文章编号:1671-4229(2025)01-0009-12

基于三维分块矩阵的卷积优化算法

吴丰桓, 唐春明
(广州大学 数学与信息科学学院,广东 广州　510006)

摘　要: 卷积是卷积神经网络的关键组成部分,其性能对网络的运行效率具有重要影响。目前对卷积的优化方法集中在计算速度和内存使用两方面。MEC 算法是一种内存高效的卷积加速方法,将输入图像紧凑地排列为二维矩阵,降低中间矩阵的内存开销。然而,在处理大尺寸输入时,生成多个瘦高的二维分块矩阵,无法充分发挥矩阵乘法的峰值性能,导致计算效率下降。文章提出了一种基于三维分块矩阵的卷积优化算法 CMEC:①采用三维窗口在原始图像上滑动获取数据,将输入和卷积核重新组织为三维中间矩阵;②并行计算输入分块矩阵与卷积核的三维矩阵乘法,利用高度优化的矩阵加速库提升计算速度;③将计算结果转换为标准的卷积输出形式。实验结果表明,与 MEC 算法相比,CMEC 算法具有相同的中间矩阵内存使用,但是在 CPU 上计算单个卷积层的平均性能提升了 61%,在 GPU 上性能最高提升 71%,在多层卷积神经网络中至少获得 56% 的性能提升。

关键词: 卷积优化; 三维分块矩阵; 数据重排
中图分类号: TP183　　**文献标志码:** A

Convolution optimization algorithm based on 3D block matrices

WU Feng-huan, TANG Chun-ming
(School of Mathematics and Information Science, Guangzhou University, Guangzhou 510006, China)

Abstract: Convolution is the core component of convolutional neural networks, and its performance significantly impacts the network's efficiency. Current convolution optimization methods focus on both computational speed and memory usage. By compactly organizing the input image into two-dimensional matrices, the MEC approach reduces the intermediate matrix's memory overhead and is a memory-efficient convolution acceleration technique. However, In the processing of large-scale inputs, generating multiple tall and narrow two-dimensional block matrices fails to fully exploit the peak performance of matrix multiplication, resulting in decreased computational efficiency. This paper proposes a convolutional optimization algorithm CMEC based on three- dimensional block matrices. First, data is acquired by sliding a three- dimensional window across the original image, and rearranging the input image and kernel into three- dimensional intermediate matrices. Further, the input block matrix and kernel matrix are multiplied in parallel, and a highly optimized matrix acceleration library is utilized to enhance the computational speed. Finally, the computational results are converted to the standard output format. The experimental results show that, compared with the MEC algorithm, the CMEC algorithm has the same memory usage of the intermediate matrix, but achieves an average performance improvement of 61% on the CPU for computing a single convolutional layer, up to 71% on the GPU,

收稿日期: 2024 - 09 - 28; **修回日期:** 2024 - 12 - 02
基金项目: 国家自然科学基金资助项目(12171114)
作者简介: 吴丰桓(2000—), 女, 硕士研究生. E-mail:2112215079@e.gzhu.edu.cn
* 通信作者. E-mail:ctang@gzhu.edu.cn
引文格式: 吴丰桓, 唐春明. 基于三维分块矩阵的卷积优化算法[J]. 广州大学学报(自然科学版),2025,24(1):9-20.

and obtains at least 50% performance improvement in the convolutional neural network.

Key words: convolution optimization; three-dimensional block matrices; data rearrangement

卷积是卷积神经网络(Convolutional Neural Network, CNN)的核心组件[1],可学习的卷积核在输入图像上滑动,计算窗口数据的加权和,实现特征提取的功能。在常见的卷积神经网络中,卷积层占据网络模型的主要部分[2]。例如在AlexNet中卷积层的数量占比高达62%。此外,卷积运算占据网络总计算量的90%以上[3]。因此,降低卷积运算的代价对提高卷积神经网络的性能至关重要。

卷积的实现方式大致可以分为两种:直接卷积[4-5]和基于数据转换的卷积[6-8]。直接卷积是指特定大小的窗口在输入图像上依据步长滑动,取小矩阵与卷积核执行内积的过程。直接卷积虽然没有额外的内存开销,但是计算过程访问的数据地址不连续,导致较低的缓存命中率和数据重用率,从而使得计算性能不高[9]。基于im2col数据转换[10]的卷积将输入图像和卷积核扁平化为矩阵,再利用基本线性代数子程序(Basic Linear Algebra Subprograms, BLAS)等高度优化的矩阵加速库计算矩阵乘法。im2col的展开操作可以改善内存访问模式,使得数据在内存中的布局更有利于缓存命中和预读取操作。该方法普遍被应用于主流神经网络计算框架中,如Pytorch[11]、Caffe[12]等。然而,im2col矩阵通常远远大于原图像,导致较高的内存占用和带宽开销。此外,在数据转换过程中,点积窗口的边缘像素被重复复制到展开后的矩阵中,这种冗余存储增加了内存占用,对于大尺寸的输入图像和卷积核内存需求急剧上升。因此,im2col算法在内存受限的设备上实现深层神经网络的性能不高[13]。

