(32, 128, 1, 1)]<br>BN(32) + ReLU<br>Conv(32, 32, 3, 3)<br>BN(32) + ReLU<br>Conv(128, 32, 1, 1)<br>BN(128) + Add + ReLU | [Conv(384, 96, 1, 1)]<br>BN(384) + SiLU<br>DWConv(384, 384, 3, 3)<br>BN(384) + SiLU + SE<br>Conv(96, 384, 1, 1)<br>BN(96) + Add + SiLU |
| Head × 1      | [AVG_POOL(128)]<br>FC(Classes, 128)       | [AVG_POOL(128)]<br>FC(Classes, 128)  | [AVG_POOL(128)]<br>FC(Classes, 128)  | [AVG_POOL(96)]<br>FC(Classes, 96)  |

为了更充分地提取高分辨率图像的抽象特征，除了两次下采样以外，本研究还设计了包含三次下采样的模型架构。在这一设计中，MobileNetV3 和



其他三种架构对应的四个区块的基础维度分别为[16, 32, 64, 96]和[64, 96, 128, 256]，不难发现，本研究选择了相对较低的基础维度，原因在于我们将采用一种基于结构重参数化的层宽度增长策略，该策略能够在不增加计算复杂度的前提下，模拟性地扩展网络层的宽度。低基础维度的初始设置为模型提供了更大的灵活性和更高的最终推理效率，使其能够更加有效地适应不同规模的数据集需求。以上选定的网络架构因其先进的结构设计和广泛的应用场景，成为了本研究实验验证的理想对象。

在明确了初始模型以及关注的具体模型架构之后，本研究正式进入卷积神经网络自增长方案的探索阶段。图 3-2 从整体上展示了本研究提出的卷积神经网络的自增长流程。

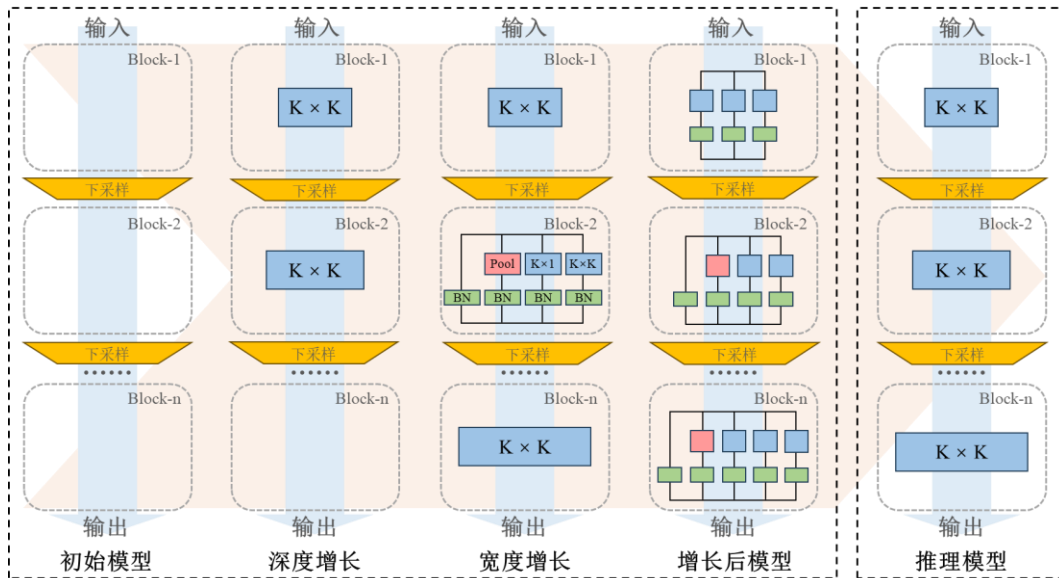


图 3-2 卷积神经网络的自增长流程

如图 3-2 所示，本研究采用的初始模型为一个“空”模型，其特点是仅包含必要的下采样层而不包括任何具体的计算区块。这种设计使得初始模型的参数数量极为有限，以 VGG 架构中采用两次下采样为例，初始模型的参数总量仅约为 169K 个，而一个完整的 VGG-11 模型拥有约 132M 个参数。因此，相对于完整模型的庞大参数规模，初始模型的参数量几乎可以忽略不计。这一极简的初始设计不仅显著降低了模型的复杂度，使训练前期的时间和能耗成本大幅降低，而且为后续的自增长过程提供了灵活性和可扩展性，为模型对特定数据集进行自适应打下了良好的基础。

以初始模型作为起始点，本研究中提出的卷积神经网络自增长框架将根据既定的规则（3.3.2），对模型进行系统的深度与宽度扩展（3.3.3）。深度扩展主要涉及向预留的“空”计算区块中加入新的层，而这些新增层的具体类

型则依赖于所采用的模型架构，详见表 3-1。理论上，每个预留的计算区块具备容纳无限数量层的潜能，图 3-2 中仅示意性地为每个区块展示了单个层以便理解。

宽度扩张主要涉及向网络中现有的一层或若干层（图 3-2 仅展示了对区块中标准层的宽度扩张，而对下采样层的同样适用性未予展示，以简化视觉呈现）的主要特征提取结构添加可重参数化的分支，以增强模型的特征提取能力。以 VGG 架构为例，该架构的核心模块来自对  $3 \times 3$  尺寸的卷积层、批归一化层以及 ReLU 激活层的堆叠。在此研究框架下，我们将“Conv-BN”模块视为主要的特征提取单元。例如，若需增加“Conv(64, 64, 3, 3)-BN(64)”模块的宽度，就可以将一个“Conv(64, 64, 1, 1)-BN(64)”模块与原有模块并联后相加。此种并联相加的结构不仅在理论上扩大了网络的宽度，还具备多重优势。首先， $3 \times 3$  卷积核与  $1 \times 1$  卷积核的并联使用相比于单独使用  $3 \times 3$  卷积核，能够在不显著增加训练负担的情况下，增强网络的特征提取能力。其次，小型卷积核的加入，有助于网络在保持特征空间维度不变的同时，实现特征通道间的线性组合，进一步提升了模型的表达能力。最后，通过结构重参数化的方式引入的宽度扩张，能够在不增加额外推理效率负担的前提下，增强模型的性能。

当有新的模块被添加到网络中时，对其参数进行恰当的初始化成为确保其能够无缝融入原始网络训练节奏的关键步骤。此外，新模块的引入导致网络的参数结构发生变化，这一变化同样要求优化器进行相应的调整，以适配更新后的网络架构（3.3.4）。

自增长结束后的网络展现为一个具有多层、多分支的复杂架构。经过结构重参数化过程之后，这一多层、多分支架构得以转化为一个更为紧凑且简洁的多层、固定分支架构。通过网络的自增长和结构重参数化带来的结构转变，模型不仅能够更好地适应特定的数据集需求，同时也在精度和计算速度方面实现了显著的优化，从而达到满足实际数据集情况下的模型性能和推理效率的理想平衡。

本研究接下来会深入研究卷积神经网络自增长框架的核心要素，涉及决策机制、网络模块选择和增长，以及如何调整初始化策略和优化器。为了检验这些策略的成效，我们计划在 CIFAR-10 数据集<sup>[9]</sup>（数据集详细介绍请参考表 3-17）上构建一系列实验，专注于对 VGG 模型的自增长。这些实验不仅能展示自增长框架在实践中的效果，还能具体分析各种策略对模型性能的影响，并得到一组最优的自增长策略。

### 3.3.2 模型自增长的决策机制

模型自增长的决策机制主要由以下三个要素构成：模型单次增长的规模、增长后模型进行训练的轮次数以及模型增长的终止条件。为了深入探究这些要素，本研究采用随机梯度下降法作为优化算法，以 Cosine 作为学习率（初始设置为 0.1）调整机制，在 CIFAR-10 数据集上对模型进行了 100 个轮次的自增长训练。在模型自增长过程中，为了确保实验的严谨性并有效控制变量，我们固定采纳了继承式的新模块初始化策略、继承式的优化器初始化策略、宽度深度单次交替增长策略、贪心式增长位置选择策略以及随机有放回的角度增长模块选择策略，这些策略将在 3.3.3 和 3.3.4 中深入讨论。

#### （1）模型单次增长的规模

在模型的单次深度增长中，可以选择为模型增长单个层或多个层，同样地，在单次模型的宽度增长中，也可以选择为模型的单个层或多个层增长宽度。因此，深入探讨在单次模型增长过程中应采取何种规模进行深度或宽度扩张，对于优化模型结构、提升性能以及保证计算效率具有重要的理论和实践价值。

首先，本研究将探讨模型在单次深度增长过程中不同的增长规模对模型自增长效果的具体影响。本研究将增长后模型的训练轮次数固定为 3 轮，并固定模型增长的终点为每个计算区块包含两个子层的结构，即形成“VGG-2-2”模式。同时我们固定每次宽度增长将涉及原有层数的 50%。本研究考察了在单次增长 1 至 6 层时模型自增长后的性能差异，如表 3-2 所示。

表 3-2 单次深度增长的规模对模型自增长性能的影响

| 单次深度增长规模      | 1 层       | 2 层   | 3 层   | 4 层   | 5 层  | 6 层   |
|---------------|-----------|-------|-------|-------|------|-------|
| 增长后架构 (Arch)  | VGG-2-2-2 |       |       |       |      |       |
| 参数量 (#Params) | 706.35K   |       |       |       |      |       |
| 计算量 (FLOPs)   | 161.38M   |       |       |       |      |       |
| 模型性能 (acc%)   | 92.01     | 91.33 | 91.35 | 91.88 | 90.7 | 91.04 |

实验结果表明，模型性能受单次增长层数的影响显著。当每次仅增长一层时，性能最优，达到 92.01%，显示单层增长是扩展深度的最优策略。随着每次增长的层数变多，性能呈下降趋势，尤其是增长至 5 层时下降最明显，达到了 90.7%。这种情况主要由两个因素造成。首先，快速增长多层虽能在理论上捕捉更复杂特征，却因结构突然扩大而引入过多新参数，这些新参数短时间内难以和现有结构有效结合，影响了模型的学习效率和整体表现。其次，在初期使用较高学习率时，如果一次增加太多层，可能导致参数调整过

于激进，引发训练的不稳定性。

进一步，本研究将深入分析单次宽度增长对自增长模型性能的影响。与先前分析的深度增长不同，宽度增长涉及为模型的单个或多个层引入新的可重参数化分支，以增强其主要特征提取能力。考虑到单个分支的参数量非常有限，本研究不单独考虑对单层进行宽度扩张的情况，而是采取按比例扩张的策略，例如选取现有网络层中的 50%进行宽度增长。为保证研究的一致性和可比较性，我们将在实验设置保持一致的情况下沿用前文实验中得到的最优结论，例如单次深度增长的规模为 1 层。单次宽度增长的规模对模型自增长性能的影响如表 3-3 所示。

**表 3-3 单次宽度增长的规模对模型自增长性能的影响**

| 单次宽度增长比例      | 0% (仅深度增长) | 30%   | 50%   | 70%   | 100% (全增长) |
|---------------|------------|-------|-------|-------|------------|
| 增长后架构 (Arch)  | VGG-2-2-2  |       |       |       |            |
| 参数量 (#Params) | 706.35K    |       |       |       |            |
| 计算量 (FLOPs)   | 161.38M    |       |       |       |            |
| 模型性能 (acc%)   | 90.88      | 92.05 | 92.01 | 92.42 | 92.09      |

实验结果表明，单次宽度增长的规模对模型的性能有所影响。相较于仅深度增长的模型性能（90.88%），模型在宽度增长比例为 30%时性能提升至 92.05%，并且在增长比例达到 50%和 70%时性能分别为 92.01%和 92.42%，显示出性能的进一步提升。这种现象反映了适当高比例的宽度增长在增加参数时能够考虑模型复杂度的逐步扩展，且能够与学习率的动态调整相结合，使得模型在训练过程中对参数增加展现出更好的容忍度，从而有效地利用新增参数捕捉更复杂的数据特征，提升性能。然而，当宽度增长比例提升到 100%，即所有层都进行宽度增长时，模型性能略微下降至 92.09%。这说明当所有层都增长导致模型复杂度进一步上升时，并非每层的增长都能有效提升性能。整体来看，在执行宽度增长时需寻找一个平衡点，发挥宽度增长的性能潜力，同时避免过度增长带来的性能下降。

## （2）增长后模型进行训练的轮次数

确定模型每增长一次后应间隔多少轮次再进行下一次增长，反映了模型自增长的频率，是决定着模型自增长性能的重要因素之一，因为它直接影响到模型对新结构的适应能力和整体训练过程的效率。合理设定这一周期可以确保模型有足够的时间来调整和优化新加入的结构，从而充分利用每次增长带来的潜在性能提升。

在实验配置统一的基础上，本研究详细考察了增长后模型进行训练的轮次数在 1 至 9 轮不等的情况下模型的性能差异，如表 3-4 所示。

表 3-4 增长后模型的训练轮次数对模型自增长性能的影响

| 增长后训练轮次       | 1     | 2     | 3         | 5     | 7     | 9     |
|---------------|-------|-------|-----------|-------|-------|-------|
| 增长后架构 (Arch)  |       |       | VGG-2-2-2 |       |       |       |
| 参数量 (#Params) |       |       | 706.35K   |       |       |       |
| 计算量 (FLOPs)   |       |       | 161.38M   |       |       |       |
| 模型性能 (acc%)   | 91.90 | 92.38 | 92.42     | 91.98 | 91.31 | 90.17 |

实验结果表明，模型自增长后所进行的训练轮次数对模型性能有着明显的影响。当增长后的训练轮次数为 1 至 3 轮时，模型性能逐步提升，其中以 3 轮训练后模型性能最优，达到了 92.42% 的准确率。这表明了，如果增长后的训练轮次设置较低，则会加快模型自增长的速度，可能使得模型无法有效整合新的层或模块，降低模型达到最优性能的潜力。适度选择模型增长后的训练轮次能够有效利用模型自增长带来的结构优化，使模型充分适应新的参数并实现性能提升。然而，随着增长后的训练轮次数继续增加至 5 轮及以上，模型性能开始呈现下降趋势：在一定的增长后训练轮次之后，模型对原有结构的适应性达到饱和，难以适应新增结构，假设增长前模型的参数为  $W$ ，并且已经在较多的训练轮次中训练至拟合或接近拟合，即  $W = W^*$ ，此时为模型增加新参数  $\Delta W$ ，那么新模型的损失函数将按照泰勒公式如下展开：

$$L(W^* + \Delta W) \approx L(W^*) + \nabla L(W^*)^T \Delta W + \frac{1}{2} \Delta W^T H \Delta W \quad (3-1)$$

其中， $H$  是损失函数在  $W^*$  处的 Hessian 矩阵，由于  $W^*$  是定值以及  $W$  已经被训练至拟合或接近拟合的事实 ( $\nabla L(W^*)^T \approx 0$ )，对新的损失函数  $L(W^* + \Delta W)$  的优化将非常容易向着将  $\Delta W$  训练至接近 0 值的方向进行，这就代表着新增参数  $\Delta W$  没有为性能优化带来贡献或仅提供了少量贡献。因此，对模型增长后训练轮次的过分增加可能会因为优化失效等问题导致性能下降。

### (3) 模型增长的终止条件

对模型自增长终止条件的研究至关重要，这不仅影响着模型的最终性能，也直接关联到模型的推理效率和模型在特定数据集中的实际应用价值。合理设定模型自增长的终止条件能够确保模型在达到性能瓶颈之前停止不必要的结构扩张，从而在保证模型精度的同时，避免过度复杂化。为了满足不同背景用户的需求，本研究准备了多样化的模型自增长终止条件，照顾到了各个领域的专家用户，以便根据其自身专业知识和实际应用场景灵活选择。

对于模型架构领域的专家，本研究提供的框架允许用户精确地定制最终的模型架构，如使用例如“VGG-2-2-2-2”或“ResNet-3-4-5-3”这样的架构标识提供约束。这样的设计方式保证了模型发展的方向与预期一致。在一些先进的模型设计中，如 Swin Transformer<sup>[49]</sup> 和 ConvNeXt<sup>[92]</sup> 等架构，都使用过

诸如“2-2-18-2”、“3-3-27-3”等模式定义模型的架构，这反映了模型设计者对模型各个区块功能的认识：模型的第三个计算区块起到了从提取中级特征到提取高层抽象语义的关键过渡作用，增加该区块中的层数被视为一种提高模型特征提取能力的有效方法。这种允许专家手动设计最终模型架构的方式，能够充分利用专家的先验知识和领域经验，从而设计出高度优化的模型。

对于应用领域内专家，他们虽然可能不太熟悉模型的架构细节，但对模型的整体规模和性能有所预期。本研究为此提供了一种基于参数量的模型自增长终止条件。这一机制允许用户通过设定模型重参数化后的最大参数量来指导模型的增长（例如设置最高 20M 参数）。为了防止宽度增长的过分扩张（宽度增长不直接影响重参数化后的模型参数量），本研究额外设定重参数化前的模型参数量不得超过设定最大参数量的 1.5 倍，从而在控制模型规模的同时，确保计算资源的有效利用。这种策略的灵活性更强，如果用户侧重宽度增长，最终可能得到一个相对浅而宽的模型，这样的模型推理成本较低（宽度方向计算可并行），但性能提升幅度可能有限。相反，如果用户侧重深度增长，最终可能得到一个深而窄的模型，这可能意味着更高的推理成本（深度方向计算不可并行），但同时也有潜力带来较大的性能提升。本研究后续还将探索一组旨在寻找宽度和深度增长之间最优分配方式的策略(3.3.3)，帮助用户寻找性能和效率的理想平衡。

