第 5 章

深度神经网络压缩

5.1 神经网络压缩的一般方法

如今，深度神经网络模型需要大量的计算、内存和能量，这在我们需要实时推理或在计算资源有限的嵌入式设备上运行模型的情况下成为瓶颈。能量效率是当前深度神经网络模型的主要关注点。解决此效率的方法之一是减小神经网络的规模。

更大的神经网络模型需要更多的内存引用，也就需要更多的能量。本节探讨在不改变网络结构的前提下减小深度神经网络的规模的方法，即假设神经网络模型已经生成，但通过一些技术来“压缩”神经网络。

具体来说，可以从如下方向着手[40]：

① 剪枝(pruning)；

② 权重共享(weight sharing)；

③ 量化(quantization)；

④ 二值/三值化网络(binary/ternary net)；

⑤ Winograd 卷积(Winograd transformation)。

5.1.1 剪枝

剪枝是一种加速神经网络推理的方法，可以有效地生成尺寸更小、内存效率更高、能量效率更高的模型，并且推理速度更快，而准确率损失最小。

深度学习从神经科学领域汲取了灵感，深度学习中的剪枝也是一个受生物学启发的概念。人类新生儿已经有约 50 万亿突触，到 1 岁时，发育到 1000 万亿突触，但到青少年时期，智力基本发育完成时，突触的数量又下降到 500 万亿，如图 5-1 所示。也就是说，一些突触是无用的，消除它们并不会影响智力。这一过程在所有哺乳动物中都发生，人类在 20 岁左右时才完成这一过程。

神经网络的剪枝试图模拟这个过程。剪枝是指如果权重在阈值之下，则删除此连接，如果某个神经元的连接都被删除，那么它也被删除。

如图 5-2 所示，神经网络通常看起来如左边所示：相邻两层之间的每个神经元都相互连接，这种全连接意味着大量的浮点数乘法运算，理想情况下，如果每个神经元仅连接到其他几个神经元上，就可以节省大量的乘法运算，这称为“稀疏”网络。稀疏模型更易于

图 5-1　人脑突触的剪枝

压缩，我们可以在推理过程中跳过那些被删除的“零”连接，从而减少推理过程的延迟。

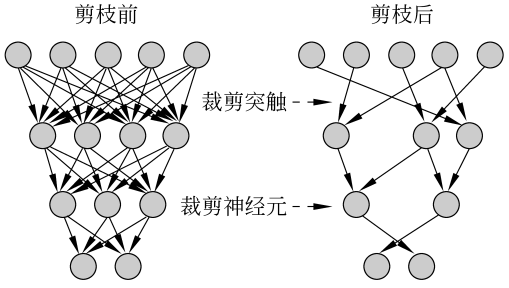


图 5-2　神经网络剪枝

具体来说，根据神经元在神经网络中的贡献程度对其进行排名，然后从网络中删除排名较低的神经元，从而使网络更小、更快。例如，可以根据神经元权重的 L1/L2 范数进行排名。

剪枝后，一般来说神经网络的准确性将下降，这时通过对网络进行多次训练-剪枝-训练-剪枝的过程进行恢复。如果一次剪枝太多，则网络可能损坏得太多，无法恢复。因此，在实践中这是一个迭代过程，通常称为“迭代剪枝”。

以 AlexNet 模型为例，通过剪枝算法，卷积层的连接最多减少到 1/3，全连接层则最多减少到 1/10。总体上，连接减少到原来的 12%，如图 5-3 所示。