为了解决im2col方法的中间矩阵内存占用率高的问题,Minsik等[14]提出了一种内存高效的卷积优化算MEC,该算法以一种紧凑的方式将输入转换为二维矩阵,再并行执行多个分块矩阵乘法以实现卷积。相比于im2col算法,MEC算法生成的中间矩阵更紧凑,极大地降低了中间矩阵的内存占用,在不损失精度的条件下,提高了卷积的计算效率。然而,MEC算法在读取输入图像和卷积核时,获取的小矩阵数据地址不连续,导致访存延时增加[15-16]。此外,在计算中间矩阵乘法时,MEC算法的分块数量取决于输入图像卷积核的尺寸,分块数量越多,获取分块矩阵的延时越大。最后,MEC算法使用多个瘦高的二维矩阵作为中间矩阵乘法的输入,无法充分发挥矩阵乘法的峰值性能。

本文提出了一种内存和计算高效的多通道卷积优化算法(Channel MEC, CMEC)。原始数据布局采用Pytorch默认的通道优先(NCHW)。针对MEC算法在数据重排阶段访存延时高的问题,本文采用三维窗口提取多通道数据,将原始数据重构为三维矩阵,按行读取实现内存的连续访问。进一步地,本文利用中间矩阵分块的思想,将卷积操作转换为多个三维分块矩阵乘法。三维分块矩阵乘法具有更高的计算并行性,可以通过高度优化的矩阵乘法库进一步提高计算速度。最后,由于三维分块矩阵乘法的输出是所有输入在每个通道下的卷积结果,需要通过置换将乘法结果重新组织为标准的卷积输出布局。此外,本文进一步从理论上分析了CMEC算法计算卷积的正确性和算法复杂度。

本文的主要贡献包含以下两个方面:

(1) 基于通道优先的数据布局,提出了内存高效的中间矩阵转换方法。中间矩阵的内存利用率比im2col算法提高了KH(卷积核尺寸)倍。

(2) 利用分块矩阵的思想,将卷积操作转换为计算高效的三维分块矩阵乘法。实验表明,在CPU上计算单层卷积的平均性能提升了61%,在GPU上性能最高提升71%,在多层卷积神经网络中至少获得56%的性能提升。

本文的组织内容:第1节介绍卷积优化的相关方法和研究现状;第2节定义了本文使用的符号,介绍了本文的基准方法im2col和MEC的实现原理;第3节详细阐述了本文提出的多通道卷积优化方法,给出了将乘法结果重塑为卷积输出的实现方法;第4节对不同卷积方法进行实验评估,从单个卷积层和卷积神经网络两个维度比较算法的性能;第5节总结了本文工作和进一步的研究方向。

1 卷积相关工作

直接卷积是最朴素的卷积实现方法,虽然没

有额外的内存开销,但是要在全局内存中不连续地访问输入数据,导致较差的内存局部性。Gural 等[17]提出了一种内存高效的直接卷积方法,在内存受限设备实现高精度分类任务,然而该方法引入额外的计算,导致较高延迟。Georganas 等[18]融合了向量化和高速缓存技术,构造即时代码生成器,动态编译生成特定神经网络下的直接卷积子程序。

im2col 算法是典型的基于数据转换的卷积优化算法,数据转换过程引入了非平凡的内存和计算开销,影响整体卷积效率。Anderson 等[19]采用类似 MEC 的思想,将卷积转换为多个小补丁矩阵的乘法,从而减少中间矩阵的存储开销。Dukhan 等[20]用更小的间接缓冲区代替了 im2col 中间矩阵,避免了 im2col 转换导致的内存开销。

另外,FFT[21-23]和 Winograd[24-26]是卷积神经网络框架中常用的卷积算法,采用数学变换将原始数据映射到另一个空间,从而将卷积运算转变为低计算复杂度的空间域运算,提高卷积计算效率。FFT 算法虽然具有高效的计算复杂度和良好的计算并行性,但是对于小尺寸卷积核,需要使用填充技术扩展至输入图像尺寸大小,会导致额外的内存和计算开销。Winograd 算法对于小尺寸的卷积加速效果更明显,更适合计算资源受限的移动端和嵌入式设备。