对于来自其他领域的专家，本框架提供了能够自动决定模型增长终止的机制。假设当模型经过一段时间的增长和训练后，性能提升已趋于饱和，则自动终止模型的进一步增长。该思路源于对模型规模与性能关系的观察，如图 3-3(a)所示，在模型规模与性能关系曲线中，存在一个最优切点，其能够代表在性能提升趋于饱和且模型规模最小化的模型状态。

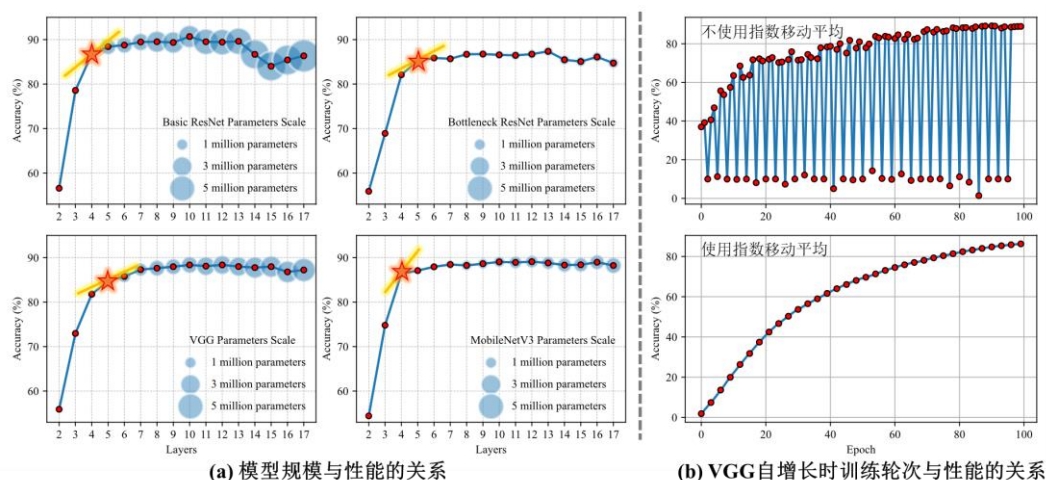


图 3-3 模型规模、自增长训练轮次与模型性能的关系

然而，正式的模型自增长训练并不会满足模型规模与性能的关系曲线，假设模型每三轮训练后进行一次增长，其规模与性能的关系就会呈现出如图 3-3(b)上图所示的频率波动性：每次新增参数后，模型的性能会出现大幅度的下降。这是因为每次模型在引入了新的参数之后，需要经过一段时间（约一个轮次）的训练才能重新适应并体现在性能提升上。

为了有效处理由模型增长引起的性能波动，本研究采用了指数移动平均<sup>[93]</sup>（Exponential Moving Average, EMA）的方式来平滑性能曲线。指数移动平均通过赋予最近的观测值更高的权重，能够有效地减少短期波动对性能趋势判断的影响，从而更加准确地捕捉到性能的长期趋势。

图 3-3(b)下图展示了添加了指数的移动平均以后的性能曲线，该曲线可以使我们更加平稳地观察模型每次自增长后性能的变化。如果在上一次模型增长后的若干轮次训练中，模型的性能提升没能达到特定阈值（默认设置为 3%，接受用户自定义），则认为模型的性能已经趋于饱和，模型自增长终止。为了适应模型在不同训练阶段的性能波动特性，模型自增长终止的条件可以适当放宽，即允许模型在自增长的过程中存在一定次数的性能增长没能超过阈值的情况（该次数默认为 1 次，接受用户自定义）。这种灵活调整的机制确保了模型自增长策略在追求性能接近饱和的同时，也兼顾了实际训练过程中的波动和不确定性。

**表 3-5 终止条件对模型自增长的影响**

| 终止条件  | 控制架构      | 控制参数量     | 自主控制      | 自主控制      | 自主控制      |
|-------|-----------|-----------|-----------|-----------|-----------|
| 控制条件  | VGG-3-3-3 | 1M        | 默认        | 阈值 6%     | 容忍度 3     |
| 增长后架构 | VGG-3-3-3 | VGG-4-3-3 | VGG-2-3-2 | VGG-1-2-2 | VGG-3-4-3 |
| 参数量   | 974.47K   | 1.01M     | 789.58K   | 669.29K   | 1.06M     |
| 计算量   | 230.20M   | 268.21M   | 182.72M   | 123.37M   | 251.53M   |
| 模型性能  | 93.01     | 92.74     | 92.14     | 92.08     | 92.57     |

表 3-5 展示了在控制其他影响因素为最优的情况下，不同模型自增长终止条件所得到的模型的性能和效率差异，在总体上，每种自增长终止条件都找到了相对高性能高效率的神经网络。

综上所述，本研究通过提供一系列灵活多样的模型增长终止条件，充分满足了不同用户背景和需求的多样性。无论是对模型架构有深刻理解的专家、还是领域内外的专家用户，本框架均能提供适宜的支持与选择。这种设计方式不仅提高了框架的通用性和适应性，也为实现高效且性能优化的模型自增长提供了坚实的基础。



### 3.3.3 网络模块的选择和增长

在模型自增长的过程中，网络模块的选择与增长策略对于优化模型性能和提高计算效率至关重要。本小节将专注于探索深度增长和宽度增长的决策方案，以及如何有效地协调深度与宽度增长以寻求更优的模型结构。本小节实验继承了 3.3.2 中确定的最优模型自增长决策机制，并继续采纳了继承式的新模块初始化策略、继承式的优化器初始化策略，以确保实验的严谨性并有效控制变量，这些策略将在 3.3.4 中深入讨论。

#### (1) 模型深度增长的决策方案

深度增长策略是自增长模型最终性能和效率的重要影响因素。对于非各向同性的卷积神经网络而言，由于这类网络计算区块间的层配置存在差异，而区块内部的层配置保持一致，因此，待增长层的类型通常是预先设定好的。因此，深度增长的研究焦点主要集中在确定最佳的深度增长位置上，包括选择在哪个计算区块以及区块内的具体哪个位置进行层的增长。本部分将详细探讨这一决策过程，旨在通过精确的深度增长位置选择，实现模型结构的优化和性能的提升。

本研究探索了三种主要的增长位置决策方法：随机选择、顺序选择和一种创新的贪心选择。随机选择方法不依赖于特定的规则，随机确定计算区块及其内部位置进行层的增长；顺序选择方法按照固定的顺序遍历计算区块，并在每个选中的区块的末尾添加新层；贪心选择方法则基于网络在训练过程中展现出的动态重要性进行决策，优先在识别为最重要的区块及位置增加深度。本研究通过分析网络中的原生的缩放行为来识别重要的增长位置，如批归一化层的权重项，该权重项可以看作是一种对输出特征图重要性的量化，从而为深度增长提供了一种基于网络性能反馈的选择依据，如图 3-4 所示。

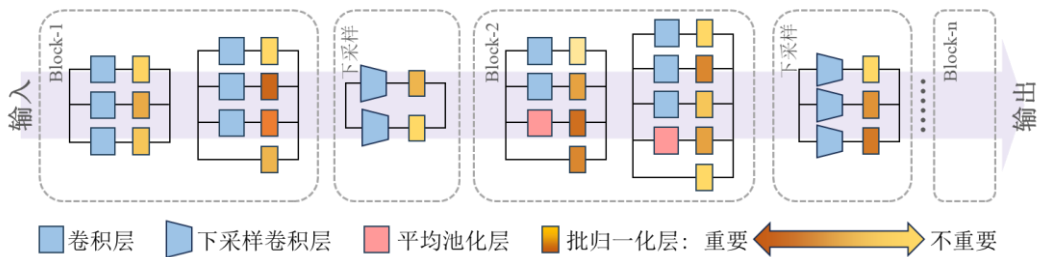


图 3-4 神经网络中原生存在的缩放的重要性

在实施贪心选择策略时，本研究首先收集所有计算区块外的批归一化层的缩放权重。考虑到某些批归一化层可能存在于多分支并联结构中，本研究



首先对这些层的缩放权重进行平均处理，从而得到反映每个计算区块重要性的缩放因子。在深度增长决策过程中，这些缩放因子的绝对值被用作衡量区块重要性的标准，本研究优先选择缩放因子绝对值最大的计算区块——即在网络性能中扮演最重要角色的计算区块进行深度扩展。接着，通过类似的方法，在选定的区块内部进一步识别并选取最重要的层作为具体地深度增长位置。表 3-6 展示了在控制其他影响因素为最优的情况下，贪心选择策略与随机选择和顺序选择策略的性能对比。

**表 3-6 深度增长策略对模型自增长性能的影响**

| 深度增长策略        | 随机选择  | 顺序选择      | 贪心选择  |
|---------------|-------|-----------|-------|
| 增长后架构 (Arch)  |       | VGG-2-2-2 |       |
| 参数量 (#Params) |       | 706.35K   |       |
| 计算量 (FLOPs)   |       | 161.38M   |       |
| 模型性能 (acc%)   | 92.07 | 92.16     | 92.42 |

实验结果表明，不同深度增长策略对模型自增长性能的影响。采用贪心选择策略的模型在性能上达到了 92.42%，相较于随机选择和顺序选择策略，分别提升了 0.35%和 0.26%，验证了为模型中重要表达位置的进行模块增长的有效性。这种基于缩放因子重要性分析的策略，不仅精准地识别了网络中关键的增长位置，还为模型结构的动态优化提供了一条明确的路径。

## (2) 模型宽度增长的决策方案

宽度增长在本文所研究的模型自增长中承担着提升模型的多样化特征提取能力的关键功能。相较于会真实提升模型宽度的增长方式，本研究采纳了基于结构重参数化的模拟宽度增长方案，以达到在提升模型特征提取能力的同时，不影响模型的高推理效率。基于这种方式，本研究将深入探索哪些层需要进行宽度增长以及进行怎样的宽度增长。

继承自前文所证明的，基于贪心选择机制以识别高重要性层的方法在模型深度增长中的有效性，本研究进一步将此思路应用于宽度增长的决策过程。具体而言，本研究首先综合评估了包括下采样层在内的各网络层的重要性缩放因子，全面捕捉模型中所有层对最终性能贡献的差异性。然后选取重要性最高的若干（根据 3.3.2 中对单次模型增长规模的探究，默认选取重要性排名前 70%的层）层作为即将进行宽度增长的候选层。

明确了宽度增长的候选层后，接下来要确定如何通过添加可重参数化分支有效地增强网络的特征提取能力。本研究以原始单分支层的最大感受野为基准，归纳了各种子感受野的大小，并在此基础上引入了如平均池化层和独立的批归一化层等常见的可重参数化分支，构成了丰富的待插入分支的搜索

空间，如图 3-5 所示。这一过程不仅遵循了可重参数化的原则，以保持模型推理时的高效性，还通过感受野大小的差异性引入了功能上的多样性。

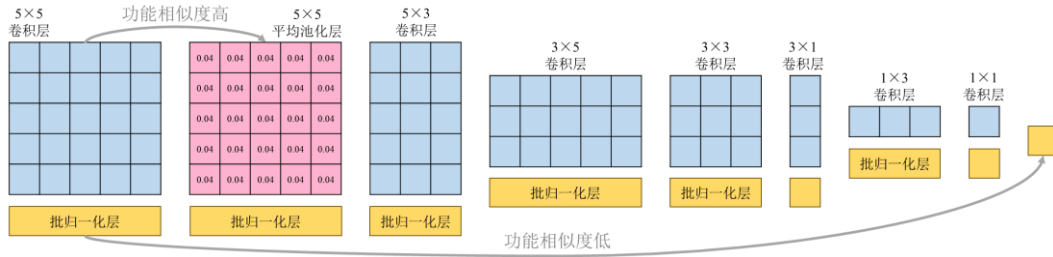


图 3-5 宽度增长中待插入分支的搜索空间

当自增长模型中的某一层满足宽度增长条件时，则需要在可插入分支的搜索空间中选取一个分支并插入该层。本研究考察了三种不同的分支选取策略：不放回随机采样、有放回随机采样和不放回贪心采样。

不放回贪心采样采取的是“最大差异化”原则：在每次选择时优先考虑与现有分支功能差异最大的分支，从而增强网络在多样性特征提取上的能力。首先，我们基于功能相似性将所有可选分支进行排序，形成一个有序的待选集合。其次，依据差异最大化原则从待选集合中选出最优分支加入模型，并从待选集合中移除该分支。这一过程持续进行，直到待选集合为空。以图 3-5 为例，基于不放回贪心采样的一种分支选取顺序为：[单独归一化层，3×3 卷积层，5×3 卷积层，1×3 卷积层，5×5 池化层，3×5 卷积层，3×1 卷积层，1×1 卷积层]。不放回贪心采样是有上限的宽度增长方法，即单层最宽将涵盖完整的可插入分支搜索空间。

不放回随机采样和有放回随机采样则采取了“随机性”原则，其中不放回的随机采样会达到“完整涵盖搜索空间”的上限，而有放回随机采样则没有上限。表 3-7 展示了在控制其他影响因素为最优的情况下，不同待插入分支选取方式对模型自增长性能的影响。

表 3-7 宽度增长中待插入分支选取方式对模型自增长性能的影响

| 分支选取方式        | 不放回随机采样   | 有放回随机采样 | 不放回贪心采样 |
|---------------|-----------|---------|---------|
| 增长后架构 (Arch)  | VGG-2-2-2 |         |         |
| 参数量 (#Params) | 706.35K   |         |         |
| 计算量 (FLOPs)   | 161.38M   |         |         |
| 模型性能 (acc%)   | 91.93     | 92.42   | 92.11   |

实验结果表明，相比于不放回随机采样的 91.93%和不放回贪心采样的 92.11%，有放回随机采样在宽度增长策略中显示出了更优的模型性能，达到了 92.42%的准确率。这一现象主要归因于有放回随机采样策略在分支选取时

是没有上限的，其能够多次选择同一分支，从而理论上提供了更高的灵活性和探索空间。在不放回的采样方法中，贪心采样比随机采样取得了更高的性能，证明了贪心采样方法在提取多样性特征方面的有效性，可以将其作为一种训练高效的分支选取方式。

### （3）模型深度增长和宽度增长的协调匹配

在深度和宽度增长策略的研究的基础上，模型结构优化的下一个阶段是探索深度与宽度增长之间的最佳匹配关系。正确平衡深度和宽度的增长比例对于提升模型的整体性能、增强特征提取能力以及优化模型推理效率都至关重要。本研究分析了不同的增长比例策略，如“每两次深度增长后进行一次宽度增长”、“每次深度增长后进行两次宽度增长”等，来探索哪种配比规则最有潜力提升模型的性能。表 3-8 展示了控制其他影响因素为最优后的相关实验结果。

**表 3-8 模型深度增长和宽度增长的协调匹配对性能的影响**

| 增长匹配方式        | 宽 3 深 1   | 宽 2 深 1 | 宽 1 深 1 | 宽 1 深 2 | 宽 1 深 3 |
|---------------|-----------|---------|---------|---------|---------|
| 增长后架构 (Arch)  | VGG-4-4-4 |         |         |         |         |
| 参数量 (#Params) | 1.24M     |         |         |         |         |
| 计算量 (FLOPs)   | 299.01M   |         |         |         |         |
| 模型性能 (acc%)   | 93.4      | 93.91   | 93.85   | 94.36   | 93.78   |

实验结果表明，不同的宽度与深度增长匹配方式对模型性能有所影响。在这些策略中，每两次深度增长后进行一次宽度增长的匹配方式以 94.36% 的准确率表现最佳，明显优于其他配比。这表明适度偏向深度增长相比于宽度增长，能更有效地提升模型的性能，这我们的经验是一致的：设计更深层的神经网络有助于性能的提升。因此，本研究推荐在追求高性能模型结构设计时，考虑适当的深度优先增长策略，以确保模型能够在保持计算效率的同时，实现最佳性能表现。

### 3.3.4 初始化策略与优化器调整

随着模型的自增长，新加入的网络模块的初始化策略以及对应优化器的调整成为了确保模型性能稳步提升的关键环节。本小节将探讨如何为新增模块选取最佳的初始化方式以快速实现模型的收敛，以及如何调整或更新优化器以适应网络结构的改变，确保新结构能够快速集成原有模型而不损害学习的效率，图 3-6 描述了新增的模型模块所带来的新参数初始化和优化器调整需求。