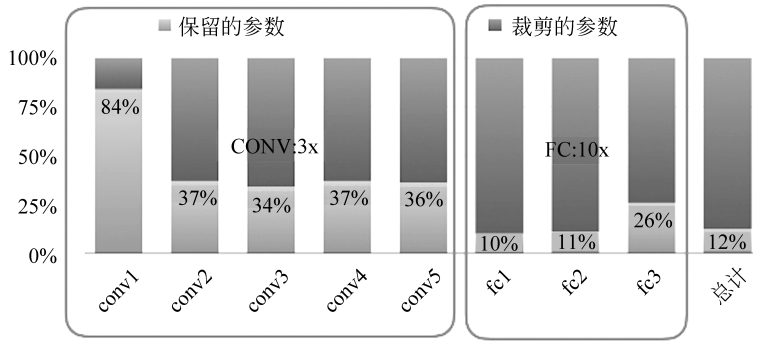


图 5-3　AlexNet 剪枝效率

当然，在剪枝之后，需要重新训练模型来恢复准确性。如图 5-4 所示，在多次迭代之后，剪去 85%以上的连接，而准确率的损失趋于 0。

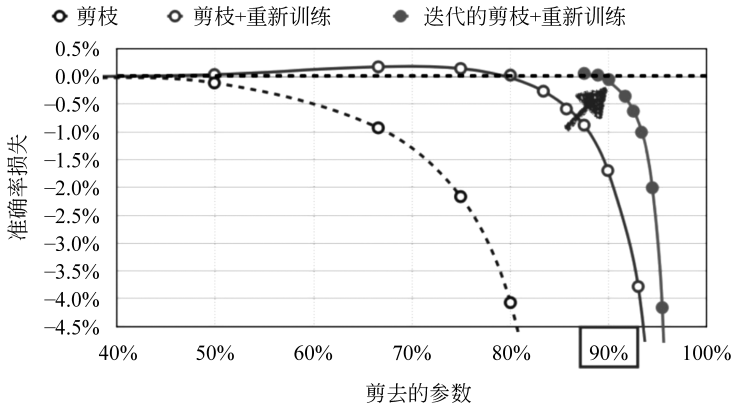


图 5-4　剪枝后重新训练以降低准确率的损失

剪枝带来的效益提升是显著的，以全连接层为例，剪枝可以使推理速度和能量效率大幅提升，如图 5-5 和图 5-6 所示。推理速度提升到 3 倍，而能量效率则提高到 6 倍。

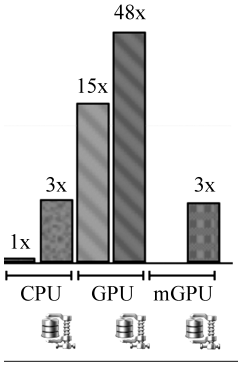


图 5-5　剪枝带来的推理速度提升

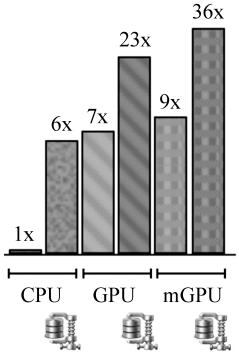


图 5-6　剪枝带来的能量效率提升

观察剪枝前后权重值的分布，就可以理解为什么会发生这样大的提升，如图 5-7 所示。

剪枝前，有大量的趋近于 0 的权值，它们对推理的准确率贡献不大。剪枝后这些权值被直接删除了，但造成准确率下降。通过重新训练，权重的数量没有变化，但它们的值分布更均匀更平滑。

5.1.2　权重共享

权重共享(weight sharing)的原理是，将具有近似值的权重聚类在一起，并存储在字典中，从而减少权重所占用的存储空间。

一种最简单的实现是：将所有权重四舍五入为 256 个级别。通过这种方法，可以将

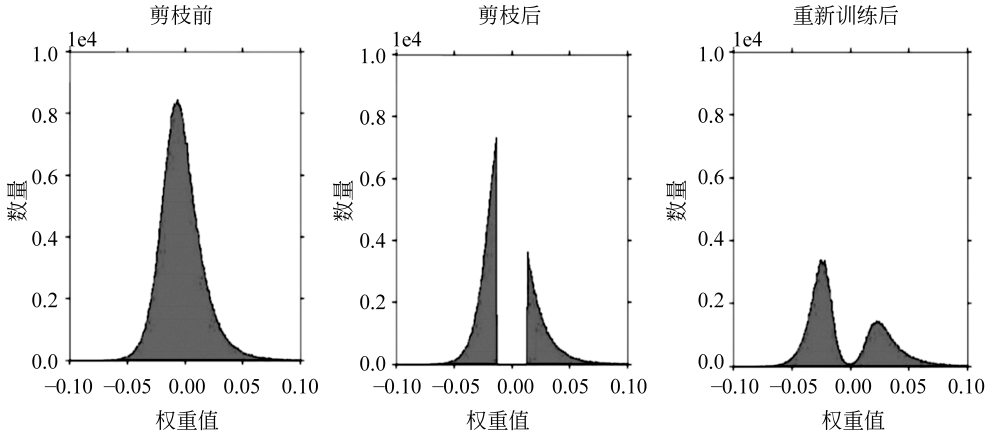


图 5-7　剪枝前后权重值的分布图

一个模型从 87MB 减少到 26MB，准确率仅降低 1%。

完整的方法如图 5-8 所示，2.09，2.12，1.92，1.87 四个权重接近，其质心为 2，它们聚类为同一个权重 2，存储在字典中。类似地，其他权重也进行聚类，最终得到权重字典，它是 1 个浮点数列矩阵，与一个整数矩阵相乘可以近似得到原权重矩阵。相比原来的 4×4 阶的浮点数矩阵，占用的存储空间减小了。