2　基础知识

2.1　符号描述

本文所用到的符号及其描述如表 1 所示。卷积运算涉及到的 3 个主要张量数据是输入图像 I、卷积核 F 和输出张量 O,将输入图像和卷积核转换后的三维矩阵分别记为输入矩阵 $\boldsymbol{L}$ 和卷积核矩阵 $\boldsymbol{K}$,卷积步长 $s_h = s_w = 1$,填充值 $pad = 0$(如果没有特殊说明,下文默认步长 $s_h = s_w = 1$,$pad = 0$)。将输入矩阵 $\boldsymbol{L}$ 的分块矩阵与卷积核矩阵 $\boldsymbol{K}$ 的乘法称为分块矩阵乘法。

表 1　符号及描述

Table 1　Notations and descriptions

名称	大小	描述
输入图像 I	(N,C,H,W)	(批次, 输入通道数, 输入图像高度, 输入图像宽度)
卷积核 F	(FN,C,KH,KW)	(卷积核个数, 输入通道数, 卷积核高度, 卷积核宽度)
输出张量 O	(N,FN,OH,OW)	(批次, 卷积核个数, 输出高度, 输出宽度)

2.2　im2col 算法和 MEC 算法

im2col 算法和 MEC 算法在单通道下实现卷积的过程如图 1 所示,在这个例子中输入图像的大小为 $(N,C,H,W)=(1,1,4,4)$,卷积核 F 大小为 $(FN,C,KH,KW)=(1,1,2,2)$,那么卷积输出 O 的大小为 $(N,FN,OH,OW)=(1,1,3,3)$。

im2col 方法计算卷积的过程如图 1(a)所示。①读取点积窗口下的元素为行向量,将输入图像转换为二维中间矩阵。同理,将每个卷积核组织为列向量矩阵。②计算输入矩阵和卷积核矩阵的乘法,再将结果转置,获得卷积输出。im2col 引入了额外的内存空间存储中间矩阵,并且在实际应用中,im2col 矩阵尺寸远大于卷积矩阵,无法发挥通用矩阵乘法(General matrix multiplication, GEMM)的最佳性能。

如图 1(b)所示,MEC 算法实现卷积分为两个步骤:转换为中间矩阵和分块矩阵乘法。

(1)转换为中间矩阵,即使用大小为 $(H,KW)=(4,2)$ 的窗口在输入上扫描,得到小矩阵 $(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C})$,将小矩阵展平为行向量并依次存入大小为 $(N\cdot OW, C\cdot H\cdot KW)=(3,8)$ 的输入矩阵 $\boldsymbol{L}$ 中。卷积核矩阵 $\boldsymbol{K}$ 的大小为 $(C\cdot KH\cdot KW, FN)=(4,1)$。如图 1(b)阴影部分所示,im2col 算法将点积窗口的元素展平并复制为输入矩阵的一行,共需要 9 次读取并复制的操作,而 MEC 只需要 3 次,并且中间矩阵的内存空间节省了 34%。然而,在转换过程中,MEC 算法读取的小矩阵的数据地址不连续,并且当输入图像的尺寸较大且卷积核的尺寸较小时,读取数据的跨度增大,可能会导致缓存命中率下降。王朝闻等[15]将小矩阵读取方式修改为按行读取,见图 1(b)阴影部分。首先读取 $\boldsymbol{A}[0,0{:}2]$,存入 $\boldsymbol{L}[0,0{:}2]$,再按内存顺序读取 $\boldsymbol{B}[0,0{:}2]$,$\boldsymbol{C}[0,0{:}2]$,依次存入输入矩阵 $\boldsymbol{L}$ 的对应位置。本文使用该技术读取数据,以保证内存地址连续,加速访存。