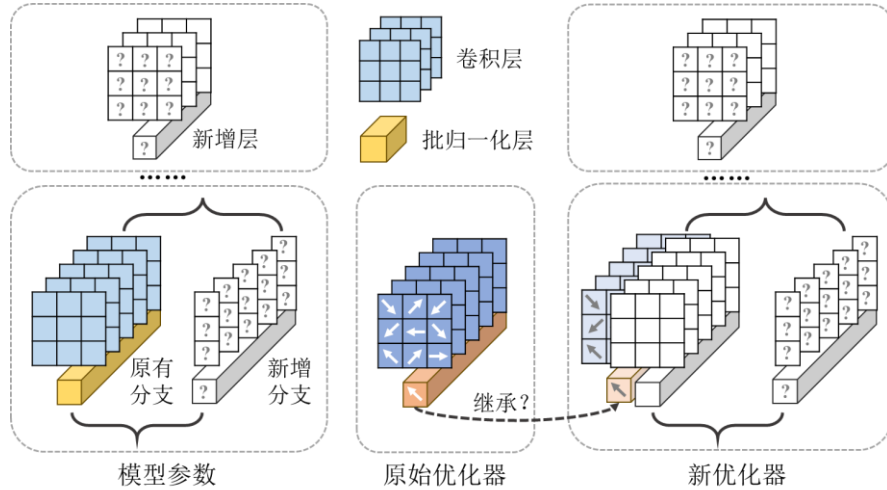


图 3-6 模型自增长中的新参数初始化和优化器调整需求

### (1) 新增模块的参数初始化策略

本研究考虑了多种初始化方法来解决新增模块的参数初始化问题，包括全零初始化、继承式初始化、均匀分布初始化、高斯分布初始化、高斯局部拟合初始化、高斯全局拟合初始化，以及一种基于局部学习初始化方法。

继承式初始化保持新增模块初始化方式与原始模型中现有模块的初始化一致（例如采用 Kaiming 初始化方法来初始化卷积的权重，使用 0 值初始化偏置等），旨在维持模型初始化的一致性。基于高斯分布的局部拟合初始化和全局拟合初始化，通过分析既有模块（局部：新增模块的既有邻居模块；全局：全部既有模块）的参数分布统计量来引导新模块的参数设置。局部学习初始化策略通过先使用较低的学习率（ $1e-3$ ）对新模块单独优化，再整合入完整模型，尝试最大化新模块的潜力，进而优化整个网络的性能。

表 3-9 新增参数初始化方法对自增长性能的影响

| 初始化方式     | 配置                   | 增长后架构<br>(Arch) | 参数量<br>(#Params) | 计算量<br>(FLOPs) | 性能<br>(acc%) |
|-----------|----------------------|-----------------|------------------|----------------|--------------|
| 全零初始化     | -                    |                 |                  |                | 40.16        |
| 均匀分布初始化   | $[-0.01, 0.01]$      |                 |                  |                | 91.00        |
| 均匀分布初始化   | $[-0.1, 0.1]$        |                 |                  |                | 91.56        |
| 均匀分布初始化   | $[-0.3, 0.3]$        |                 |                  |                | 91.80        |
| 高斯分布初始化   | $\mu=0; \sigma=0.01$ |                 |                  |                | 91.56        |
| 高斯分布初始化   | $\mu=0; \sigma=0.1$  | VGG-2-2-2       | 706.35K          | 161.38M        | 91.34        |
| 高斯分布初始化   | $\mu=0; \sigma=0.3$  |                 |                  |                | 91.28        |
| 高斯局部拟合初始化 | -                    |                 |                  |                | 91.27        |
| 高斯全局拟合初始化 | -                    |                 |                  |                | 91.71        |
| 局部学习初始化   | Adam                 |                 |                  |                | 90.68        |
| 继承式初始化    | -                    |                 |                  |                | 92.42        |

表 3-9 展示了在控制其他影响因素为最优的情况下，不同初始化方法对性能的影响。实验结果表明，继承式初始化以 92.42% 的最高准确率表现出色，证明了与原始模型保持初始化一致性的重要性。相较而言，全零初始化的性能大幅下降至 40.16%，暴露了全零初始化对模型训练的不利影响，因为这种初始化方法破坏了神经网络权重的非对称性，导致在训练过程中新增神经元在反向传播时更新相同，从而丧失了神经网络的多样性和表达能力。均匀分布初始化和高斯分布初始化在不同配置下的性能表现各异，显示了超参数范围对模型训练结果的影响较为敏感，其中均匀分布初始化在 $[-0.3, 0.3]$ 区间取值采样下达到了 91.8% 的准确率，略优于高斯分布初始化。高斯局部拟合初始化和高斯全局拟合初始化的表现略低，分别为 90.68% 和 91.27%，可能因为过分依赖现有模块的参数分布而限制了新模块的潜力。局部学习初始化通过单独优化新加入模块，实现了 91.71% 的较高准确率，显示出针对性优化新模块的有效性。综上所述，继承式初始化因其能够与原始模型保持高度的一致性，在所有探索的初始化方法中表现最优，充分证实了这种策略在模型自增长过程中对于保持训练稳定性和促进性能提升的重要作用。

## （2） 优化器调整策略

随着模型结构的自增长引入了新的参数，优化器的适应性和状态管理同样成为了模型训练效率和最终性能优化的关键因素。在构建新的优化器时，面临的主要问题是是否需要从旧优化器中继承状态信息。优化器状态通常包括梯度的一阶和二阶动量信息、学习率调整策略等，这些状态对于优化过程的稳定性和收敛速度可能至关重要。本研究旨在通过实验验证，在为增长后的模型创建新优化器的同时，继承旧优化器状态对模型训练的具体影响，以决定最佳的优化器调整策略。表 3-10 比较了在控制其他影响因素为最优的情况下，全新无状态优化器与继承旧状态优化器在模型性能提升和训练效率上的差异。

**表 3-10 优化器调整策略对模型自增长性能的影响**

| 优化器调整策略       | 全新优化器     | 继承式优化器 |
|---------------|-----------|--------|
| 增长后架构 (Arch)  | VGG-2-2-2 |        |
| 参数量 (#Params) | 706.35K   |        |
| 计算量 (FLOPs)   | 161.38M   |        |
| 模型性能 (acc%)   | 92.21     | 92.42  |

实验结果表明，采用继承式优化器的策略在模型性能上达到了 92.42% 的准确率，相较于全新优化器策略下的 92.21%，有着明显的优势。这一结果指出，优化器的状态继承对于保持模型训练的连续性和加速模型收敛具有重要

作用。全新优化器虽然提供了从零开始的纯净状态，但缺乏对历史训练过程的记忆，可能需要更长的时间来调整和优化新加入参数的权重。相反，继承式优化器通过保留旧优化器中的动量等状态信息，能够更快地适应模型结构的变化，从而促进了更高效的参数更新和更快的性能提升。因此，本研究推荐在进行模型结构调整时优先考虑采用继承式优化器策略，以充分利用历史训练积累的优化状态，实现模型性能的有效提升。

### 3.4 Transformer 模型的自增长方案

#### 3.4.1 Transformer 模型的自增长流程

Transformer 因其强大的全区域建模能力成为了当前研究的热点，在自然语言处理和计算机视觉领域都展现了显著的优势。鉴于此，将模型自增长策略扩展应用到 Transformer 上，不仅是技术发展的必然趋势，也对推动 Transformer 的性能优化具有重要意义。

对 Transformer 进行自增长需要考虑其架构特性：Transformer 作为一个严格各向同性的架构，其层间的宽度一致性对于维持模型的性能和推理效率具有决定性影响，非各向同性的设计可能会由于剧烈变化的层宽度而导致计算效率下降<sup>[94]</sup>。此外 Transformer 依赖全连接层的全感受野进行特征提取，这种机制并不支持基于结构重参数化的模拟宽度增长策略，而是必须通过实质性的宽度扩展来实现增长。

基于对 Transformer 的各向同性特性的讨论，本文进一步探索将自增长框架应用于具体的 Transformer 模型，包括 Vision Transformer<sup>[22]</sup> 和 BERT<sup>[24]</sup> 两种主流架构。两种模型都属于编码器架构，其采用的 Transformer 主干结构是一致的，主要区别在于其对数据的输入输出处理上。对于 Vision Transformer，它通过将输入图像划分为多个小块（Patch），并将这些小块线性嵌入到一个高维空间中，加上一个待分类标记，共同构成模型 Transformer 主干的输入序列。而 BERT 则主要以字符数据作为输入，通过对输入文本序列进行令牌化，并采用位置编码来保留序列中的顺序信息。同时，BERT 通过使用特殊标记来统一不同序列的差异，如“[CLS]”标记用于为分类预留预测位置、“[MASK]”标记用于处理掩码语言建模任务、全零标记用于处理输入句子长度不一致情况下的句式补全。

在明确了两类初始模型的架构之后，本研究正式进入对 Transformer 自增长方案探索的阶段。

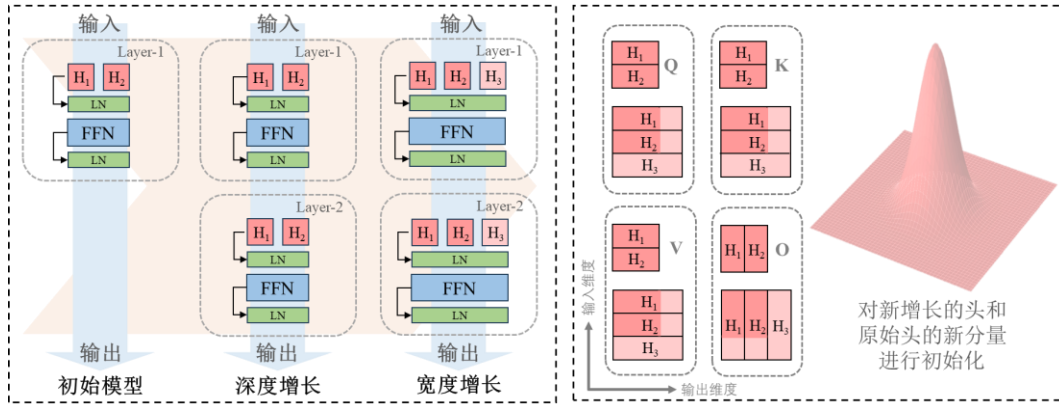


图 3-7 Transformer 的自增长流程

图 3-7 从整体上展示了本研究提出的 Transformer 自增长流程。本研究以一个具有单层双头的 Transformer 作为初始模型，探索模型在自增长框架下通过若干次深度与宽度增长的优化过程。考虑到 Transformer 固有的各向同性特征，其与卷积神经网络的主要区别在于深度和宽度将采取不同的增长策略。

理论上允许在 Transformer 的任意位置进行深度增长，继承前文证明有效的贪心式深度增长策略，本研究采用层后归一中的缩放因子作为衡量层重要性的指标，进而选取表现最重要的层作为新层插入的优先位置。此外，我们还考虑了首层、尾层，以及随机位置作为候选的位置。为了验证贪心选择策略的有效性，本研究针对 CIFAR-10 数据集上的 Vision Transformer 展开实验。实验保持了严格的变量控制，包括将模型增长至 4 层 6 注意力头、采用基于注意力头个数的宽度增长策略、每次增长后训练 5 轮、采用高斯分布初始化新参数并创建全新优化器，以及在进行两次宽度增长后执行一次深度增长的增长比例策略。Transformer 深度增长策略对模型性能的影响如表 3-11 所示。

表 3-11 Transformer 深度增长策略对模型性能的影响

| 深度增长策略        | 首层增长                | 尾层增长  | 随机增长  | 贪心增长         |
|---------------|---------------------|-------|-------|--------------|
| 增长后架构 (Arch)  | ViT-Depth4-Width384 |       |       |              |
| 参数量 (#Params) | 7.10M               |       |       |              |
| 计算量 (FLOPs)   | 1.83G               |       |       |              |
| 模型性能 (acc%)   | 75.64               | 73.92 | 70.48 | <b>77.12</b> |

实验结果表明，采用贪心增长策略的模型以 77.12% 的准确率，显著领先其他策略，体现了该策略在性能优化上的优势。贪心增长策略因此被证明能够基于层重要性的精准评估，有效识别并针对性地增强模型结构中的关键层。相比之下，首层增长、尾层增长以及随机增长策略因缺乏对模型关键部分的精确感知，其性能提升未能达到贪心增长的效果。这进一步证明了，在模型自增长过程中，通过贪心选择关键层进行增长是提升模型性能的有效途径。

对 Transformer 模型进行宽度增长时,可以考虑增长其注意力头的个数或者注意力头的维度。其中,增加注意力头的个数时,每个头的正交方向会出现维度增加,并有一个新的注意力头参与运算,其单个头本身不会出现维度增加的情况;增加注意力头的维度时,每个头的正交方向及每个头本身都会出现维度增加,而头的个数保持不变。为了验证这两种宽度增长方式对模型性能的影响,本研究沿用贪心深度增长策略,并保持其他变量固定进行实验,表 3-12 展示了 Transformer 宽度增长方式对模型性能的影响。

表 3-12 Transformer 宽度增长方式对模型性能的影响

| 宽度增长方式        | 增长注意力头的维度           | 增长注意力头的个数 |
|---------------|---------------------|-----------|
| 增长后架构 (Arch)  | ViT-Depth4-Width384 |           |
| 参数量 (#Params) | 7.10M               |           |
| 计算量 (FLOPs)   | 1.83G               |           |
| 模型性能 (acc%)   | 74.33               | 77.12     |

实验结果显示,在 Transformer 模型的宽度增长实验中,通过增加注意力头的个数所实现的模型性能达到了 77.12%,明显优于仅增加注意力头维度的 74.33%。这一结果反映了增加注意力头数量能够有效增强模型的功能丰富性,新增的注意力头可以捕捉到不同的模式,从而提升了模型对输入数据的理解能力。相比之下,仅增加单个头的维度虽然可以加强头内部的表示能力,但并不能像增加头的数量那样,引入更多角度的信息处理能力。因此,增加注意力头的个数在提升 Transformer 模型的整体性能方面展现出更显著的优势。

3.4.2 Transformer 模型的自增长策略寻优

在明确了 Transformer 的核心自增长流程的基础上,本研究将进一步深入探讨 Transformer 细致化的自增长策略,包括模型单次增长后的训练轮次数,以确保模型能够充分适应新的结构变化;深度增长与宽度增长的合理配比,旨在找到最优的结构调整比例以平衡模型的性能和推理效率;新增参数的恰当初始化方法,以促进模型训练的稳定性 and 效率;以及优化器的适应性调整,确保模型增长后的训练过程高效稳定。

表 3-13 Transformer 模型单次增长后训练轮次数对性能的影响

| 增长后训练轮次       | 1                   | 3     | 5     | 7     | 9     | 15    |
|---------------|---------------------|-------|-------|-------|-------|-------|
| 增长后架构 (Arch)  | ViT-Depth4-Width384 |       |       |       |       |       |
| 参数量 (#Params) | 7.10M               |       |       |       |       |       |
| 计算量 (FLOPs)   | 1.83G               |       |       |       |       |       |
| 模型性能 (acc%)   | 73.88               | 76.50 | 77.12 | 76.34 | 75.84 | 73.32 |



表 3-13 展示了其他变量固定为最优的条件下，Transformer 单次增长后训练轮次数的不同对性能的影响。实验结果表明，当增长后训练轮次设置为 5 轮时，模型达到了最高的准确率 77.12%，相较于卷积神经网络在每 3 轮增长一次的策略，Transformer 模型由于缺乏具体的结构先验且收敛速度较慢，因此在每次结构增长后需要更多的训练轮次以充分适应新的模型结构并实现性能提升。随着训练轮次的增加到 15 轮，模型性能有所下降，反映了过久的训练可能导致过拟合或其他优化问题。

**表 3-14 Transformer 模型深度增长与宽度增长的配比对性能的影响**

| 增长后训练轮次       | 宽 3 深 1              | 宽 2 深 1 | 宽 1 深 1 | 宽 1 深 2 | 宽 1 深 3 |
|---------------|----------------------|---------|---------|---------|---------|
| 增长后架构 (Arch)  | ViT-Depth10-Width512 |         |         |         |         |
| 参数量 (#Params) | 31.52M               |         |         |         |         |
| 计算量 (FLOPs)   | 8.11G                |         |         |         |         |
| 模型性能 (acc%)   | 76.32                | 77.49   | 76.88   | 76.96   | 74.66   |

表 3-14 展示了其他变量固定为最优的条件下，Transformer 深度增长与宽度增长的配比对性能的影响。实验结果表明，进行两次宽度增长后进行一次深度增长的配比方式以 77.49% 的准确率实现了最优性能，表现出在特定情况下宽度增长相对于深度增长的优势。在 Transformer 模型自增长过程中，找到合适的深宽增长配比对于实现最佳模型性能至关重要，“宽 2 深 1”的策略提供了一种在提升模型能力的同时保持结构平衡的有效路径。

**表 3-15 Transformer 模型新增参数初始化方法对自增长性能的影响**

| 初始化方式     | 配置                   | 增长后架构<br>(Arch)             | 参数量<br>(#Params) | 计算量<br>(FLOPs) | 性能<br>(acc%) |
|-----------|----------------------|-----------------------------|------------------|----------------|--------------|
| 全零初始化     | -                    | ViT-<br>Depth4-<br>Width384 | 7.10M            | 1.83G          | 65.91        |
| 均匀分布初始化   | [-0.01, 0.01]        |                             |                  |                | 67.87        |
| 均匀分布初始化   | [-0.1, 0.1]          |                             |                  |                | 71.53        |
| 均匀分布初始化   | [-0.3, 0.3]          |                             |                  |                | 75.40        |
| 高斯分布初始化   | $\mu=0; \sigma=0.01$ |                             |                  |                | 68.05        |
| 高斯分布初始化   | $\mu=0; \sigma=0.1$  |                             |                  |                | 77.12        |
| 高斯分布初始化   | $\mu=0; \sigma=0.3$  |                             |                  |                | 75.30        |
| 高斯局部拟合初始化 | 左局部                  |                             |                  |                | 68.34        |
| 高斯局部拟合初始化 | 右局部                  |                             |                  |                | 69.43        |
| 高斯局部拟合初始化 | 中心局部                 |                             |                  |                | 69.45        |
| 高斯全局拟合初始化 | -                    |                             |                  |                | 76.64        |
| 局部学习初始化   | Adam                 |                             |                  |                | 77.12        |
| 继承式初始化    | -                    |                             |                  |                | 76.27        |