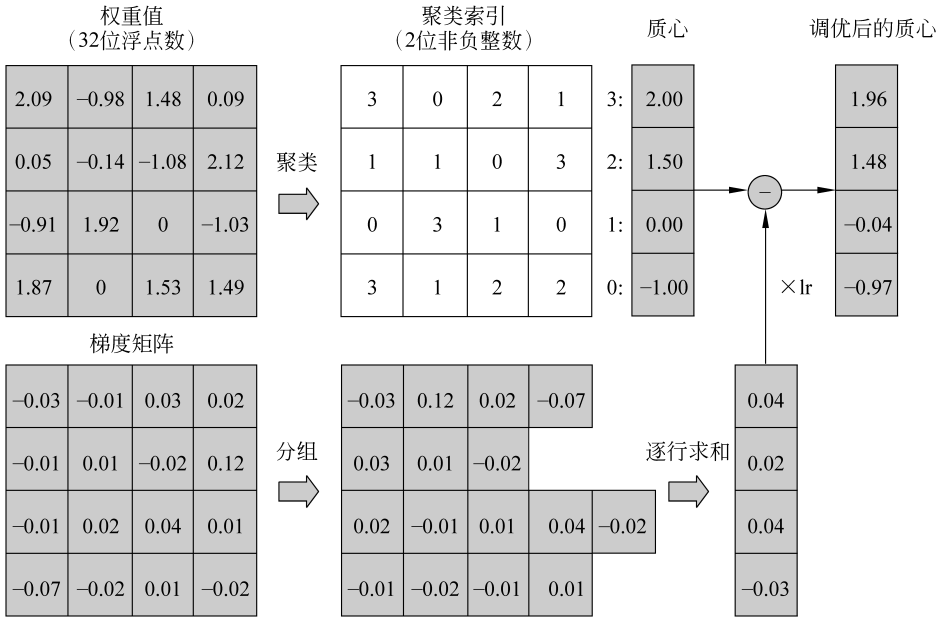
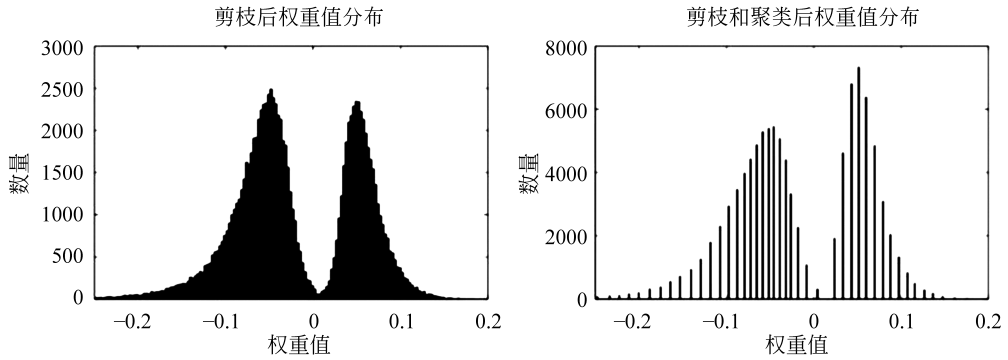


图 5-8　权重共享

同时，梯度矩阵也进行了相应的分组运算，上述 4 个权重对应的梯度分为一组，然后求和，得到 1 个新的梯度列矩阵，它与前面的权重字典合并计算，得到调优后的字典矩阵。