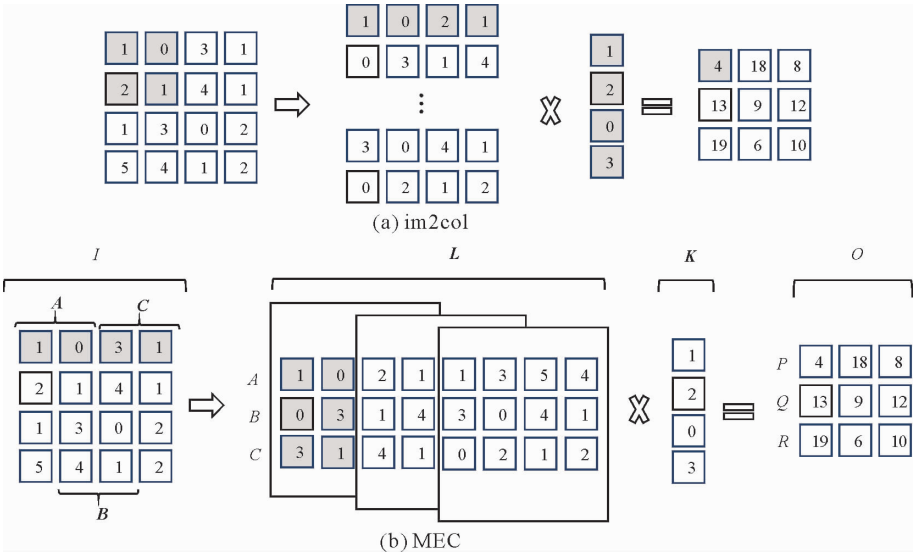

图1 两种方法实现卷积

Fig. 1 Two approaches to implement convolution

(2)分块矩阵乘法,即输入矩阵 $\boldsymbol{L}$ 的每一个分块大小为($N \cdot OW, C \cdot KH \cdot KW$) = (3,4),将分块矩阵($\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_3$)分别与卷积核矩阵 $\boldsymbol{K}$ 做矩阵乘法,而后将乘法结果变形,得到卷积输出 O。im2col 算法直接计算输入矩阵和卷积核矩阵的矩阵乘法,而 MEC 算法将输入矩阵分块,计算分块矩阵与卷积核矩阵的乘法。使用分块矩阵可以充分发挥 GEMM 的性能,进一步增加计算的并行性。本文使用 MEC 算法分块矩阵乘法的思想,获取输入矩阵的三维分块矩阵,并行计算分块矩阵乘法,从而实现卷积过程。

2.3 不同数据布局分析

在深度学习中,数据有两种常用的布局格式:通道优先(NCHW)和通道最后(NHWC)。Pytorch、Caffe 等深度学习框架使用的数据布局为 NCHW,而 tensorflow、OpenCV 等默认使用 NHWC。NCHW 和 NHWC 的内存排布如图2所示。

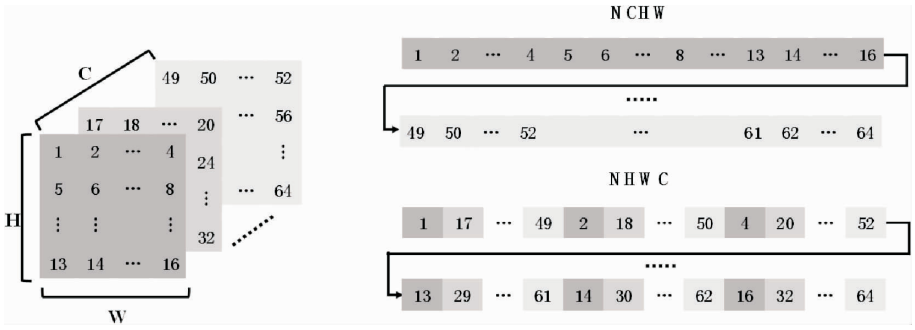

图2 NCHW 和 NHWC 数据布局

Fig. 2 NCHW and NHWC data layouts

在 NCHW 数据布局中,按照 WHCN 的顺序存储数据,相同通道的元素在内存上是顺序排列的。这种内存布局方式在涉及到按通道计算的操作(如池化)时具有一定的优势。此外,NCHW 的内存布局更契合 GPU 的访存模式,访问同一个通道下的像素是连续的,简化了计算的控制逻辑,可以充分利用 GPU 的并行计算能力。NHWC 将不同通道的同一个空间位置的像素排列在一起,再存储下一个像素的所有通道值。这种数据布局更适合对不同通道的像素执行相同计算的操作,例如 1 * 1 卷积。NHWC 数据布局拥有更优的局部访存性能,更适合内存宽带相对较小设备。因此,在不同的计算场景下,选择合适的数据布局,以优化数据的访存和计算性能。因为 NCHW 数据布局更契合 CMEC 算法的中间矩阵读取方式,所以本文选择 Pytorch 默认的 NCHW 为数据的内存布局方

式,而 MEC 算法采用 NHWC 数据布局。

3　多通道卷积优化算法 CMEC

3.1　CMEC 算法实现过程

在深度神经网络中,常常采用多个小卷积核来提取特征。例如 VGG16 采用了大量的 3×3 和 1×1 卷积核,其主要原因包括以下两点:①多个小尺寸卷积核堆叠可以获取与单层大卷积核相同的感受范围,但是前者的参数量和计算量远远小于后者。②多个小尺寸卷积核的堆叠,使得模型获取更大的感受野,提取更多复杂的特征。由于每个卷积操作后都会使用非线性激活函数,从而使得网络引入更多非线性,提高模型的非线性表达能力。当输入通道数很大,并且采用多个卷积核对输入图像进行卷积时,MEC 算法生成的中间矩阵的宽较大,在列方向上的分块矩阵增多。此外,使用 GPU 并行计算分块矩阵乘法时,大量的小规模矩阵占用较高的数据传输成本,并且在并行计算过程资源分配和调度更复杂。因此,本文对 MEC 算法进一步优化,在生成中间矩阵的过程中,修改数据读取的方式,使用三维矩阵代替二维矩阵,将分块矩阵乘法转化为三维分块矩阵乘法,可以有效地降低主机到设备的传输成本,将一部分并行计算合并到三维矩阵乘法中。