表 3-15 展示了其他变量固定为最优的条件下，Transformer 自增长后不同新增参数初始化方法对性能的影响。实验结果表明，局部学习初始化和高

斯分布初始化（均值为 0，方差为 0.1）以相同的最高准确率 77.12%表现最佳，说明这两种初始化方法能有效促进模型对新参数的适应，进而提升性能。与之相比，全零初始化的效果最差，仅达到 65.91%的准确率。均匀分布初始化和高斯全局拟合初始化表现出相对较好的性能。而高斯局部拟合初始化的各种配置相比较，则性能提升有限，证明局部特征可能不足以指导全局参数的有效初始化。根据简单性原则，本研究选取了成本更低、更快捷的高斯分布初始化作为理想的 Transformer 模型新增参数初始化方法。

和卷积神经网络不同，Transformer 在宽度进行增长后，优化器的形状会相较于原始优化器增加，因此在继承原始优化器的状态之后，还需要使用一定方式给优化器新增位置的状态进行初始化。

**表 3-16 优化器调整对 Transformer 模型自增长性能的影响**

| 优化器调整方式    | 配置                   | 增长后架构<br>(Arch)             | 参数量<br>(#Params) | 计算量<br>(FLOPs) | 性能<br>(acc%) |
|------------|----------------------|-----------------------------|------------------|----------------|--------------|
| 全新优化器      | -                    |                             |                  |                | 77.12        |
| 继承后状态全零初始化 | -                    |                             |                  |                | 77.28        |
| 继承后状态均匀初始化 | [-0.01, 0.01]        | ViT-<br>Depth4-<br>Width384 | 7.10M            | 1.83G          | 76.41        |
| 继承后状态均匀初始化 | [-0.1, 0.1]          |                             |                  |                | 76.86        |
| 继承后状态均匀初始化 | [-0.3, 0.3]          |                             |                  |                | 73.10        |
| 继承后状态高斯初始化 | $\mu=0; \sigma=0.01$ |                             |                  |                | 77.18        |
| 继承后状态高斯初始化 | $\mu=0; \sigma=0.1$  |                             |                  |                | 76.96        |
| 继承后状态高斯初始化 | $\mu=0; \sigma=0.3$  |                             |                  |                | 75.87        |


表 3-16 展示了其他变量固定为最优的条件下，不同优化器调整方式对 Transformer 模型自增长后性能的影响，结果显示，采用继承后状态全零初始化的优化器调整策略，模型性能可以达到最高的 77.28%。因此，对优化器进行调整时，有必要继承原始的优化器状态，但不适合初始化新增优化器状态。

## 3.5 实验验证与案例分析

### 3.5.1 数据集介绍与实验配置

为了验证本研究所提的模型自增长框架能否训练出性能强、效率高、有良好的数据集适应性神经网络模型，本研究继续展开对卷积神经网络模型（具体包括 VGG、ResNet、ResNet-Bottleneck 和 MobileNetV3）和 Transformer 模型（具体包括 Vision Transformer 和 BERT）在各种数据集上的性能和效率的验证分析，对数据集的介绍如表 3-17 所示。

表 3-17 实验涉及数据集介绍

| 数据集  | 数据集描述  |
|--|--|
|    | <p>CIFAR10<sup>[91]</sup> 包含 60,000 张 <math>32^2</math> 像素的彩色三通道图像，分为 50,000 张训练图像和 10,000 张测试图像。这些图像分布在 10 个类别中，每个类别有 6,000 张图像。CIFAR-10 主要用于低分辨率视觉任务，不涉及细粒度识别，数据量适中。</p> |
|    | <p>CIFAR100<sup>[91]</sup> CIFAR10 的细粒度版本，这些图像分布在 100 个类别中，每个类别有 600 张图像。图像分辨率较低，数据量较小。</p>  |
|    | <p>SVHN<sup>[95]</sup> 用于 10 类数字识别的彩色门牌号图像。分辨率为 <math>32^2</math> 像素，包含 73,257 张训练图像，以及 26,032 张测试图像。这是一个低分辨率的多通道图像数据集，不涉及细粒度识别，数据量适中。</p>                                 |
|   | <p>MNIST<sup>[96]</sup> 经典的手写数字识别任务，包含 <math>28^2</math> 像素的单通道（灰度）图像。训练集包含 60,000 张图像，测试集包含 10,000 张图像。这是一个低分辨率的非细粒度任务数据集，数据量适中。</p>                                      |
|  | <p>ImageWoof<sup>[97]</sup> 细粒度狗品种识别，包含 <math>224^2</math> 大小的彩色三通道 9,025 张训练和 3,929 张验证图像，共 10 个类别，细粒度识别，数据量少。</p>  |
|  | <p>ImagenetTE<sup>[97]</sup> 高分辨率易分类图像识别，包含 <math>224 \times 224</math> 大小的彩色三通道 9,649 张训练图像和 3,925 张验证图像，共 10 个类别，不涉及细粒度识别，数据量少。</p>                                      |
|  | <p>Tiny-Imagenet<sup>[97]</sup> Tiny-Imagenet 包含 200 个类别，每个类别有 500 张训练图像、50 张验证图像和 50 张测试图像。图像是分辨率为 <math>64 \times 64</math> 的彩色三通道图像，是一个高挑战性的紧凑数据集，不涉及细粒度识别，数据量少。</p>    |
|  | <p>Mini-Imagenet<sup>[97]</sup> Mini-Imagenet 包含 100 个类别的彩色三通道 <math>224 \times 224</math> 大小的图像。共 48,000 个训练样本和 12,000 个验证样本。不涉及细粒度识别，数据量少。</p>                           |

|           |  |            |
|-----------|--|------------|
| IMDB [98] | I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967. I also heard that at first it was seized by U.S. customs if it ever tried to enter this country, therefore being a fan of films considere.. | 0 neg      |
|           | "I Am Curious: Yellow" is a risible and pretentious steaming pile. It doesn't matter what one's political views are because this film can hardly be taken seriously on any level. As for the claim that frontal male nudity is an automatic NC-17, that isn't true. I've seen..    | 0 neg      |
| SST2 [99] | If only to avoid making this type of film in the future. This film is interesting as an experiment but tells no cogent story.<br />One might feel virtuous for sitting thru it because it touches on so many IMPORTANT issues but it does so without any discernable..             | 0 neg      |
|           | contains no wit , only labored gags  | 0 negative |
|           | that loves its characters and communicates something rather beautiful about human nature   | 1 positive |
|           | remains utterly satisfied to remain the same throughout  | 0 negative |

电影评论情感类任务，包含 25,000 条训练文本和 25,000 条验证文本，分为正面和负面两个类别，数据量适中。

综合情感分析任务，包含超过 673,000 条训练文本和 872 条验证文本，分为正面和负面两个类别，数据量大。

根据前文对自增长策略的探索寻优，本部分实验将为神经网络的自增长设置如表 3-18 所示的配置。

表 3-18 模型自增长配置方案

| 配置项         | 配置内容   |
|-------------|--|
| 全局配置        | 自增长初始学习率 0.1；采用基于 Cosine 的学习率调整机制；SGD 优化器；实验框架为 Pytorch <sup>[100]</sup> ；效率测试平台为 NVIDIA A100 PCIE 40GB GPU；固定规模模型采取 Huggingface/Timm 提供的架构；采用 Adam 优化器；5e-3 初始学习率；采用基于 Cosine 的学习率调整机制；采用强度为 5e-4 的二阶权重正则化。 |
| 特殊配置        | 由于 MNIST 数据集的简单性，将手动控制每个计算区块增长至最多一层退出；其他数据集通过自动增长结束方案控制。   |
| 卷积神经网络      | 总训练轮次 100；新增参数初始化方法为继承式初始化；模块增长后优化器调整方法为继承式调整；新增模块后单次训练为 3 轮；增长位置采用贪心选择；采取随机有放回的方式来采样宽度增长中的可重参数化分支；采取增长一次宽度后增长两次深度的配比方案。   |
| Transformer | 总训练轮次 200；新增参数初始化方法为高斯初始化法；模块增长后优化器调整方法为继承式调整并全零初始化新增状态；新增模块后单次训练为 5 轮；增长位置采用贪心选择；采取增长两次宽度后增长一次深度的配比方案。  |

### 3.5.2 卷积神经网络模型的数据集适应性评估

为了评估自增长卷积神经网络在不同数据集上的适应性，本研究选择了一系列多样化的数据集，包括 CIFAR-10、CIFAR-100、SVHN、MNIST、ImagenetTE、ImageWoof、Tiny-ImageNet 和 Mini-ImageNet，涵盖了从简单到复杂、从低分辨率到高分辨率、从非细粒度识别到细粒度识别、从单通道到多通道的各种图像识别数据集。本研究选取的基准（Baseline）是一系列固定规模的经典模型，包括 ResNet（含普通架构和 Bottleneck 瓶颈架构）、VGG 以及 MobileNetV3。

表 3-19 CIFAR-10 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 558.39M | 3.83     | 87.76    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 1.16G   | 6.92     | 87.60    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 1.31G   | 9.25     | 86.12    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 2.53G   | 17.87    | 86.02    |
| 自增长 ResNet            | 1-2-2               | 1.17M   | 222.99M | 1.90     | 92.22    |
| 自增长 ResNet            | 1-2-2-1             | 2.65M   | 246.66M | 2.11     | 93.14    |
| 自增长 Bottleneck-ResNet | 2-2-2               | 235.07K | 41.09M  | 2.18     | 90.08    |
| 自增长 Bottleneck-ResNet | 2-2-2-2             | 673.21K | 48.14M  | 2.89     | 90.94    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 440.32M | 3.14     | 89.17    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 601.25M | 3.26     | 90.48    |
| 自增长 VGG               | 1-2-2               | 669.29K | 123.37M | 1.08     | 90.72    |
| 自增长 VGG               | 1-1-2-2             | 2.06M   | 125.72M | 1.37     | 91.44    |
| MobileNetV3-small     | 2-2-3-2-2           | 2.84M   | 8.98M   | 7.64     | 65.84    |
| MobileNetV3-large     | 2-2-3-4-2-2         | 5.74M   | 17.78M  | 16.98    | 70.32    |
| 自增长 MobileNetV3       | 2-3-3               | 811.40K | 20.37M  | 3.71     | 89.32    |
| 自增长 MobileNetV3       | 2-2-3-2             | 670.52K | 7.24M   | 4.40     | 88.31    |

表 3-19 展示了在 CIFAR-10 数据集上，自增长卷积神经网络与固定规模网络性能的对比。自增长模型在整体效率超高的情况下，实现了巨大的准确性提升，这种提升在性能偏差的 MobileNetV3 系列模型上更为明显，证明了自增长策略在模型优化和数据集适应性提升上的优势。

表 3-20 CIFAR-100 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 558.39M | 3.83     | 55.80    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 1.16G   | 6.92     | 55.40    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 1.31G   | 9.25     | 53.16    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 2.53G   | 17.87    | 53.40    |
| 自增长 ResNet            | 2-2-2               | 1.24M   | 299.01M | 1.88     | 70.64    |
| 自增长 ResNet            | 1-2-2-2             | 3.83M   | 265.57M | 2.43     | 72.90    |
| 自增长 Bottleneck-ResNet | 2-3-3               | 262.99K | 44.91M  | 2.54     | 67.52    |
| 自增长 Bottleneck-ResNet | 2-2-3-2             | 691.03K | 49.30M  | 3.39     | 69.78    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 440.32M | 3.14     | 59.27    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 601.25M | 3.26     | 59.32    |
| 自增长 VGG               | 2-3-3               | 937.42K | 192.19M | 1.40     | 67.84    |
| 自增长 VGG               | 2-2-3-2             | 2.33M   | 194.53M | 1.79     | 69.46    |

|                   |             |         |        |       |       |
|-------------------|-------------|---------|--------|-------|-------|
| MobileNetV3-small | 2-2-3-2-2   | 2.84M   | 8.98M  | 7.64  | 34.04 |
| MobileNetV3-large | 2-2-3-4-2-2 | 5.74M   | 17.78M | 16.98 | 33.80 |
| 自增长 MobileNetV3   | 3-4-3       | 903.18K | 25.80M | 4.97  | 66.54 |
| 自增长 MobileNetV3   | 3-3-3-3     | 855.29K | 9.33M  | 5.73  | 63.82 |

表 3-20 展示了在 CIFAR-100 数据集上，自增长卷积神经网络与固定架构模型性能的比较结果，自增长方法在数据集适应性方面再次展现了其卓越能力，同时实现了性能与效率之间的平衡。尤其是对于难以胜任 CIFAR-100 数据集的 MobileNetV3 系列模型，自增长模型将其性能提升了约一倍（从 33.04% 提升至 66.54%）。CIFAR-100 是 CIFAR-10 的细粒度版本，相比之下，总类别数更多，单个类别的数据量更少，整体难度更高，因此自增长模型的最终规模会超过 CIFAR-10 数据集，这体现了自增长方法能够具体数据集具体分析的优点。即便如此，自增长模型相比于固定规模的模型，依然有着更高的推理效率，同时体现在参数量、计算量和推理延迟上。

表 3-21 SVHN 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 558.39M | 3.83     | 95.07    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 1.16G   | 6.92     | 95.41    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 1.31G   | 9.25     | 96.13    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 2.53G   | 17.87    | 96.57    |
| 自增长 ResNet            | 1-1-1               | 706.35K | 161.38M | 1.17     | 96.01    |
| 自增长 ResNet            | 1-2-1-1             | 2.35M   | 227.72M | 1.82     | 96.55    |
| 自增长 Bottleneck-ResNet | 1-2-1               | 212.68K | 35.08M  | 1.52     | 95.56    |
| 自增长 Bottleneck-ResNet | 1-1-2-1             | 588.08K | 39.50M  | 1.99     | 96.21    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 440.32M | 3.14     | 95.76    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 601.25M | 3.26     | 95.91    |
| 自增长 VGG               | 1-2-1               | 521.45K | 113.90M | 1.03     | 95.87    |
| 自增长 VGG               | 1-1-2-1             | 1.47M   | 116.26M | 1.21     | 95.80    |
| MobileNetV3-small     | 2-2-3-2-2           | 2.84M   | 8.98M   | 7.64     | 92.26    |
| MobileNetV3-large     | 2-2-3-4-2-2         | 5.74M   | 17.78M  | 16.98    | 93.76    |
| 自增长 MobileNetV3       | 2-2-2               | 578.57K | 16.19M  | 2.86     | 95.74    |
| 自增长 MobileNetV3       | 2-2-2-2             | 597.05K | 6.54M   | 4.19     | 95.92    |

表 3-21 展示了在 SVHN 数据集上，自增长卷积神经网络与固定架构模型性能的比较结果，自增长策略依然体现出强大的数据集适应性，并展示出理想的性能效率平衡：SVHN 数据集相对比较简单，刺激了更轻量化的自增长模型的构建。

表 3-22 MNIST 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.68M  | 36.12M  | 3.59     | 99.80    |
| ResNet-34             | 3-4-6-3             | 21.79M  | 73.97M  | 6.46     | 99.84    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.55M  | 84.76M  | 9.17     | 99.76    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.54M  | 160.94M | 17.24    | 99.76    |
| 自增长 ResNet            | 1-1-1               | 705.19K | 160.20M | 1.24     | 99.76    |
| 自增长 ResNet            | 1-1-1-1             | 2.18M   | 183.88M | 1.55     | 99.84    |
| 自增长 Bottleneck-ResNet | 1-1-1               | 201.43K | 31.25M  | 1.31     | 99.72    |
| 自增长 Bottleneck-ResNet | 1-1-1-1             | 569.11K | 37.15M  | 2.03     | 99.76    |
| VGG-11                | 1-1-2-2-2           | 132.86M | 275.23M | 2.49     | 99.60    |
| VGG-16                | 2-2-3-3-3           | 138.36M | 435.66M | 3.16     | 99.48    |
| 自增长 VGG               | 1-1-1               | 437.07K | 91.39M  | 0.81     | 99.66    |
| 自增长 VGG               | 1-1-1-1             | 1.32M   | 105.61M | 1.10     | 99.78    |
| MobileNetV3-small     | 2-2-3-2-2           | 2.84M   | 8.91M   | 8.58     | 99.64    |
| MobileNetV3-large     | 2-2-3-4-2-2         | 5.74M   | 17.71M  | 16.46    | 99.56    |
| 自增长 MobileNetV3       | 1-1-1               | 326.86K | 9.14M   | 1.64     | 99.62    |
| 自增长 MobileNetV3       | 1-1-1-1             | 338.52K | 3.69M   | 2.20     | 99.64    |

表 3-22 展示了在 MNIST 数字识别数据集上，自增长模型与固定规模模型性能的差异，自增长模型明显以更高的效率，实现了对数据集的高度适配。MNIST 是所有数据集中最简单的，因此即便手动规定每个计算区块只增长一次，也能在该数据集上找到性能和效率的理想平衡。