权重共享后的权重值分布图如图 5-9 所示。

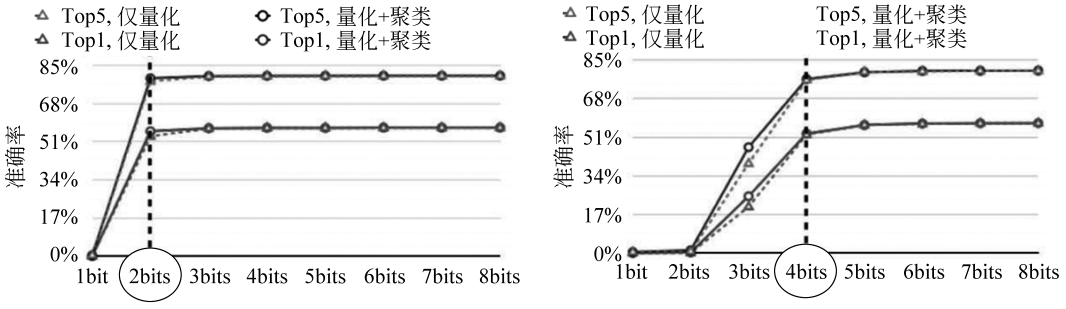


(a) 剪枝后的权重分布　　(b) 进行权重共享后的分布图

图 5-9　权重共享后的权重值分布图

可以看到，值近似的权重聚类为同一权重，原来连续的权重值分布变为离散的分布。

通过以上方法，在准确率没有太大损失的情况下，每权重占用的比特数下降了，全连接层的权重从原来的 32 比特下降到 2 比特，卷积层的权重则从 32 比特下降到 4 比特。如图 5-10 所示。



(a) 所有全连接层中每个有效权重值占用比特数　　(b) 所有卷积层中每个有效权重值占用比特数

图 5-10　每权重比特数下降

权重共享算法对于各种模型都是适用的。例如，对于 AlexNet，模型的大小从 240MB 压缩到 6.9MB，压缩率为原来的 1/35，同时，准确率反而从 80.27% 提高到 80.30%。其他模型的情况类似，如图 5-11 所示。

5.1.3　量化

量化(quantization)的原理是将训练好的模型中的浮点数都量化为定点数，然后进行神经网络的推理。因为定点数运算的效率远高于浮点运算。

权重值和激活值都被量化。具体过程如图 5-12 所示。

- 以浮点数格式训练模型。
- 量化训练好的模型中的权重值和激活值：首先收集权重值和激活值的统计信息，然后选择合适的小数点位置。

网络模型	原始大小	压缩后大小	压缩率	原始准确率	压缩模型准确率
LeNet-300	1070KB	27KB	40x	98.36%	98.42%
LeNet-5	1720KB	44KB	39x	99.20%	99.26%
AlexNet	240MB	6.9MB	35x	80.27%	80.30%
VGGNet	550MB	11.3MB	49x	88.68%	89.09%
GoogLeNet	28MB	2.8MB	10x	88.90%	88.92%
SqueezeNet	4.8MB	0.47MB	10x	80.32%	80.35%

图 5-11　各种模型运用权重共享后的效果

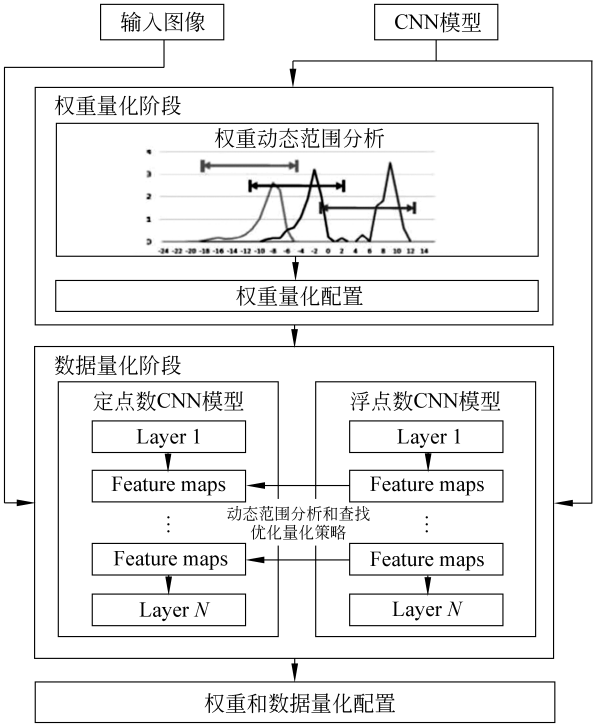


图 5-12　量化过程

- 以浮点数格式微调模型。
- 将权重值和激活值都转化为定点数格式，可以是 32 位、16 位或 8 位定点数。

由于这种方法只是忽略了小数点某一位之后的余数，因此量化后的模型的准确率只有很小的下降。如图 5-13 所示，分别在 GoogLeNet 和 VGG-16 网络上运用量化方法，可以看到，以 32 位定点数量化时准确率基本没有下降，以 16 位定点数量化时准确率略有下降，但可以接受。