CMEC 算法计算多通道卷积的过程如图 3 所示,不同颜色表示不同通道,主要的实现过程分为 3 个步骤:数据重排、三维分块矩阵乘法和数据布局转换。

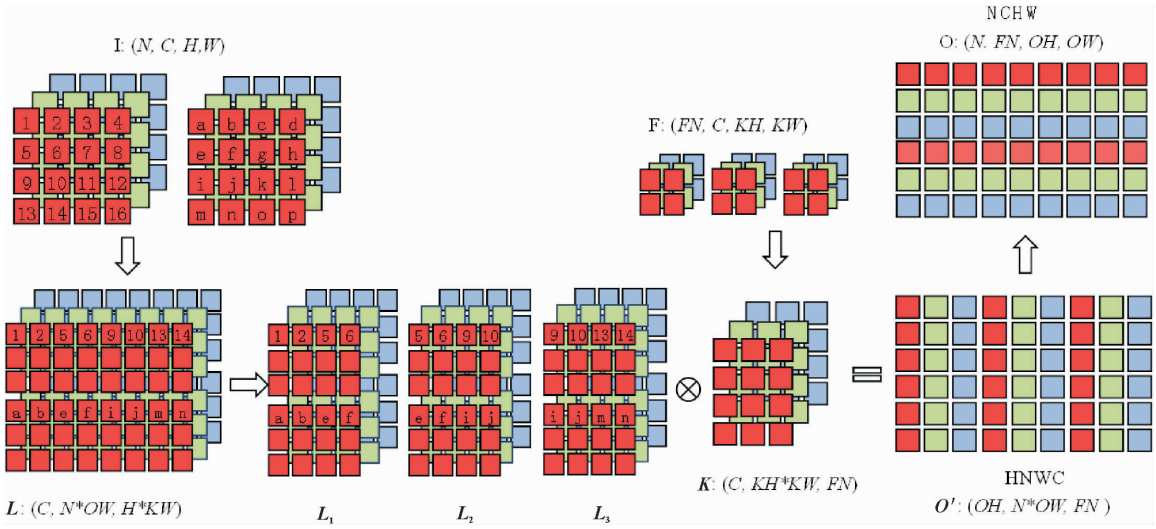

图 3　CMEC 算法实现多通道卷积

Fig. 3　CMEC algorithm for multi-channel convolution

(1)数据重排。数据重排是指重新组织数据,在计算机领域,数据重排通常用于优化内存访问,提高数据访存的局部性和缓存命中率。CMEC 算法采用新的数据组织方式,使用形状为(C,H,KW)的三维窗口在原始数据上扫描,得到三维的小矩阵。使用按行读取技术读取原始数据,存入形状为(C,$N\cdot OW$,$H\cdot KW$)的三维输入矩阵 $\boldsymbol{L}$ 中。同理将卷积核重塑为三维卷积核矩阵 $\boldsymbol{K}$,其形状为(C,$KH\cdot KW$,FN)。

(2)三维分块矩阵乘法。与 MEC 算法不同的是,这里采用三维矩阵乘法替代二维矩阵乘法,计算输入矩阵的三维分块矩阵与卷积核矩阵的乘法。①在输入矩阵 $\boldsymbol{L}$ 中取大小为(C,$N\cdot OW$,$KH\cdot KW$)的三维分块矩阵,并且以 $s_w\cdot KW$ 为步长在 L 上滑动,得到三维分块矩阵($\boldsymbol{L}_1$,$\boldsymbol{L}_2$,$\boldsymbol{L}_3$)。②将三维分块矩阵分别与卷积核矩阵计算三维矩阵乘法,得到大小为(OH,$N\cdot OW$,FN)的输出矩阵 $\boldsymbol{O}'$。需要注意的是,使用直接卷积法计算多通道卷积时,将卷积核作用在对应通道的输入特征图上提取特征。③将所有通道下的计算结果相加,得到一个输入图像和一个卷积核的卷积输出。im2col 算法和 MEC 算法将通道结果相加的操作嵌入矩阵乘法中,直接计算矩阵乘法即可获得卷积输出。然而 CMEC 算法计算三维分块矩阵乘法获得的输出是所有输入在每个通道下的卷积结果,因此,需要再将通道结果相加,获得卷积输出。