表 3-23 ImageWoof 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 27.34G  | 3.80     | 85.25    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 57.01G  | 6.81     | 88.14    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 64.27G  | 11.95    | 88.91    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 123.99G | 23.41    | 88.40    |
| 自增长 ResNet            | 3-3-3               | 1.78M   | 21.40G  | 2.78     | 90.06    |
| 自增长 ResNet            | 2-2-3-2             | 4.20M   | 17.67G  | 2.84     | 90.64    |
| 自增长 Bottleneck-ResNet | 4-4-3               | 282.22K | 2.81G   | 3.56     | 88.05    |
| 自增长 Bottleneck-ResNet | 3-3-4-3             | 793.99K | 2.90G   | 4.18     | 88.92    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 15.64G  | 2.68     | 87.99    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 23.53G  | 3.24     | 87.89    |
| 自增长 VGG               | 3-4-3               | 1.06M   | 12.32G  | 1.72     | 89.04    |
| 自增长 VGG               | 3-3-3-3             | 3.04M   | 12.90G  | 2.09     | 89.20    |



|                   |             |         |         |      |       |
|-------------------|-------------|---------|---------|------|-------|
| MobileNetV3-small | 2-2-3-2-2   | 2.84M   | 378.02M | 8.17 | 77.72 |
| MobileNetV3-large | 2-2-3-4-2-2 | 5.74M   | 743.14M | 17.5 | 83.72 |
| 自增长 MobileNetV3   | 4-4-4       | 1.08M   | 1.45G   | 5.91 | 88.86 |
| 自增长 MobileNetV3   | 3-3-4-3     | 928.76K | 472.60M | 6.24 | 89.87 |

表 3-23 展示了在 ImageWoof 数据集上，自增长模型与固定规模模型的性能差异，在数据集适应和性能效率方面，自增长方法都显示出了强大的潜力。ImageWoof 数据集是 ImageNet 的子集，是一个细粒度、高分辨率狗分类任务。由于细粒度带来的识别难度，自增长模型的规模偏高，但依然展现出相比固定规模模型更优的性能和推理效率。除此以外，多次（3 次）下采样设置下的自增长模型相比少量下采样设置下的自增长模型，因此其更加适合高分辨率图像的识别。

表 3-24 ImagenetTE 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 27.34G  | 3.80     | 93.99    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 57.01G  | 6.81     | 94.30    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 64.27G  | 11.95    | 92.77    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 123.99G | 23.41    | 95.72    |
| 自增长 ResNet            | 2-3-2               | 1.41M   | 16.74G  | 2.35     | 93.68    |
| 自增长 ResNet            | 2-2-2-2             | 3.90M   | 16.74G  | 2.70     | 95.47    |
| 自增长 Bottleneck-ResNet | 3-4-3               | 277.66K | 2.57G   | 3.17     | 92.83    |
| 自增长 Bottleneck-ResNet | 2-3-3-2             | 704.14K | 2.55G   | 3.30     | 94.40    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 15.64G  | 2.68     | 94.40    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 23.53G  | 3.24     | 94.71    |
| 自增长 VGG               | 3-3-3               | 974.47K | 11.28G  | 1.54     | 94.05    |
| 自增长 VGG               | 2-3-3-3             | 3.01M   | 11.04G  | 1.90     | 95.42    |
| MobileNetV3-small     | 2-2-3-2-2           | 2.84M   | 378.02M | 8.17     | 91.04    |
| MobileNetV3-large     | 2-2-3-4-2-2         | 5.74M   | 743.14M | 17.50    | 91.96    |
| 自增长 MobileNetV3       | 3-4-3               | 903.18K | 1.25G   | 4.66     | 95.57    |
| 自增长 MobileNetV3       | 3-3-3-3             | 855.29K | 439.85M | 4.97     | 95.98    |

表 3-24 展示了在 ImagenetTE 数据集上，自增长卷积神经网络与固定架构模型性能的比较结果，自增长模式不仅证实了对数据集的适用性，也在性能与效率之间实现了平衡。ImagenetTE 数据集是一个易分类的 ImageNet 子集，不涉及细粒度识别，因此相比 ImageWoof 数据集，自增长框架仅需要为模型装载更少的参数，就能够达到超过固定规模模型的性能，以此实现更为理想的模型性能和推理效率的平衡。

表 3-25 Tiny-Imagenet 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 2.23G   | 3.78     | 49.96    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 4.65G   | 6.86     | 49.56    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 5.25G   | 11.21    | 53.92    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 10.12G  | 22.77    | 52.72    |
| 自增长 ResNet            | 4-5-5               | 8.10M   | 4.40G   | 2.88     | 63.92    |
| 自增长 ResNet            | 3-4-4-4             | 26.67M  | 4.78G   | 3.15     | 66.28    |
| 自增长 Bottleneck-ResNet | 5-5-5               | 886.46K | 440.85M | 3.69     | 56.58    |
| 自增长 Bottleneck-ResNet | 4-4-5-4             | 3.22M   | 550.74M | 4.33     | 62.24    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 1.39G   | 2.81     | 42.28    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 2.03G   | 3.98     | 41.00    |
| 自增长 VGG               | 5-5-5               | 4.30M   | 2.43G   | 1.62     | 60.55    |
| 自增长 VGG               | 4-4-5-4             | 14.79M  | 2.81G   | 2.03     | 61.30    |
| MobileNetV3-small     | 2-2-3-2-2           | 2.84M   | 32.05M  | 7.86     | 39.72    |
| MobileNetV3-large     | 2-2-3-4-2-2         | 5.74M   | 63.12M  | 17.91    | 44.56    |
| 自增长 MobileNetV3       | 5-6-5               | 2.04M   | 177.74M | 6.06     | 55.24    |
| 自增长 MobileNetV3       | 5-5-5-5             | 3.40M   | 103.80M | 7.48     | 59.02    |

表 3-25 展示了在 Tiny-Imagenet 数据集上，自增长卷积神经网络与固定架构模型性能的比较结果，自增长模型同样展示了在性能增强与效率优化之间的理想平衡。Tiny-Imagenet 数据集的难度较高，因此自增长框架找到的模型规模稍高，但在性能上超过了固定规模模型 10%甚至更高。

表 3-26 Mini-Imagenet 卷积神经网络自增长效果

| 模型                    | 架构                  | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------|---------------------|---------|---------|----------|----------|
| ResNet-18             | 2-2-2-2             | 11.69M  | 27.34G  | 3.80     | 80.23    |
| ResNet-34             | 3-4-6-3             | 21.80M  | 57.01G  | 6.81     | 81.93    |
| ResNet-50             | 3-4-6-3 bottleneck  | 25.56M  | 64.27G  | 11.95    | 80.77    |
| ResNet-101            | 3-4-23-3 bottleneck | 44.55M  | 123.99G | 23.41    | 82.17    |
| 自增长 ResNet            | 4-4-4               | 6.60M   | 46.53G  | 2.90     | 82.18    |
| 自增长 ResNet            | 3-3-4-3             | 21.60M  | 51.13G  | 3.13     | 85.93    |
| 自增长 Bottleneck-ResNet | 5-6-5               | 878.58K | 5.63G   | 4.20     | 73.35    |
| 自增长 Bottleneck-ResNet | 4-4-5-4             | 3.16M   | 6.75G   | 4.44     | 81.15    |
| VGG-11                | 1-1-2-2-2           | 140.72M | 15.64G  | 2.68     | 78.83    |
| VGG-16                | 2-2-3-3-3           | 146.22M | 23.53G  | 3.24     | 79.27    |
| 自增长 VGG               | 4-5-5               | 4.24M   | 27.95G  | 1.82     | 82.00    |
| 自增长 VGG               | 4-4-5-4             | 14.74M  | 34.43G  | 2.17     | 84.13    |

|                   |             |       |         |       |       |
|-------------------|-------------|-------|---------|-------|-------|
| MobileNetV3-small | 2-2-3-2-2   | 2.84M | 378.02M | 8.17  | 73.43 |
| MobileNetV3-large | 2-2-3-4-2-2 | 5.74M | 743.14M | 17.50 | 74.03 |
| 自增长 MobileNetV3   | 5-6-5       | 2.03M | 2.17G   | 6.41  | 81.30 |
| 自增长 MobileNetV3   | 4-5-5-4     | 2.95M | 1.17G   | 7.39  | 80.33 |

表 3-26 展示了在中等难度的 Mini-Imagenet 数据集上，自增长卷积神经网络与固定架构模型性能的对比，自增长策略再次展现了其强大的性能、效率和数据集适应性。

经过在 8 个不同数据集上的实验验证，自增长策略显著地证明了其出色的数据集适应性和在性能效率平衡方面的优势。无论数据集的难易，自增长模型均展现出了比传统固定架构模型更高的准确率和更优的计算效率。除此之外，小型模型训练的简单性使得自增长训练不再像训练大规模模型那样依赖复杂的优化器、细致的正则化以及超低的初始学习率（详见表 3-18），反而可以以更简单直接的配置实现更快的训练。

### 3.5.3 Transformer 模型的数据集适应性评估

为了全面评估自增长 Transformer 模型在不同数据集上的适应性与效率，本研究选择了一系列多样化的数据集，包括 CIFAR-10、ImagenetTE、Tiny-ImageNet 和 Mini-ImageNet 用于验证 ViT 的效果，以及 IMDB 和 SST2 用于验证 BERT 的效果，涵盖了计算机视觉和自然语言处理领域下的相关任务。本研究同时选取一系列固定规模的经典模型作为基准，具体包括 ViT（Small 版、Large 版）和 BERT（Small 版、Large 版）。

表 3-27 计算机视觉任务上的 Transformer 模型自增长效果

| 数据集           | 模型                    | 架构      | 参数量    | 计算量    | 推理延迟<br>(ms) | 性能<br>(acc%) |
|---------------|-----------------------|---------|--------|--------|--------------|--------------|
| CIFAR10       | Small-ViT-Patch2-32   | D10W512 | 31.52M | 8.11G  | 7.18         | 65.04        |
|               | Base-ViT-Patch2-32    | D12W768 | 85.05M | 21.88G | 7.86         | 74.60        |
|               | 自增长 ViT-Patch2-32     | D4W384  | 7.10M  | 1.83G  | 1.96         | 77.74        |
| ImagenetTE    | Small-ViT-Patch16-224 | D10W512 | 31.91M | 6.29G  | 6.22         | 64.05        |
|               | Base-ViT-Patch16-224  | D12W768 | 85.63M | 16.88G | 7.65         | 62.02        |
|               | 自增长 ViT-Patch16-224   | D5W448  | 12.42M | 2.45G  | 2.47         | 77.94        |
| Tiny-Imagenet | Small-ViT-Patch4-64   | D10W512 | 31.54M | 8.12G  | 6.58         | 20.94        |
|               | Base-ViT-Patch4-64    | D12W768 | 85.08M | 21.89G | 8.17         | 32.08        |
|               | 自增长 ViT-Patch4-64     | D6W384  | 10.74M | 2.75G  | 2.77         | 35.70        |

|               |                       |         |        |        |      |       |
|---------------|-----------------------|---------|--------|--------|------|-------|
| Mini-Imagenet | Small-ViT-Patch16-224 | D10W512 | 31.91M | 6.29G  | 6.02 | 33.80 |
|               | Base-ViT-Patch16-224  | D12W768 | 85.63M | 16.88G | 8.08 | 34.23 |
|               | 自增长 ViT-Patch16-224   | D6W448  | 14.87M | 2.92G  | 2.86 | 46.12 |

表 3-27 展示了在计算机视觉任务上,不同规模 ViT 模型自增长效果的比较。自增长 ViT 模型在所有数据集中均展现出优越的性能和效率,显著高于同数据集下的 Small 和 Base 版本 ViT 模型。

**表 3-28 自然语言处理任务上的 Transformer 模型自增长效果**

| 数据集  | 模型         | 架构      | 参数量    | 计算量    | 推理延迟(ms) | 性能<br>(acc%) |
|------|------------|---------|--------|--------|----------|--------------|
| IMDB | Small-BERT | D10W512 | 23.77M | 6.30G  | 6.77     | 83.92        |
|      | Base-BERT  | D12W768 | 63.94M | 16.78G | 7.68     | 83.18        |
|      | 自增长 BERT   | D3W384  | 4.03M  | 1.08G  | 1.44     | 85.37        |
| SST2 | Small-BERT | D10W512 | 23.77M | 6.30G  | 6.77     | 82.8         |
|      | Base-BERT  | D12W768 | 63.94M | 16.78G | 7.68     | 83.03        |
|      | 自增长 BERT   | D4W384  | 5.37M  | 1.44G  | 1.83     | 85.77        |

表 3-28 展示了在自然语言处理任务上,不同规模的 BERT 模型自增长效果的比较。在 IMDB 和 SST2 两个数据集中,自增长 BERT 模型均实现了显著的性能提升,分别达到了 85.37%和 85.77%的准确率,超过了 Small-BERT 和 Base-BERT 模型的性能。自增长 BERT 模型在保持较低参数量和计算量的同时,还大幅减少了推理延迟,展现了极高的效率。

通过在视觉和自然语言处理任务上的广泛实验, Vision Transformer 和 BERT 模型的自增长策略均展现了显著的性能提升和高效的计算效率。自增长方法在各自领域内,不仅以较小的参数量和计算成本实现了超越固定架构模型的性能,还大幅降低了推理延迟,证明了其在优化模型结构和提升模型数据集适应性方面的有效性。自增长训练还大大降低了 Transformer 的训练难度: Transformer 的训练难度一部分来自于其庞大的参数规模,难以稳定训练,另一部分来自其无先验假设的特性,导致较高的大数据依赖。自增长训练显然对庞大参数规模引起的训练难度的降低起到了非常关键的作用。

### 3.5.4 与模型剪枝的对比

模型剪枝可以看作是模型自增长的对偶策略,二者各自采取了相反的路径以达成相似的目标:自增长策略通过从较小的模型出发,逐步扩展模型的规模以增强其性能,而模型剪枝则从一个较大的模型开始,通过去除不必要

的参数来提高模型的运行效率。本部分将比较这两种方法在最终模型性能和效率上的差异，并分析优化过程的复杂度。

本研究采用在 CIFAR-10 上训练的 ResNet-50 和 ViT-Base 模型进行剪枝实验，以探索自增长方法和不同剪枝方法效果的对比。对于 ResNet-50，我们采用了三种剪枝方法作为基线：一是基于批归一化层的缩放因子进行的结构化剪枝<sup>[51]</sup>，二是利用分支分组搜索实现的动态结构化剪枝 DSP<sup>[54]</sup>，三是一种基于参数幅度进行的半结构化和非结构化剪枝<sup>[101]</sup>。而在 ViT-Base 的剪枝实验中，我们选取了基于参数幅度和基于激活幅度<sup>[102]</sup>的半结构化及非结构化剪枝方法作为对比基准。

表 3-29 ResNet 剪枝结果与模型自增长的对比

| 方法                 | 稀疏度      | 参数量     | 计算量     | 延迟<br>(ms) | 性能<br>(acc%) | 优化时间   |       |       |       |
|--------------------|----------|---------|---------|------------|--------------|--------|-------|-------|-------|
|                    |          |         |         |            |              | 预训练    | 剪枝    | 微调    | 总体    |
| ResNet-50          | -        | 25.56M  | 1.31G   | 9.25       | 86.12        | 2536s  | 0s    | 0s    | 2536s |
| 批归一化               | 50%结构化   | 12.78M  | 0.66G   | 6.11       | 15.78        | 2536s  | 12s   | 0s    | 2548s |
|                    | 50%结构化   | 12.78M  | 0.66G   | 6.11       | 87.85        | 2536s  | 12s   | 587s  | 3135s |
|                    | 70%结构化   | 17.91M  | 0.92G   | 5.2        | 10.13        | 2536s  | 12s   | 0s    | 2548s |
|                    | 70%结构化   | 17.91M  | 0.92G   | 5.2        | 86.83        | 2536s  | 12s   | 587s  | 3135s |
| DSP                | 50%结构化   | 12.78M  | 0.66G   | 7.37       | 14.67        | 2536s  | 5764s | 0s    | 8300s |
|                    | 50%结构化   | 12.78M  | 0.66G   | 7.37       | 91.79        | 2536s  | 5764s | 1526s | 9826s |
|                    | 70%结构化   | 17.91M  | 0.92G   | 7.19       | 11.95        | 2536s  | 5764s | 0s    | 8300s |
|                    | 70%结构化   | 17.91M  | 0.92G   | 7.19       | 90.41        | 2536s  | 5764s | 1526s | 9826s |
| 参数幅度               | 50%非结构化  | 12.79M  | 1.31G   | 9.25       | 61.39        | 2536s  | 9s    | 0s    | 2545s |
|                    | 50%非结构化  | 12.79M  | 1.31G   | 9.25       | 84.4         | 2536s  | 9s    | 812s  | 3357s |
|                    | 70%非结构化  | 17.91M  | 1.31G   | 9.25       | 21.35        | 2536s  | 9s    | 0s    | 2545s |
|                    | 70%非结构化  | 17.91M  | 1.31G   | 9.25       | 82.47        | 2536s  | 9s    | 812s  | 3357s |
|                    | 2:4 半结构化 | 12.79M  | 0.67G   | 8.04       | 31.52        | 2536s  | 9s    | 0s    | 2545s |
|                    | 2:4 半结构化 | 12.79M  | 0.67G   | 8.04       | 82.97        | 2536s  | 9s    | 812s  | 3357s |
| ResNet-3-3-3       | -        | 1.17M   | 222.99M | 1.9        | 92.22        | 13560s | 0s    | 0s    | 1360s |
| ResNet-2-2-3-2     | -        | 2.65M   | 246.66M | 2.11       | 93.14        | 1649s  | 0s    | 0s    | 1649s |
| Bottleneck-4-4-3   | -        | 235.07K | 41.09M  | 2.18       | 90.08        | 1416s  | 0s    | 0s    | 1416s |
| Bottleneck-3-3-4-3 | -        | 673.21K | 48.14M  | 2.89       | 90.94        | 1723s  | 0s    | 0s    | 1723s |