扫码看
彩图

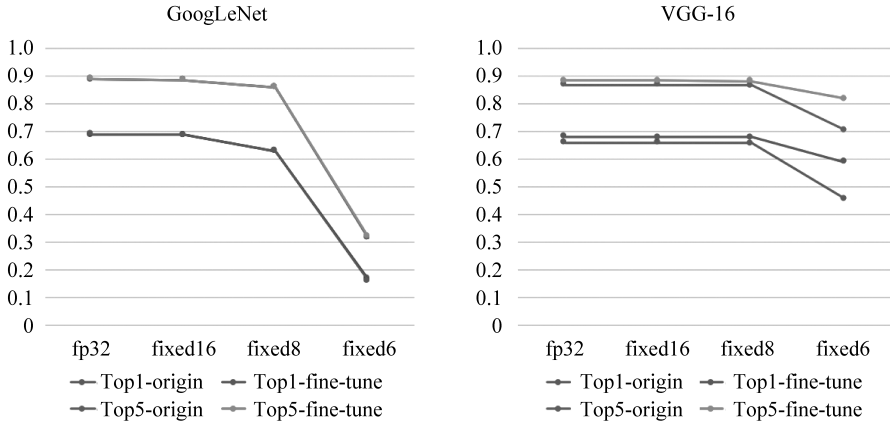


图 5-13　量化后模型的准确率变化

5.1.4　二值/三值化

二值/三值化(binary/ternary)的原理是将权重简化到－1，＋1 两个数值，或者－1，0，＋1 三个数值，这样将大大加速神经网络的推理，因为这时卷积运算只需要通过加法和减法近似实现。三值化权重示意如图 5-14 所示。

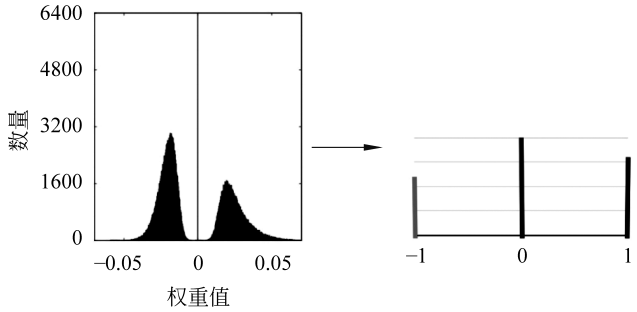


图 5-14　三值化权重示意

具体来说：

- 权重值和输入值均进行二值化或三值化。
- 卷积用二进制点积运算实现，仅有加、减运算。
- 进一步地，点积运算的加减法还可以通过异或和位计数运算实现，从而进一步提高运算效率。

通过将权重二值化，推理时占用的内存减少到原来的 1/32，计算速度提高了 2 倍，而准确率并没有下降。如果进一步将卷积中的加减运算用异或-位计数替代，将输入也进行二值化，则计算速度可以进一步提高到原来的 58 倍，当然，准确率从 56.7％下降到 44.2％，如图 5-15 所示。

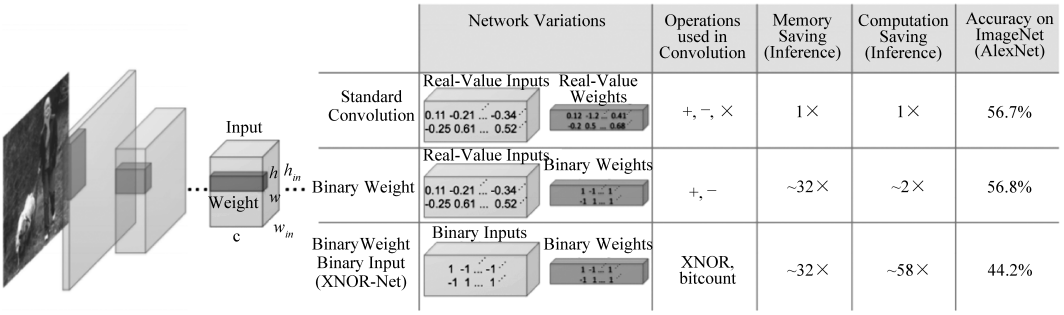


图 5-15　二值化优化结果

5.1.5　Winograd 卷积

Winograd 卷积（Winograd transformation，或称 Winograd 变换）的原理是利用 Winograd 算法，将卷积运算中耗时的乘法运算用不太耗时的加减法运算替代，从而达到减少算法时间度的目的。