三维分块矩阵乘法过程的具体计算公式如公式(1)所示,其中,×为三维矩阵乘法,$SUM(x)$表示将张量x沿着第一个维度求和。

$$\boldsymbol{O}' = [SUM(\boldsymbol{L}_1 \times \boldsymbol{K}), \cdots, SUM(\boldsymbol{L}_3 \times \boldsymbol{K})] \quad (1)$$

(3)数据布局转换。值得注意的是,按通道分块矩阵乘法输出的内存布局是HNWC格式,即它是按照CWNH的顺序读取并存储数据的,因此,需要将HNWC格式的输出数据重新组织,转换为标准的NCHW的数据布局,如图4所示。

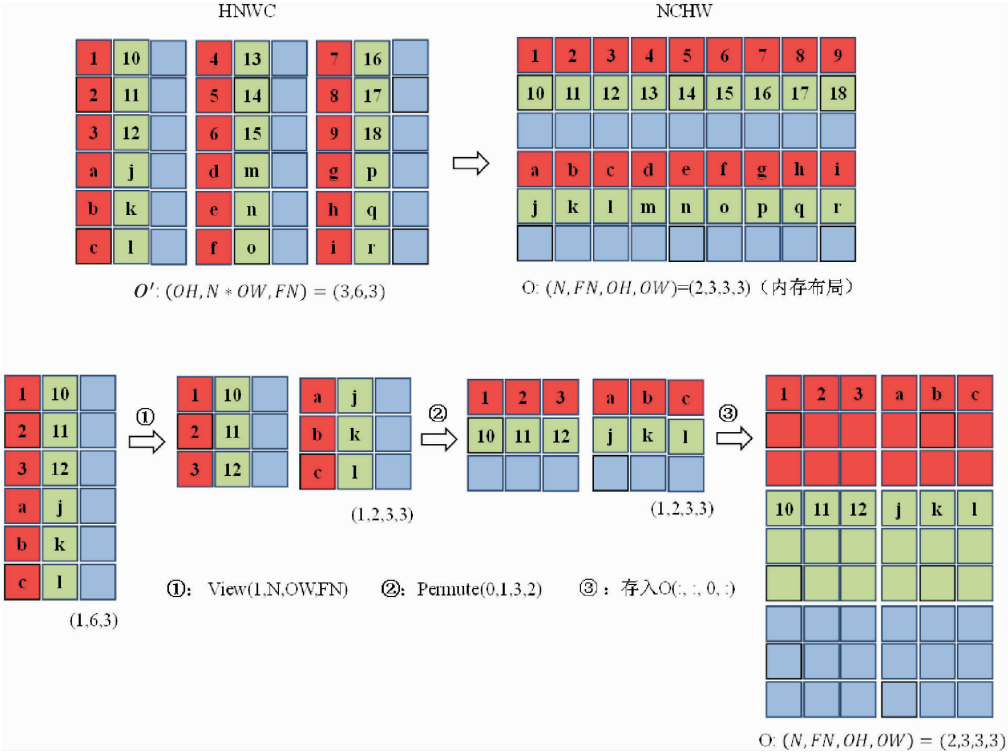

图4 HNWC和NCHW数据布局转换

Fig. 4 Data layout conversion between HNWC and NCHW

如图4可知,已知输入图像的大小为$(N, C, H, W) = (2,3,4,4)$,卷积核的大小为$(FN, C, KH, KW) = (3,3,2,2)$,卷积的输出张量$O$的大小为$(N, FN, OH, OW) = (2,3,3,3)$,输入矩阵$\boldsymbol{L}$的形状为$(C, N \cdot OW, H \cdot KW)$,卷积核矩阵$\boldsymbol{K}$的形状为$(C, KH \cdot KW, FN)$。CMEC算法在三维分块矩阵乘法阶段得到的输出记为矩阵$\boldsymbol{O}'$,$\boldsymbol{O}'$的大小为$(OH, N \cdot OW, FN) = (3,6,3)$,内存布局按照HNWC格式排列。观察其分布可知,$O'$的第一个维度的下的二维矩阵分别对应输出张量$O$的第三个维度($OH$维度)的信息。例如$\boldsymbol{O}'[0]$的数据对应$O[:,:,0,:]$。将HNWC排列的分块矩阵乘法输出转换为NCHW排列的卷积输出O的主要步骤有3步:①将$\boldsymbol{O}'$的第一个维度下的第i个二维矩阵的形状重塑为$(1, N, OW, FN)$,其中,$i \in \{0, 1, \cdots, OH-1\}$;②交换第三和第四维度,得到形状为(1,2,3,3)的张量;③将张量存入输出张量$O[:,:,i,:]$,此时输出张量O的内存布局为NCHW,形状为$(N, FN, OH, OW) = (2,3,3,3)$。在具体实现过程中,采用循环遍历$\boldsymbol{O}'$的第一个维度的数据,在循环的每一次迭代中只处理一个维度的数据,有利于提高数据访问的局部性和缓存命中率,并且$\boldsymbol{O}'$每个维度的数据不存在依赖关系,可以并行处理,以提高程序的效率和性能。然而,当使用多个小卷积核提取大输入图像的特征时,$\boldsymbol{O}'$也随之变大,此时,置换过程会在一定程度上影响程序的效率,相对于CMEC算法整体效率的提升来说,这种影响可以忽略不计。