表 3-29 展示了 ResNet-50 剪枝与模型自增长策略在 CIFAR10 数据集上的对比结果，体现了两种策略在模型性能、计算资源效率及优化时间上的差异。剪枝策略中，虽然 DSP 方法在 50%结构化剪枝后能够达到 91.79%的较高准确率，但其优化时间显著增长至 9826 个 GPU 秒（在 NVIDIA RTX 3090

24GB GPU 上记录），远超原始模型预训练和自增长模型的优化时间。相比之下，自增长模型不仅在参数量和计算量上大幅减少，推理延迟显著降低，而且在较短的优化时间内（例如 1416 秒）便实现了更高的准确率（高达 93.14%），展示了自增长策略在保持甚至提升性能的同时，显著提高了模型的训练效率。

**表 3-30 Vision Transformer 剪枝结果与模型自增长的对比**

| 方法         | 稀疏度      | 参数量    | 计算量   | 延迟<br>(ms) | 性能<br>(acc%) | 优化时间   |    |       |        |
|------------|----------|--------|-------|------------|--------------|--------|----|-------|--------|
|            |          |        |       |            |              | 预训练    | 剪枝 | 微调    | 总体     |
| ViT-Base   | -        | 31.52M | 8.11G | 7.18       | 74.6         | 15921s | 0s | 0s    | 15921s |
| 参数幅度       | 50%非结构化  | 15.76M | 8.11G | 7.18       | 71.37        | 15921s | 1s | 0s    | 15922s |
|            | 50%非结构化  | 15.76M | 8.11G | 7.18       | 74.61        | 15921s | 1s | 1830s | 17752s |
|            | 70%非结构化  | 9.46M  | 8.11G | 7.18       | 66.94        | 15921s | 1s | 0s    | 15922s |
|            | 70%非结构化  | 9.46M  | 8.11G | 7.18       | 74.31        | 15921s | 1s | 1830s | 17752s |
|            | 50%行半结构化 | 15.76M | 8.11G | 7.18       | 42.12        | 15921s | 1s | 0s    | 15922s |
|            | 50%行半结构化 | 15.76M | 8.11G | 7.18       | 72.16        | 15921s | 1s | 1830s | 17752s |
| 激活幅度       | 50%非结构化  | 15.76M | 8.11G | 7.18       | 73.53        | 15921s | 3s | 0s    | 15924s |
|            | 50%非结构化  | 15.76M | 8.11G | 7.18       | 76.71        | 15921s | 3s | 1830s | 21354s |
|            | 70%非结构化  | 9.46M  | 8.11G | 7.18       | 72.82        | 15921s | 3s | 0s    | 15924s |
|            | 70%非结构化  | 9.46M  | 8.11G | 7.18       | 74.46        | 15921s | 3s | 1830s | 21354s |
|            | 50%行半结构化 | 15.76M | 8.11G | 7.18       | 67.34        | 15921s | 3s | 0s    | 15924s |
|            | 50%行半结构化 | 15.76M | 8.11G | 7.18       | 73.97        | 15921s | 3s | 1830s | 21354s |
| ViT-D4W384 | -        | 7.10M  | 1.83G | 1.96       | 77.74        | 11724s | 0s | 0s    | 11724s |

表 3-30 对 ViT 模型的剪枝结果与自增长效果进行了对比，展示了两种策略在性能、计算资源消耗以及优化耗时上的明显差异。对于剪枝来说，虽然通过非结构化剪枝和半结构化剪枝后的 ViT 模型在某些情况下能够维持或略微提升性能，但优化总耗时显著增加，特别是包括了微调阶段后，总优化时间可高达 21354 个 GPU 秒。相反，自增长 ViT 模型以远低于原始模型的参数量和计算量，实现了更高的性能（77.74%），同时大幅减少了推理延迟和优化时间（仅 11724 个 GPU 秒）。这证明了自增长策略能够在提高模型性能的同时提升其训练和推理效率。相比之下，虽然剪枝方法能在一定程度上减少模型的参数量和计算量，但其在优化过程中存在着明显的时间和资源方面的负担，尤其是在需要微调以恢复或提升性能的情况下。

通过对模型剪枝与自增长的对比，本研究发现尽管剪枝方法能够有效减轻模型负担，但在性能恢复的过程中通常需要微调，导致优化时间延长。相比之下，自增长模型通过动态调整模型结构，不仅实现了更高的准确率，而

且显著降低了推理延迟和总体优化耗时，展现了其在模型优化中的高效性。

### 3.5.5 与其他自增长方法的对比

业界对模型自增长方法的探索处于初步阶段，目前已经出现几种有前景的自增长方法，主要包括基于随机深度自增长的 RandGrow<sup>[27]</sup>、侧重于保持宽度增长前后输出一致性的 SplitGrow<sup>[28]</sup>，以及利用参数动量初始化新层的 MoGrow<sup>[29]</sup>。这些方法采取不同的策略来动态调整模型结构，在各异的数据集下实施相应的模型优化。本研究将深入对比这些自增长方法与我们提出的数据集自适应的自增长方法，记为 AdaGrow，以评估它们在各种数据集上的性能效率区别。

表 3-31 不同自增长方法的效果对比

| 模型                          | 架构      | 参数量     | 计算量     | 推理延迟(ms) | 性能(acc%) |
|-----------------------------|---------|---------|---------|----------|----------|
| RandGrow-ResNet             | 1-3-2   | 3.69M   | 492.90M | 3.28     | 89.76    |
| RandGrow-ResNet             | 1-2-2-2 | 14.02M  | 587.34M | 5.49     | 89.68    |
| RandGrow-Bottleneck-ResNet  | 2-3-2   | 576.39K | 72.42M  | 4.12     | 89.48    |
| RandGrow-Bottleneck-ResNet  | 2-2-3-2 | 2.37M   | 100.28M | 5.22     | 90.28    |
| SplitGrow-ResNet            | 1-2-2   | 2.33M   | 443.58M | 2.8      | 91.33    |
| SplitGrow-ResNet            | 1-2-2-1 | 5.291M  | 490.90M | 3.37     | 91.47    |
| SplitGrow-Bottleneck-ResNet | 2-2-2   | 435.47K | 71.21M  | 2.84     | 90.52    |
| SplitGrow-Bottleneck-ResNet | 2-2-2-2 | 1.81M   | 120.21M | 4.77     | 91.3     |
| MoGrow-ResNet               | 2-2-2   | 3.48M   | 494.73M | 2.09     | 90.41    |
| MoGrow-ResNet               | 1-2-2-2 | 14.03M  | 588.78M | 2.28     | 90.88    |
| MoGrow-Bottleneck-ResNet    | 2-2-2   | 559.08K | 68.03M  | 2.22     | 88.05    |
| MoGrow-Bottleneck-ResNet    | 2-3-3-2 | 2.39M   | 105.19M | 3.15     | 90.71    |
| AdaGrow-ResNet              | 1-2-2   | 1.17M   | 222.99M | 1.9      | 92.22    |
| AdaGrowResNet               | 1-2-2-1 | 2.65M   | 246.66M | 2.11     | 93.14    |
| AdaGrow-Bottleneck-ResNet   | 2-2-2   | 235.07K | 41.09M  | 2.18     | 90.08    |
| AdaGrow-Bottleneck-ResNet   | 2-2-2-2 | 673.21K | 48.14M  | 2.89     | 90.94    |
| RandGrow-VGG                | 2-3-2   | 2.07M   | 304.16M | 2.06     | 89.52    |
| RandGrow-VGG                | 2-2-2-2 | 7.82M   | 360.85M | 2.51     | 90.08    |
| SplitGrow-VGG               | 1-2-2   | 1.34M   | 244.35M | 1.77     | 90.01    |
| SplitGrow-VGG               | 1-1-2-2 | 4.12M   | 249.01M | 2.17     | 90.48    |
| MoGrow-VGG                  | 2-2-2   | 1.92M   | 367.32M | 1.06     | 88.33    |
| MoGrow-VGG                  | 2-2-2-2 | 7.83M   | 361.83M | 1.95     | 89.65    |



|                       |         |         |         |      |       |
|-----------------------|---------|---------|---------|------|-------|
| AdaGrow-VGG           | 1-2-2   | 669.29K | 123.37M | 1.08 | 90.72 |
| AdaGrow-VGG           | 1-1-2-2 | 2.06M   | 125.72M | 1.37 | 91.44 |
| RandGrow-MobileNetV3  | 3-4-3   | 1.27M   | 28.46M  | 7.05 | 85.52 |
| RandGrow-MobileNetV3  | 3-3-4-2 | 1.82M   | 16.86M  | 8.31 | 86.52 |
| SplitGrow-MobileNetV3 | 2-3-3   | 1.07M   | 30.16M  | 5.02 | 84.59 |
| SplitGrow-MobileNetV3 | 2-2-3-2 | 931.77K | 11.59M  | 5.69 | 85.1  |
| MoGrow-MobileNetV3    | 3-3-3   | 1.20M   | 26.51M  | 4.11 | 87.03 |
| MoGrow-MobileNetV3    | 3-3-3-2 | 1.67M   | 15.86M  | 4.85 | 87.6  |
| AdaGrow-MobileNetV3   | 2-3-3   | 811.40K | 20.37M  | 3.71 | 89.32 |
| AdaGrow-MobileNetV3   | 2-2-3-2 | 670.52K | 7.24M   | 4.4  | 88.31 |

表 3-31 展示了不同自增长方法在 CIFAR-10 图像识别中的效果。实验结果表明，本研究所提的 AdaGrow 在提高模型准确率的同时，有效降低了参数量和计算量，展现出优异的性能和高效的计算资源利用。与其他自增长策略相比，AdaGrow 在全模型上取得了更高的性能，证明了其在模型结构优化和提升计算效率方面的显著优势。

### 3.6 本章小结

第三章深入探讨了卷积神经网络和 Transformer 模型自增长的各个方面，包括宽度增长和深度增长的策略、新增模块的参数初始化方法、优化器的调整策略、模型增长的频率以及增长的终止条件。宏观来看，为模型的重要表达位置添加功能更丰富的特征提取结构，能够实现更优的自增长效果。通过对这些关键因素的系统分析和优化，本研究建立了一个全面的模型自增长框架，旨在动态调整模型结构以适应不同的数据集需求。实验结果验证了模型自增长策略在不同数据集上的出色适应性，并展示了其在性能效率平衡方面的显著能力。与传统的模型剪枝方法及其他自增长策略相比，本研究所提方法在提升模型性能、优化计算资源使用以及降低优化复杂度等方面表现出明显优势，证明了其在深度学习模型优化领域的实用性和有效性。

## 4 基于误差弥补和权重重排的自增长模型量化方法

### 4.1 自增长模型的量化挑战

本研究在深入探索自增长模型的过程中，发现这种结构调整策略给模型量化带来了新的挑战。首先，自增长模型通过动态调整其结构以适应特定的数据集需求，通常体现出相对较小的模型规模。这种低规模的模型将不再有明显的过参数化现象，即模型中的冗余参数或冗余位宽的比例降低。模型量化是模型部署加速的经典方式之一，过参数化被视为量化成功的关键，因为删除参数所承载的冗余位宽不会显著影响模型的性能。然而，自增长模型由于对数据集的高度适应，每个参数所承载的冗余位宽较少，因此对量化过程中的误差更为敏感，容易造成较高的性能损失，图 4-1 形象地展示了自增长模型的低过参数化现象。

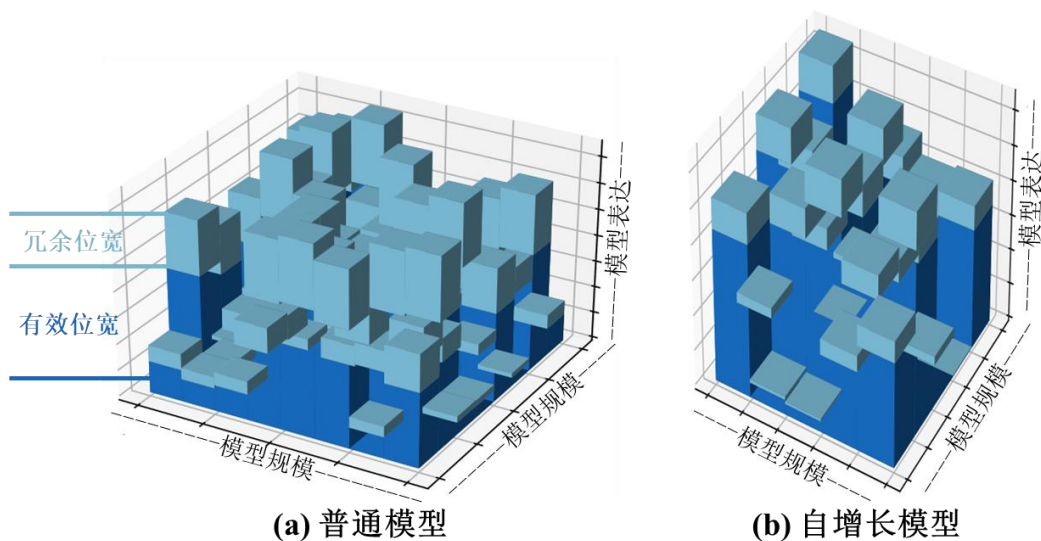


图 4-1 自增长模型的低过参数化现象

另一方面，卷积神经网络在自增长的过程中，采用了基于结构重参数化模拟宽度增长策略，这种策略进一步加剧了量化的难度：在重参数化过程中，不同大小的卷积核将会向最大的卷积核合并，例如  $5 \times 5$ 、 $3 \times 3$  和  $1 \times 1$  卷积核合并成一个  $5 \times 5$  卷积核。这将导致合并后卷积核中心区域的参数值被多次叠加放大，使得整个卷积核的参数分布出现了高度非均匀的特点。这种分布形态远离了量化策略常假设的高斯分布或均匀分布，而是形成了一种不利于量化的具有更高方差的尖峰分布。另一方面，卷积核中心部分的大幅度值的重要性将明显高于其他值，造成了量化敏感度的不均匀现象：对大幅度值进行

量化产生的损失将会明显高于量化其他参数。以上因素将导致量化后模型性能的显著下降，图 4-2 展示了上述现象。

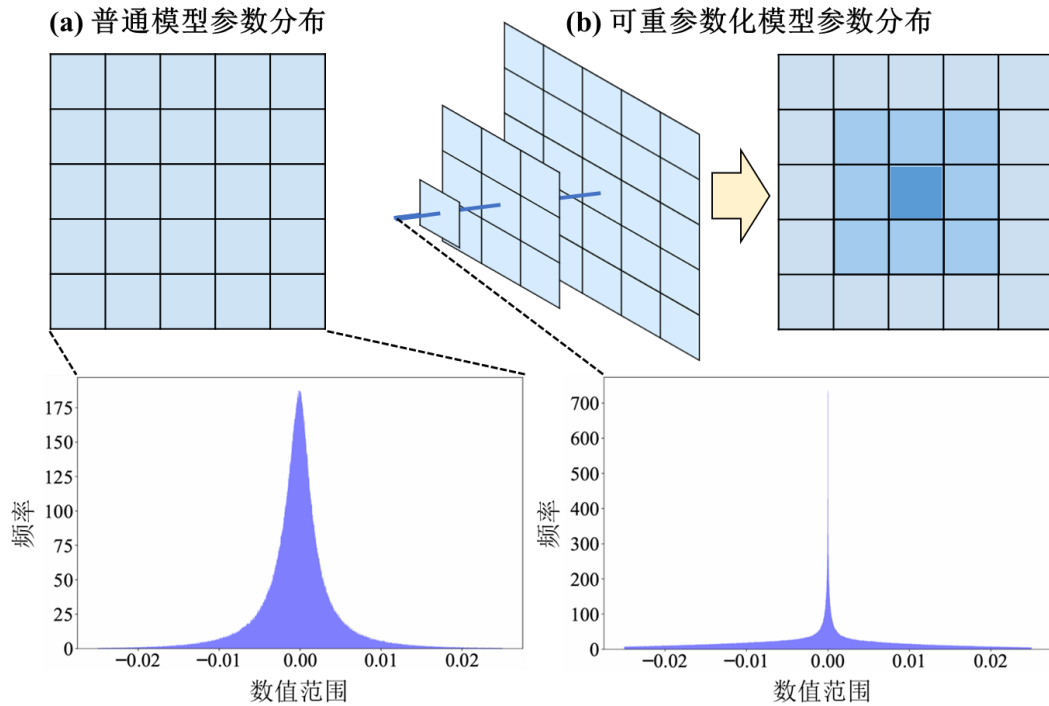


图 4-2 可重参数化模型的尖锐权重分布现象

综上所述，自增长模型的结构特性使得其在量化过程中面临额外的挑战。这不仅涉及更低的模型规模减少了过参数化的空间，还包括重参数化策略引入的权重分布的非均匀性，均对量化过程的效率和效果产生了影响。因此，为了实现自增长模型的有效量化，需要进一步探索和开发能应对这些特殊挑战的量化策略。