Winograd 算法最早于 1980 年由 Shmuel Winograd 在论文 *Arithmetic complexity of computations* 中提出，主要用来减少 FIR 滤波器的计算量。

该算法类似 FFT（快速傅里叶变换），它将数据映射到另一个空间（与 FFT 不同的是，Winograd 算法将数据映射到一个实数空间，而非复数空间）。用加减运算代替部分乘法运算，由于加减运算速度远高于乘法运算，因此达到了明显的加速效果。

比如，直接实现一个有 m 个输出、r 个参数的 FIR 滤波器 $F(m,r)$，一共需要 $m\times r$ 次乘法运算。但使用 Winograd 算法，忽略变换过程的话，仅需要 $m+r-1$ 次乘法运算。

将 Winograd 算法运用到卷积运算的方法如图 5-16 所示，将输入特征图和卷积核均进行 Winograd 变换，然后进行点积（point-wise multiplication）运算，并把多个通道的点积运算结果相加，再进行 Winograd 逆变换，获得最终的输出。

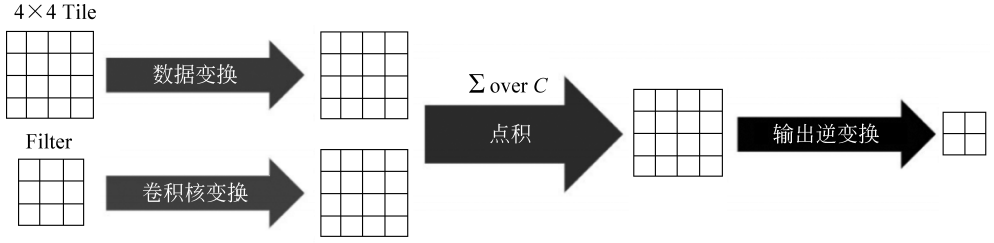


图 5-16　将 Winograd 算法运用到卷积运算的方法

例如，用直接卷积，4 个输出需要 $9\times C\times 4 = 36\times C$ 次乘加运算，其中 C 是通道数。而采用 Winograd 卷积，4 个输出仅需要 $16\times C$ 次乘加运算。

Winograd 卷积通常由硬件实现。

最后，需要指出的是，无论采用哪种方法，神经网络压缩都是一柄双刃剑，压缩率越

高，模型的准确率就越差。运用不同的剪枝率造成不同的稀疏程度，稀疏程度越高，模型尺寸成比例下降，但准确率也会缓慢下降，当稀疏程度达到一定阈值，准确率将断崖式下跌，导致模型不可用。

因此，在运用神经网络压缩的过程中应小心地找到一个合适的平衡点。

5.2 压缩-编译协同设计

压缩-编译(compression-compilation)协同设计[41]的原理是同时对深度神经网络模型进行压缩并对压缩后的模型可执行文件进行编译。这种协同方法可以有效地优化深度神经网络模型的大小和速度，还可以大大缩短压缩过程的调整时间，从而将深度神经网络模型部署在嵌入式设备的主流处理器(如 CPU/GPU)上，并在大多数 AI 应用上实现实时性，而这些 AI 应用原本被认为只有使用特殊的 AI 加速器(如 ASIC/FPGA)才能达到实时运行的效果。其优势如下。

- **成本**：开发专用的 AI 加速器成本高昂，将它们添加到现有系统中会产生额外的费用。
- **技术成熟度**：与通用处理器不同，专用硬件的生产量要小；因此，其生产可用的技术通常比通用处理器落后。例如，目前大多数 AI 加速器都基于 28～65nm CMOS 技术，其晶体管密度显著低于最新的移动 CPU 或 GPU。
- **速度**：由于采用了旧技术，因此专用处理器的运行速度比通用处理器要慢。
- **生态系统**：通用处理器具有完善的生态系统(调试工具、优化工具、安全措施)，这使得高质量应用程序的开发比使用特殊处理器要容易得多。
- **上市时间**：AI 加速器的开发通常需要数年时间。为新开发的硬件加速器创建关联的编译器和系统软件会进一步延长该过程。使用此类硬件的应用程序经常需要使用特殊的 API 并满足许多特殊的约束，这会延长 AI 产品的上市时间。
- **可用性**：由于上述所有原因，使用特殊处理器通常仅限于创建该处理器的公司及其很少的密切客户。结果，为特殊处理器开发的 AI 应用程序仅可以被有限数量的设备所采用。