3.2 正确性分析

不同于以往基于数据转换的卷积优化方法,本文将输入图像和卷积核降维表示为三维矩阵,从而将多通道卷积转变为三维分块矩阵乘法。为了进一步分析CMEC算法计算卷积的正确性,本节给出直接卷积、MEC和CMEC 3种算法计算卷

积输出 O 的一个元素的公式,通过分析 3 个公式在计算数值上的等价性,说明 CMEC 算法三维分块矩阵乘法计算卷积正确性。本节不考虑将计算结果的数据布局转换为标准卷积输出的转置过程,仅从数值上分析计算的正确性。

输入图像 I 的大小为 (N,C,H,W),卷积核 F 的大小为 (FN,C,KH,KW),卷积输出 O 的大小为 (N,FN,OH,OW)。MEC 算法生成的输入矩阵 $\boldsymbol{L}$ 的大小为 $(N\cdot OW,C\cdot H\cdot KW)$,卷积核矩阵 $\boldsymbol{K}$ 的大小为 $(C\cdot KH\cdot KW,FN)$,CMEC 算法生成的输入矩阵 $\boldsymbol{L}$ 的大小为 $(C,N\cdot OW,H\cdot KW)$,卷积核矩阵 $\boldsymbol{K}$ 的大小为 $(C,KH\cdot KW,FN)$。I_i 表示第 i 个输入图像,$I_{i,j}$ 表示第 i 个输入的第 j 个通道下的特征图。$I_{i,j,k,l}$ 表示第 i 个输入的第 j 个通道下索引为 $(k-1,l-1)$ 的元素,其中,维度索引从 1 开始,矩阵索引从 0 开始,$i\in\{1,2,\cdots,N\}$,$j\in\{1,2,\cdots,C\}$,$k\in\{1,2,\cdots,H\}$,$l\in\{1,2,\cdots,W\}$,$m\in\{1,2,\cdots,FN\}$,$n\in\{1,2,\cdots,KH\}$,$p\in\{1,2,\cdots,KW\}$,$u\in\{1,2,\cdots,OH\}$,$v\in\{1,2,\cdots,OW\}$。$\boldsymbol{A}[a,:]$ 表示取矩阵 $\boldsymbol{A}$ 的第一个维度索引为 a 的所有元素,$\boldsymbol{B}[a,b]$ 表示取矩阵 $\boldsymbol{B}$ 的索引为 (a,b) 的元素。$\odot$ 表示卷积,$*$ 表示内积,$\times$ 表示矩阵乘法或元素乘法。使用如上符号,将卷积过程符号化。输入图像和卷积核分别表示为

$$I=[I_1,\cdots,I_i,\cdots,I_N]=[(I_{1,1},\cdots,I_{1,C}),\cdots,(I_{N,1},\cdots,I_{N,C})],\tag{2}$$

$$F=[F_1,\cdots,F_m,\cdots,F_{FN}]=[(F_{1,1},\cdots,F_{1,C}),\cdots,(F_{FN,1},\cdots,F_{FN,C})]。\tag{3}$$

计算单个输入图像和卷积核的卷积公式为

$$I_i\odot F_i=I_{1,1}\odot F_{1,1}+\cdots+I_{1,C}\odot F_{1,C},\tag{4}$$

多通道卷积公式为

$$O=I\odot F=[[I_1\odot F_1,\cdots,I_1\odot F_{FN}],\cdots,[I_N\odot F_1,\cdots,I_N\odot F_{FN}]]。\tag{5}$$

为了方便表示,取定 $(N,C,H,W)=(2,3,4,4)$,$(FN,C,KH,KW)=(2,3,2,2)$,使用 3 种算法计算输出的第一个通道索引为 $(0,0)$ 的元素 $O_{1,1,1,1}$。