## 4.2 自增长模型的量化方案

面对自增长模型在量化过程中可能遇到的较高量化误差问题，本研究拟通过误差弥补的策略来缓和总体量化误差的降低。误差弥补策略的核心在于，通过对模型部分参数进行初步量化并计算由此引起的量化误差，随后根据最优化原理，对尚未量化的参数部分执行有针对性的权重调整，用以补偿由于初步量化所产生的误差，以期在执行完成所有轮次的量化时，整体的量化误差有所减少。通过这样迭代优化的方法，就有可能在保持模型性能的同时，有效减轻量化带来的负面影响，进而促进自增长模型在资源受限环境下的高效量化部署。

### 4.2.1 误差弥补

本研究计划采取的基本误差弥补方法是 Optimal Brain Surgeon<sup>[103]</sup> (OBS)，这是一种基于 Hessian 矩阵的二次近似方法，专注于最小化因参数量化而造成的性能损失。OBS 通过分析模型损失函数对参数的二阶导数，即 Hessian 矩阵，来确定调整某些参数对模型性能的影响程度并对它们进行压缩或调整，以尽可能保持模型整体性能的稳定性。对于一个以  $X_l$  作为输入数据， $W_l$  作为参数的卷积层或全连接层，假设  $\hat{W}_l$  是压缩后的权重，优化的目标则为：

$$\operatorname{argmin}_{\hat{W}_l} \|W_l X_l - \hat{W}_l X_l\|_2^2 \quad (4-1)$$

根据 OBS 的理论对(4-1)进行优化，针对量化压缩方式，结论如下：

$$w_p = \operatorname{argmin}_{w_p} \frac{(\operatorname{quant}(w_p) - w_p)^2}{[H^{-1}]_{pp}} \quad (4-2)$$

$$\delta_p = -\frac{w_p - \operatorname{quant}(w_p)}{[H^{-1}]_{pp}} H_{:,p}^{-1} \quad (4-3)$$

其中  $w_p$  是待量化权重，它们的量化误差较小，然而在实际操作中，由于需要对模型中所有权重进行相同粒度的量化，因此除非特别需要采取混合精度量化，通常采用的都是顺序量化方法，即按照权重矩阵的行或列方向依次进行量化。 $\operatorname{quant}(w_p)$  代表了量化后的权重值。 $\delta_p$  代表的是待量化权重完成量化后，未量化权重需要做出的调整量，即  $w_{p:} = w_{p:} - \delta_p$ 。 $H$  代表的是 Hessian 矩阵，一般使用  $XX^T$  来近似， $X$  可以选取小部分训练数据作为校准集。

在自增长模型的量化过程中，上述方法将能够有效地补偿量化所引入的误差：通过有目的地调整那些未经量化的参数，减少整体量化误差，从而降低模型在量化后潜在的性能下降问题。

### 4.2.2 权重重新排布

权重调整虽然能够通过弥补量化误差，部分解决自增长模型量化后的性能下降的问题，然而，4.2.1 所提供的常规误差弥补方法没有考虑到卷积神经网络的基于结构重参数化的模拟宽度增长方法，因此难以有针对性地对自增长卷积神经网络进行误差弥补。

在结构重参数化的过程中，中心位置的参数由于多次累积计算，其数值幅度相比边缘位置的参数被明显加强，导致融合后的权重分布呈现出中心重要性趋势。这种现象发生是因为在结构重参数化过程中，中心位置的参数通常参与更多路径的信息整合，接收来自多个不同卷积核的信息输入，使得这些位置的权重在整体网络表达中占据更加关键的作用。而边缘位置的参数通常只与局部的输入信息相关联，其在网络中的作用相对较小。这种中心位置参数强化的现象使得权重分布偏离了常规的均匀或高斯分布，而呈现出一种中心峰值型的高方差尖锐分布。在自增长卷积神经网络中，中心位置的权重变得更为关键，其对模型性能的贡献远超其他位置。因此，如果仅采取常规的量化策略对权重进行压缩，不加区分地对整个权重进行统一地量化，则会引入较大的量化误差，导致性能下降。

为了解决上述问题，本研究注意到能够反映权重量化误差幅度的重要性指标  $\frac{(\text{quant}(w_p) - w_p)^2}{[H^{-1}]_{pp}}$ ，并提出如下方法：保持原始的顺序量化逻辑不变，将最重要的中心位置所对应的权重列暂时置换到首列的位置，并优先量化，这时就会有更多的其他权重为它做出调整来更充分地弥补其量化误差。以同样的思路应对“次中心位置”的参数，这些参数对应着多列，这些列的整体将被暂时置换至较早被量化的位置，而这些列的内部，将按照重要性指标进行重排。以此类推，保证了越重要的参数越被优先量化，越有更多未量化参数帮其弥补误差，这种方式通过优化误差弥补的充分性提升了量化后模型的性能。

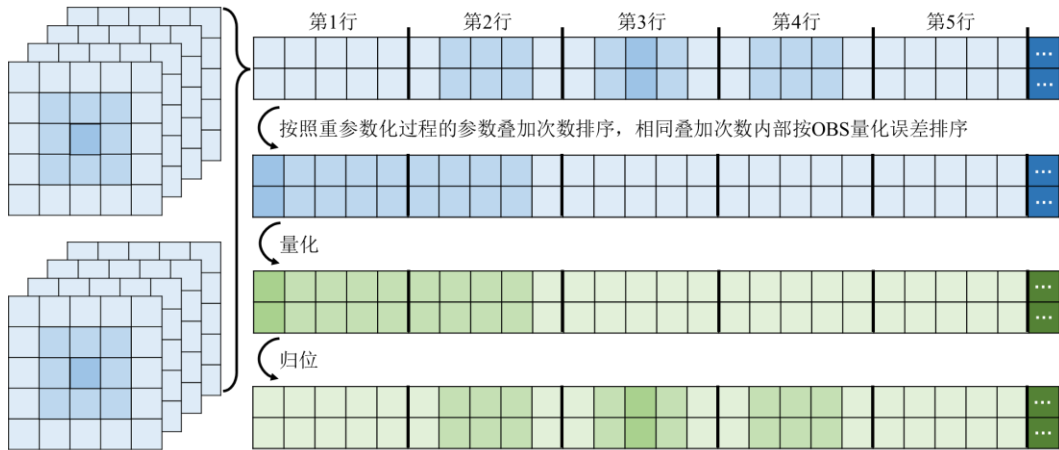


图 4-3 基于误差弥补和权重重新排布的自增长模型量化方法

图 4-3 展示了对卷积层参数的处理方式，以一个参数形状为[2, 4, 5, 5]的卷积层为例，在对它进行量化前，需要将其从输入通道维度进行展开，得到形状为[2, 100]的二维矩阵（图中仅展示了一个输入通道的情况，实际的处理流程应当全局统一重新排布）。首先将权重按照其在结构重参数化过程中的

被叠加次数进行重新排布，然后在每种不同的叠加次数下，按照重要性指标进行内部的重新排布，得到一个可以按照顺序压缩逻辑进行量化的矩阵。接着对这个矩阵进行量化和误差弥补，并按照重新排布过程中记录的排布顺序，进行一次反向重新排布，使参数的排布回归正常顺序，以此作为量化后的参数（需要再次进行卷积展开的反过程，还原至 $[2, 4, 5, 5]$ 形状）。以上方法可以无缝适用于自增长 Transformer 结构，其只需要进行全局重新排布即可。

基于误差弥补和权重重新排布的自增长模型量化方法有效地利用了重要性指标，通过优化参数的量化顺序，能够显著降低自增长卷积神经网络在量化过程中的误差，缓解模型性能下降的问题，为自增长模型的压缩与部署提供了有针对性的高效解决方案。

### 4.3 自增长模型的量化加速实践

通过对自增长模型量化挑战的深入分析（4.1）和针对这些挑战提出的优化量化策略（4.2）。本研究将在本部分通过实际案例，在不同的硬件平台上验证模型的实际量化效果，具体包括 AMD EPYC 7642 128G 96 核 CPU 和 NVIDIA RTX 4090 24GB GPU。选定的自增长网络模型为 AdaGrow-Basic-ResNet 和 AdaGrow-ViT，针对 CIFAR10 数据集。这些模型将经历三种不同的量化位宽处理：Floatpoint-16（FP16）、Integer-8（INT8）以及 Weight-Integer-4-Activation-Floatpoint-16（W4A16），并以 Pytorch 或 TensorRT 后端执行推理，以此展示自增长网络在实际部署环境下的应用潜力及效率。

表 4-1 自增长卷积神经网络的量化加速实践

| 模型                       | 量化配置     |       |          | 推理延迟    |                | 性能<br>(acc%) |
|--------------------------|----------|-------|----------|---------|----------------|--------------|
|                          | 方法       | 位宽    | 推理后端     | CPU     | GPU            |              |
| ResNet-50                | 直接<br>量化 | FP32  | Pytorch  | 10.95ms | 3.52ms         | 86.12        |
|                          |          | FP32  | TensorRT | -       | 948.63 $\mu$ s | 86.12        |
|                          |          | FP16  | TensorRT | -       | 693.50 $\mu$ s | 86.48        |
|                          |          | INT8  | TensorRT | -       | 675.65 $\mu$ s | 85.91        |
|                          |          | W4A16 | Pytorch  | -       | 3.05ms         | 85.04        |
| AdaGrow-Basic-<br>ResNet | 直接<br>量化 | FP32  | Pytorch  | 2.94ms  | 1.39ms         | 93.14        |
|                          |          | FP32  | TensorRT | -       | 261.11 $\mu$ s | 93.14        |
|                          |          | FP16  | TensorRT | -       | 166.46 $\mu$ s | 93.06        |
|                          |          | INT8  | TensorRT | -       | 155.88 $\mu$ s | 92.27        |
|                          |          | W4A16 | Pytorch  | -       | 1.03ms         | 89.13        |

|                      |             |       |          |        |                |       |
|----------------------|-------------|-------|----------|--------|----------------|-------|
| AdaGrow-Basic-ResNet | OBS         | FP32  | Pytorch  | 2.94ms | 1.39ms         | 93.14 |
|                      |             | FP32  | TensorRT | -      | 261.11 $\mu$ s | 93.14 |
|                      |             | FP16  | TensorRT | -      | 166.46 $\mu$ s | 93.12 |
|                      |             | INT8  | TensorRT | -      | 155.88 $\mu$ s | 92.56 |
|                      |             | W4A16 | Pytorch  | -      | 1.03ms         | 91.66 |
| AdaGrow-Basic-ResNet | 本研究<br>所提方法 | FP32  | Pytorch  | 2.94ms | 1.39ms         | 93.14 |
|                      |             | FP32  | TensorRT | -      | 261.11 $\mu$ s | 93.14 |
|                      |             | FP16  | TensorRT | -      | 166.46 $\mu$ s | 93.16 |
|                      |             | INT8  | TensorRT | -      | 155.88 $\mu$ s | 93.05 |
|                      |             | W4A16 | Pytorch  | -      | 1.03ms         | 92.98 |

表 4-1 展示了自增长卷积神经网络的量化加速实践结果，ResNet-50 作为固定规模的基线模型，直接对其进行量化并未产生较高的量化损失，反而表现出较好的量化鲁棒性。相反，自增长模型 AdaGrow-Basic-ResNet 在直接量化后性能损失更为明显，这契合了 4.1 中介绍的自增长模型在量化过程中面临的挑战。在实施 FP16 和 INT8 精度下的直接量化时，AdaGrow-Basic-ResNet 模型在精度上有所下降，但 OBS 方法以及本研究所提的基于误差弥补和权重重新排布的自增长模型量化方法有效减缓了性能损失，且后者的性能损失更低。在更严格的 W4A16 精度下，本研究所提方法显著提升了量化后的模型精度，进一步超越了 OBS 方法，证明了本研究提出的方法的有效性。在推理延迟方面，通过 TensorRT 优化后的自增长模型展现出显著加速效果，几乎比 ResNet-50 快了 4 倍，这说明更简洁的模型更能够在后续编译优化中获益，展示了自增长模型在推理加速方面的潜力。

表 4-2 自增长 Vision Transformer 的量化加速实践

| 模型          | 量化配置     |       |          | 推理延迟    |                 | 性能<br>(acc%) |
|-------------|----------|-------|----------|---------|-----------------|--------------|
|             | 方法       | 位宽    | 推理后端     | CPU     | GPU             |              |
| ViT-Base    | 直接<br>量化 | FP32  | Pytorch  | 56.65ms | 5.62ms          | 74.6         |
|             |          | FP32  | TensorRT | -       | 1457.52 $\mu$ s | 74.6         |
|             |          | FP16  | TensorRT | -       | 700.17 $\mu$ s  | 74.6         |
|             |          | INT8  | TensorRT | -       | 638.82 $\mu$ s  | 74.42        |
|             |          | W4A16 | Pytorch  | -       | 5.28ms          | 73.83        |
| AdaGrow-ViT | 直接<br>量化 | FP32  | Pytorch  | 7.96ms  | 2.16ms          | 77.74        |
|             |          | FP32  | TensorRT | -       | 278.82 $\mu$ s  | 77.74        |
|             |          | FP16  | TensorRT | -       | 192.62 $\mu$ s  | 77.68        |
|             |          | INT8  | TensorRT | -       | 176.44 $\mu$ s  | 77.11        |
|             |          | W4A16 | Pytorch  | -       | 1.82ms          | 75.81        |



|             |             |       |          |        |                |       |
|-------------|-------------|-------|----------|--------|----------------|-------|
| AdaGrow-ViT | OBS         | FP32  | Pytorch  | 7.96ms | 2.16ms         | 77.74 |
|             |             | FP32  | TensorRT | -      | 278.82 $\mu$ s | 77.74 |
|             |             | FP16  | TensorRT | -      | 192.62 $\mu$ s | 77.74 |
|             |             | INT8  | TensorRT | -      | 176.44 $\mu$ s | 77.43 |
|             |             | W4A16 | Pytorch  | -      | 1.82ms         | 76.67 |
| AdaGrow-ViT | 本研究<br>所提方法 | FP32  | Pytorch  | 7.96ms | 2.16ms         | 77.74 |
|             |             | FP32  | TensorRT | -      | 278.82 $\mu$ s | 77.74 |
|             |             | FP16  | TensorRT | -      | 192.62 $\mu$ s | 77.74 |
|             |             | INT8  | TensorRT | -      | 176.44 $\mu$ s | 77.81 |
|             |             | W4A16 | Pytorch  | -      | 1.82ms         | 76.88 |

表 4-2 展示了自增长 Vision Transformer 的量化加速实践结果，直接量化的自增长模型 AdaGrow-ViT 同样显示出相较于 ViT-Base 更明显的性能损失。采用 OBS 及本研究所提方法进行量化优化后，性能损失显著减少，特别是在低位宽配置下，本研究所提方法能更好地提升性能，验证了本研究所提优化量化方法的有效性。TensorRT 的优化，再次显著加速了自增长模型在 GPU 上的推理。

## 4.4 本章小结

在本章中，我们探讨了自增长模型在量化过程中面临的挑战，并提出了一种基于误差弥补和权重重新排布的量化方法来解决这种挑战，该方法能够降低量化引入的误差，以此优化模型性能。通过对比实验，本研究证明了该方法能够有效提高自增长模型的量化后性能：无论是自增长得到的卷积神经网络模型还是 Transformer 模型，该方法均展现出了优秀的量化误差弥补能力并有效提升了模型的量化后性能，为模型量化提供了新的思路和方法。

## 5 结论与展望

面对在特定数据集条件下实现神经网络性能与推理效率最优平衡的需求,本研究提出了一种模型自增长框架,通过在训练过程中逐步扩展模型的规模,适应性地达到针对特定数据集的最优架构,并通过深入研究和优化增长方案,有效实现了更高的识别精度和更快的推理速度。为高效部署神经网络模型提供了一种新的解决方案。

(1) 本研究提出的模型自增长训练范式,旨在通过逐步增长网络的规模来实现对特定数据集的自适应,达成理想的性能效率平衡。本研究深入探索了网络自增长的关键因素,包括深度与宽度增长方式、增长频率、增长终止条件、单次增长的规模、新增模块的初始化方法以及优化器的调整策略等,并据此构建了一套最优的模型自增长框架。特别地,本研究通过结构重参数化技术实现了卷积神经网络宽度模拟增长,既保持了模型性能,又有效地避免了直接增宽可能导致的效率损失,进一步体现了在性能保持与计算效率之间的优秀权衡。此外,该框架还成功地扩展到了基于 Transformer 的若干主流架构上。本框架不仅在特定数据集下实现了卓越的性能和高效率,还在训练过程中简化了正则化、学习率调整和优化器设计,显著提升了训练效率,体现了其广泛的实用价值。

(2) 本研究进一步发现了自增长模型在量化加速应用中可能遇到的挑战:缺乏明显的过参数化现象,以及卷积神经网络中由结构重参数化引起的权重分布异常等因素导致了在量化过程中可能产生较大的误差,进而影响量化后的模型性能。为此本研究提出了一种创新的基于误差弥补和权重重新排布的量化方法,这一方法有效降低了量化带来的性能损失,为高效量化自增长模型提供了新的解决方案。

通过广泛的实验验证,本研究所提出的模型自增长框架在计算机视觉和自然语言处理等多种数据集任务上均展示了其优异的性能与效率。同时,本研究提出的基于误差弥补和权重重新排布的量化方法,能够成功应用于卷积神经网络模型和 Transformer 模型,实现了在更低性能损失下进行更高效的量化压缩。这些成果为面向数据集自适应的模型高效训练和推理提供了新的技术路线。