因此，压缩-编译协同设计的方案是嵌入式人工智能的一种低成本、软件为主的实现方式。下面介绍压缩-编译协同设计软件算法方案的原理。

5.2.1 压缩-编译协同设计的概念

压缩和编译是在通用硬件上适配深度神经网络模型并高效执行的两个关键步骤。模型压缩是减少深度神经网络模型的大小并提高其速度的常用技术，在前面的章节中已经予以阐述，在压缩-编译协同设计中主要使用剪枝和量化两种技术。编译是指从给定的深度神经网络模型生成可执行代码的过程。本质上，编译是将深度神经网络中的高级运算映射到基础硬件支持的低级指令的过程。编译过程在优化代码以有效执行中起着关键作用。

压缩-编译协同设计的原理是同时完成压缩与编译两个组件的设计，这种协同作用体现在以下三个层次上。

- **需求/偏好层次**：在这个层次上，协同作用是指在设计一个组件时考虑另一个组件的偏好或需求。一个例子是，主流处理器通常偏好具有某些计算模式(pattern)的代码。如果模型压缩步骤可以考虑这种偏好，就可以创建一个更适合的模型，使得编译步骤有效地工作。
- **视角/洞察层次**：在这个层次上，协同作用是指在处理其中一个组件的问题时采取另一个组件对该问题的视角或洞察。一个例子就是可组合性或模块化原则，这个原则在保持程序设计系统和编译器均高效且可扩展方面一直发挥着至关重要的作用。
- **方法层次**：在这个层次上，协同作用是指将两个组件的方法紧密集成在一起。例如，编译器可以通过自动生成代码寻找新的深度神经网络剪枝方案，这样可以获得高达 180 倍的加速。

因此，压缩-编译协同系统是压缩器和编译器的有机结合，充分体现了协同设计的原理。它们分别实现了基于模式的 DNN 剪枝和模式感知的代码生成，并通过协同作用生成高效的 DNN 执行代码。

生成器框架概述如图 5-17 所示，具体包括两个组件：

(1) 基于模式的压缩组件，它执行卷积核模式剪枝和连通性剪枝。

(2) 执行代码生成的编译组件，基于模式对压缩后的模型进行多重有效优化。

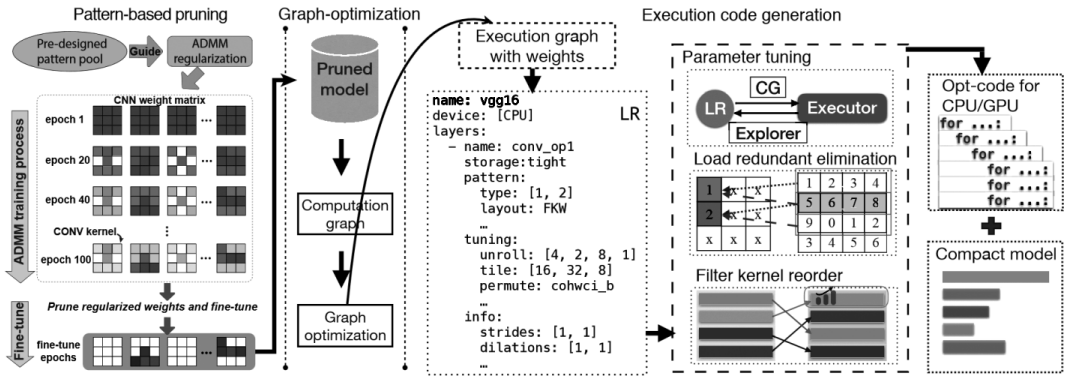


图 5-17　生成器框架概述(来源为 arXiv:2003.06700)

5.2.2　压缩器

压缩器主要采用权重剪枝的方法对模型进行压缩。

权重剪枝是主流的模型压缩技术。但是，现有的剪枝方法要么与现代并行架构不兼容，导致推理延迟长(例如，非结构化的细粒度剪枝)，要么造成严重的准确率下降(例如，结构化的粗粒度剪枝)。

压缩器通过引入粗粒度结构内的细粒度剪枝，实现新的权重剪枝技术。由于细粒度