本研究虽然实现了良好的数据集自适应性和理想的性能效率平衡,却仍然存在一些改进的空间:

(1) 面对数据集自适应性的需求,现有的模型自增长框架尚未能够提供

一个以目标性能为基准的增长终止策略。设定一个具体的性能目标，并据此自动决定模型增长的停止时机，对于用户来说极为便利和实用。这就要求设计一种机制，来提前预知当前模型在训练至收敛后的最终性能。为此，本研究计划在未来引入一种创新的解决方案：通过构建和训练一个辅助预测模型来预测当前网络结构在训练完成后可能达到的性能水平。挑战在于如何有效收集和利用模型结构与其最终性能之间的关系数据，并确保该预测模型能够跨数据集泛化。

(2) 基于误差弥补和权重重新排布的量化方法在进行按照结构重参数化叠加次数（如有）以及量化误差大小对权重的行/列进行重新排布后，将按照顺序进行权重量化和未量化权重的调整，每次误差调整结束后，未量化的权重列的重要性分布可能会出现变化，本研究考虑到复杂性问题，没有进一步依照新的重要性指标来再次重新排布权重，这将在未来的工作中继续探索。

## 参考文献

- [1] 张驰,郭媛,黎明.人工神经网络模型发展及应用综述[J].计算机工程与应用,2021,57(11):57-69.
- [2] Smys S, Chen J I Z, Shakya S. Survey on neural network architectures with deep learning[J]. Journal of Soft Computing Paradigm (JSCP), 2020, 2(03): 186-194.
- [3] Mcculloch W S, Pitts W. A logical calculus of the ideas immanent in nervous activity[J]. The bulletin of mathematical biophysics, 1943, 5: 115-133.
- [4] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors[J]. nature, 1986, 323(6088): 533-536.
- [5] Al-Kaff A, Martin D, Garcia F, et al. Survey of computer vision algorithms and applications for unmanned aerial vehicles[J]. Expert Systems with Applications, 2018, 92: 447-463.
- [6] Eronen A J, Peltonen V T, Tuomi J T, et al. Audio-based context recognition[J]. IEEE Transactions on Audio, Speech, and Language Processing, 2005, 14(1): 321-329.
- [7] Shen D, Wu G, Suk H-I. Deep learning in medical image analysis[J]. Annual review of biomedical engineering, 2017, 19: 221-248.
- [8] Lv Y, Duan Y, Kang W, et al. Traffic flow prediction with big data: A deep learning approach[J]. Ieee transactions on intelligent transportation systems, 2014, 16(2): 865-873.
- [9] Li Z, Liu F, Yang W, et al. A survey of convolutional neural networks: analysis, applications, and prospects[J]. IEEE transactions on neural networks and learning systems, 2021, 33(12): 6999-7019.
- [10] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]. Advances in neural information processing systems, 2017: 5998-6008.
- [11] 林景栋,吴欣怡,柴毅等.卷积神经网络结构优化综述[J].自动化学报,2020,46(01):24-37.
- [12] 石磊,王毅,成颖等.自然语言处理中的注意力机制研究综述[J].数据分析与知识发现,2020,4(05):1-14.
- [13] Han K, Wang Y, Chen H, et al. A survey on vision transformer[J]. IEEE

- transactions on pattern analysis and machine intelligence, 2022, 45(1): 87-110.
- [14] Md V, Misra S, Ma G, et al. Distgmn: Scalable distributed training for large-scale graph neural networks[C]. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021: 1-14.
  - [15] Zhang C, Zhang C, Li C, et al. One small step for generative ai, one giant leap for agi: A complete survey on chatgpt in aigc era[J]. arXiv preprint arXiv:2304.06488, 2023.
  - [16] 王金强,黄航,郅朋等. 自动驾驶发展与关键技术综述[J]. 电子技术应用, 2019, 45(06): 28-36.
  - [17] Xiao P, Qin Z, Chen D, et al. FastNet: A lightweight convolutional neural network for tumors fast identification in mobile computer-assisted devices[J]. IEEE Internet of Things Journal, 2023, 10(11): 9878-9891.
  - [18] Anwar S, Hwang K, Sung W. Structured pruning of deep convolutional neural networks[J]. ACM Journal on Emerging Technologies in Computing Systems (JETC), 2017, 13(3): 1-18.
  - [19] Xiao G, Lin J, Seznec M, et al. Smoothquant: Accurate and efficient post-training quantization for large language models[C]. International Conference on Machine Learning, 2023: 38087-38099.
  - [20] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
  - [21] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016: 770-778.
  - [22] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: transformers for image recognition at scale[C]. International Conference on Learning Representations, 2020.
  - [23] Tarwani K M, Edem S. Survey on recurrent neural network in natural language processing[J]. Int. J. Eng. Trends Technol, 2017, 48(6): 301-304.
  - [24] Kenton J D M-W C, Toutanova L K. BERT: Pre-training of deep bidirectional transformers for language understanding[C]. Proceedings of NAACL-HLT, 2019: 4171-4186.

- [25] Ding X, Guo Y, Ding G, et al. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks[C]. Proceedings of the IEEE/CVF international conference on computer vision, 2019: 1911-1920.
- [26] Ren P, Xiao Y, Chang X, et al. A comprehensive survey of neural architecture search: Challenges and solutions[J]. ACM Computing Surveys (CSUR), 2021, 54(4): 1-34.
- [27] Wen W, Yan F, Chen Y, et al. Autogrow: Automatic layer growing in deep convolutional networks[C]. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020: 833-841.
- [28] Yuan X, Savarese P, Maire M. Accelerated training via incrementally growing neural networks using variance transfer and learning rate adaptation[C]. Advances in Neural Information Processing Systems, 2023: 3601-3619.
- [29] Li C, Zhuang B, Wang G, et al. Automated progressive learning for efficient training of vision transformers[C]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022: 12486-12496.
- [30] Bjorck N, Gomes C P, Selman B, et al. Understanding batch normalization[C]. Advances in neural information processing systems, 2018: 7705-7716.
- [31] Han J, Moraga C. The influence of the sigmoid function parameters on the speed of backpropagation learning[C]. International workshop on artificial neural networks, 1995: 195-201.
- [32] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines[C]. Proceedings of the 27th international conference on machine learning (ICML-10), 2010: 807-814.
- [33] Hendrycks D, Gimpel K. Gaussian error linear units (gelus)[J]. arXiv preprint arXiv:1606.08415, 2016.
- [34] Nader A, Azar D. Searching for activation functions using a self-adaptive evolutionary algorithm[C]. Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, 2020: 145-146.
- [35] Yu W, Luo M, Zhou P, et al. Metaformer is actually what you need for vision[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022: 10819-10829.
- [36] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional

- neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.
- [37] Tan M, Le Q. Efficientnet: Rethinking model scaling for convolutional neural networks[C]. International conference on machine learning, 2019: 6105-6114.
  - [38] Sandler M, Howard A, Zhu M, et al. Mobilenetv2: Inverted residuals and linear bottlenecks[C]. Proceedings of the IEEE conference on computer vision and pattern recognition, 2018: 4510-4520.
  - [39] Howard A, Sandler M, Chu G, et al. Searching for mobilenetv3[C]. Proceedings of the IEEE/CVF international conference on computer vision, 2019: 1314-1324.
  - [40] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]. Proceedings of the IEEE conference on computer vision and pattern recognition, 2018: 7132-7141.
  - [41] Goel K, Vohra R, Sahoo J K. Polyphonic music generation by modeling temporal dependencies using a rnn-dbn[C]. Artificial Neural Networks and Machine Learning–ICANN 2014: 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014. Proceedings 24, 2014: 217-224.
  - [42] Zhu Q, Luo J. Generative pre-trained transformer for design concept generation: an exploration[J]. Proceedings of the design society, 2022, 2: 1825-1834.
  - [43] Xu J, Sun X, Zhang Z, et al. Understanding and improving layer normalization[C]. Advances in neural information processing systems, 2019: 4383-4393.
  - [44] Liu Y, Ott M, Goyal N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.
  - [45] Lan Z, Chen M, Goodman S, et al. Albert: A lite bert for self-supervised learning of language representations[J]. arXiv preprint arXiv:1909.11942, 2019.
  - [46] Raffel C, Shazeer N, Roberts A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. Journal of machine learning research, 2020, 21(140): 1-67.
  - [47] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report[J]. arXiv preprint



arXiv:2303.08774, 2023.

- [48] Touvron H, Cord M, Douze M, et al. Training data-efficient image transformers & distillation through attention[C]. International conference on machine learning, 2021: 10347-10357.
- [49] Liu Z, Lin Y, Cao Y, et al. Swin transformer: Hierarchical vision transformer using shifted windows[C]. Proceedings of the IEEE/CVF international conference on computer vision, 2021: 10012-10022.
- [50] Blalock D, Gonzalez Ortiz J J, Frankle J, et al. What is the state of neural network pruning?[J]. Proceedings of machine learning and systems, 2020, 2: 129-146.
- [51] Liu Z, Li J, Shen Z, et al. Learning efficient convolutional networks through network slimming[C]. Proceedings of the IEEE international conference on computer vision, 2017: 2736-2744.
- [52] He Y, Zhang X, Sun J. Channel pruning for accelerating very deep neural networks[C]. Proceedings of the IEEE international conference on computer vision, 2017: 1389-1397.
- [53] Chollet F. Xception: Deep learning with depthwise separable convolutions[C]. Proceedings of the IEEE conference on computer vision and pattern recognition, 2017: 1251-1258.
- [54] Park J-H, Kim Y, Kim J, et al. Dynamic structure pruning for compressing CNNs[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 2023: 9408-9416.
- [55] Fang G, Ma X, Song M, et al. Depgraph: Towards any structural pruning[C]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023: 16091-16101.
- [56] Yin M, Uzket B, Shen Y, et al. Gohsp: A unified framework of graph and optimization-based heterogeneous structured pruning for vision transformer[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 2023: 10954-10962.
- [57] Kwon W, Kim S, Mahoney M W, et al. A fast post-training pruning framework for transformers[J]. Advances in Neural Information Processing Systems, 2022, 35: 24101-24116.
- [58] Sanh V. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and

- lighter[C]. Proceedings of Thirty-third Conference on Neural Information Processing Systems (NIPS2019), 2019.
- [59] Lecun Y, Denker J, Solla S. Optimal brain damage[C]. Advances in neural information processing systems, 1989: 598-605.
- [60] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding[J]. arXiv preprint arXiv:1510.00149, 2015.
- [61] Frantar E, Alistarh D. Sparsegpt: Massive language models can be accurately pruned in one-shot[C]. International Conference on Machine Learning, 2023: 10323-10337.
- [62] Fang C, Zhou A, Wang Z. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2022, 30(11): 1573-1586.
- [63] Balasubramaniam A, Sunny F, Pasricha S. R-TOSS: A framework for real-time object detection using semi-structured pruning[C]. 2023 60th ACM/IEEE Design Automation Conference (DAC), 2023: 1-6.
- [64] Jiang P, Ergu D, Liu F, et al. A Review of Yolo algorithm developments[J]. Procedia computer science, 2022, 199: 1066-1073.
- [65] Lin T-Y, Goyal P, Girshick R, et al. Focal loss for dense object detection[C]. Proceedings of the IEEE international conference on computer vision, 2017: 2980-2988.
- [66] Grimaldi M, Ganji D C, Lazarevich I, et al. Accelerating deep neural networks via semi-structured activation sparsity[C]. Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023: 1179-1188.
- [67] Holmes C, Zhang M, He Y, et al. NxMTransformer: semi-structured sparsification for natural language understanding via ADMM[J]. Advances in neural information processing systems, 2021, 34: 1818-1830.
- [68] Boyd S, Parikh N, Chu E, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers[J]. Foundations and Trends® in Machine learning, 2011, 3(1): 1-122.
- [69] Yang J, Shen X, Xing J, et al. Quantization networks[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019: 7308-7316.

- [70] Cai Y, Yao Z, Dong Z, et al. Zeroq: A novel zero shot quantization framework[C]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020: 13169-13178.
- [71] Nagel M, Amjad R A, Van Baalen M, et al. Up or down? adaptive rounding for post-training quantization[C]. International Conference on Machine Learning, 2020: 7197-7206.
- [72] Li R, Wang Y, Liang F, et al. Fully quantized network for object detection[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019: 2810-2819.
- [73] Bhalgat Y, Lee J, Nagel M, et al. Lsq+: Improving low-bit quantization through learnable offsets and better initialization[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, 2020: 696-697.
- [74] Li Y, Xu S, Zhang B, et al. Q-vit: Accurate and fully quantized low-bit vision transformer[J]. Advances in neural information processing systems, 2022, 35: 34451-34463.
- [75] Frantar E, Ashkboos S, Hoefler T, et al. Gptq: Accurate post-training quantization for generative pre-trained transformers[J]. arXiv preprint arXiv:2210.17323, 2022.
- [76] Ding X, Zhang X, Han J, et al. Diverse branch block: Building a convolution as an inception-like unit[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021: 10886-10895.
- [77] Ding X, Zhang X, Ma N, et al. Repvgg: Making vgg-style convnets great again[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021: 13733-13742.
- [78] Ding X, Hao T, Tan J, et al. Resrep: Lossless cnn pruning via decoupling remembering and forgetting[C]. Proceedings of the IEEE/CVF international conference on computer vision, 2021: 4510-4520.
- [79] Ding X, Chen H, Zhang X, et al. Repmlpnet: Hierarchical vision mlp with re-parameterized locality[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022: 578-587.
- [80] Liu H, Simonyan K, Yang Y. Darts: Differentiable architecture search[J]. arXiv preprint arXiv:1806.09055, 2018.

- [81] Zoph B, Le Q. Neural Architecture Search with reinforcement learning[C]. International Conference on Learning Representations, 2017.
- [82] Tan M, Chen B, Pang R, et al. MnasNet: platform-aware neural architecture search for mobile[C]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019: 2820-2828.
- [83] Cai H, Gan C, Wang T, et al. Once for all: Train one network and specialize it for efficient deployment[C]. International Conference on Learning Representations, 2020.
- [84] Real E, Liang C, So D, et al. AutoML-Zero: Evolving machine learning algorithms from scratch[C]. International Conference on Machine Learning, 2020: 8007-8019.
- [85] Pham H, Guan M, Zoph B, et al. Efficient neural architecture search via parameters sharing[C]. International Conference on Machine Learning, 2018: 4095-4104.
- [86] Wu B, Dai X, Zhang P, et al. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search[C]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019: 10734-10742.
- [87] Pham C, Teterwak P, Nelson S, et al. MixtureGrowth: growing neural networks by recombining learned parameters[C]. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024: 2800-2809.
- [88] Yang L, Lin S, Zhang J, et al. Grown: Grow only when necessary for continual learning[J]. arXiv preprint arXiv:2110.00908, 2021.
- [89] Fernandes F E, Yen G G. Automatic searching and pruning of deep neural networks for medical imaging diagnostic[J]. IEEE Transactions on Neural Networks and Learning Systems, 2020, 32(12): 5664-5674.
- [90] Bethlehem R A, Seidlitz J, White S R, et al. Brain charts for the human lifespan[J]. Nature, 2022, 604(7906): 525-533.
- [91] Sharma N, Jain V, Mishra A. An analysis of convolutional neural networks for image classification[J]. Procedia computer science, 2018, 132: 377-384.
- [92] Liu Z, Mao H, Wu C-Y, et al. A convnet for the 2020s[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022:

11976-11986.

- [93] Lawrance A, Lewis P. An exponential moving-average sequence and point process (EMA1)[J]. Journal of Applied Probability, 1977, 14(1): 98-113.
- [94] Xia M, Gao T, Zeng Z, et al. Sheared LLaMA: Accelerating language model pre-training via structured pruning[C]. The Twelfth International Conference on Learning Representations, 2023.
- [95] Netzer Y, Wang T, Coates A, et al. Reading digits in natural images with unsupervised feature learning[C]. NIPS workshop on deep learning and unsupervised feature learning, 2011: 7.
- [96] Baldominos A, Saez Y, Isasi P. A survey of handwritten character recognition with mnist and emnist[J]. Applied Sciences, 2019, 9(15): 3169.
- [97] Deng J, Dong W, Socher R, et al. Imagenet: A large-scale hierarchical image database[C]. 2009 IEEE conference on computer vision and pattern recognition, 2009: 248-255.
- [98] Maas A, Daly R E, Pham P T, et al. Learning word vectors for sentiment analysis[C]. Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies, 2011: 142-150.
- [99] Wang A, Singh A, Michael J, et al. GLUE: A multi-task benchmark and analysis platform for natural language understanding[C]. Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, 2018: 353-355.
- [100] Paszke A, Gross S, Massa F, et al. Pytorch: An imperative style, high-performance deep learning library[C]. Advances in neural information processing systems, 2019: 8024-8035.
- [101] Liao Z, Quéru V, Nguyen V-T, et al. Can unstructured pruning reduce the depth in deep neural networks?[C]. Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023: 1402-1406.
- [102] Sun M, Liu Z, Bair A, et al. A simple and effective pruning approach for large language models[C]. The Twelfth International Conference on Learning Representations, 2023.
- [103] Hassibi B, Stork D G, Wolff G J. Optimal brain surgeon and general network pruning[C]. IEEE international conference on neural networks, 1993: 293-299.