(1)直接卷积

$$\begin{aligned}O_{1,1,1,1}&=I_1\odot F_1[0,0]=\\&[I_{1,1}\odot F_{1,1}+I_{1,2}\odot F_{1,2}+I_{1,3}\odot F_{1,3}][0,0]=\\&I_{1,1}[0:2,0:2]*F_{1,1}+I_{1,2}[0:2,0:2]*F_{1,2}+I_{1,3}[0:2,0:2]*F_{1,3}=\\&I_{1,1,1,1}\times F_{1,1,1,1}+I_{1,1,1,2}\times F_{1,1,1,2}+I_{1,1,2,1}\times F_{1,1,2,1}+I_{1,1,2,2}\times F_{1,1,2,2}+I_{1,2,1,1}\times F_{1,2,1,1}+\\&I_{1,2,1,2}\times F_{1,2,1,2}+I_{1,2,2,1}\times F_{1,2,2,1}+I_{1,2,2,2}\times F_{1,2,2,2}+I_{1,3,1,1}\times F_{1,3,1,1}+I_{1,3,1,2}\times F_{1,3,1,2}+\\&I_{1,3,2,1}\times F_{1,3,2,1}+I_{1,3,2,2}\times F_{1,3,2,2}。\end{aligned}\tag{6}$$

(2)MEC 算法

$$\begin{aligned}O_{1,1,1,1}&=I_1\odot F_1[0,0]=\\&\boldsymbol{L}_1[0,:]\times\boldsymbol{K}[:,0]=\\&I_{1,1,1,1}\times F_{1,1,1,1}+I_{1,2,1,1}\times F_{1,2,1,1}+I_{1,3,1,1}\times F_{1,3,1,1}+I_{1,1,1,2}\times F_{1,1,1,2}+I_{1,2,1,2}\times F_{1,2,1,2}+\\&I_{1,3,1,2}\times F_{1,3,1,2}+I_{1,1,2,1}\times F_{1,1,2,1}+I_{1,2,2,1}\times F_{1,2,2,1}+I_{1,3,2,1}\times F_{1,3,2,1}+I_{1,1,2,2}\times F_{1,1,2,2}+\\&I_{1,2,2,2}\times F_{1,2,2,2}+I_{1,3,2,2}\times F_{1,3,2,2}。\end{aligned}\tag{7}$$

(3)CMEC 算法

$$\begin{aligned}O_{1,1,1,1}&=I_1\odot F_1[0,0]=\\&SUM(\boldsymbol{L}_1[:,0,:]\times\boldsymbol{K}[:,:,0])=\\&\boldsymbol{L}_1[0,0,:]\times\boldsymbol{K}[0,:,0]+\boldsymbol{L}_1[1,0,:]\times\boldsymbol{K}[1,:,0]+\boldsymbol{L}_1[2,0,:]\times\boldsymbol{K}[2,:,0]=\\&I_{1,1,1,1}\times F_{1,1,1,1}+I_{1,1,1,2}\times F_{1,1,1,2}+I_{1,1,2,1}\times F_{1,1,2,1}+I_{1,1,2,2}\times F_{1,1,2,2}+I_{1,2,1,1}\times F_{1,2,1,1}+\\&I_{1,2,1,2}\times F_{1,2,1,2}+I_{1,2,2,1}\times F_{1,2,2,1}+I_{1,2,2,2}\times F_{1,2,2,2}+I_{1,3,1,1}\times F_{1,3,1,1}+I_{1,3,1,2}\times F_{1,3,1,2}+\\&I_{1,3,2,1}\times F_{1,3,2,1}+I_{1,3,2,2}\times F_{1,3,2,2}。\end{aligned}\tag{8}$$

进一步地,对于大小为 (N,C,H,W) 的输入图像和大小为 (FN,C,KH,KW) 的卷积核,给出直接卷积、MEC 算法和 CMEC 算法计算卷积输出的任意一点 $O_{i,m,u,v}$ 的公式,$O_{i,m,u,v}$ 表示第 i 个输入和第 m 个卷积核的卷积输出,索引为 $(u-1,v-1)$ 的元素,易知 $(u-1,v-1)$ 为卷积窗口左上角的元素索引。

(1)直接卷积

直接卷积计算每个通道下的点积结果,再将点积结果相加,即可获得卷积输出。对于卷积输出的任意一点 $O_{i,m,u,v}$,直接卷积方法计算 I_i 和 F_m 的卷积,再获取索引为 $(u-1,v-1)$ 的元素。

$$\begin{aligned}O_{i,m,u,v}&=I_i\odot F_m[u-1,v-1]=\\&[I_{i,1}\odot F_{m,1}+\cdots+I_{i,C}\odot F_{m,C}][u-1,v-1]=\\&I_{i,1}[u-1:u-1+KH,v-1:v-1+KW]*F_{m,1}+\cdots+I_{i,C}[u-1:u-1+KH,v-1:v-1+KW]*F_{m,C}=\\&\sum_{p=1}^{KW}(I_{i,1,u,v+p-1}\times F_{m,1,1,p})+\cdots+\\&\sum_{p=1}^{KW}(I_{i,1,u+KH-1,v+p-1}\times F_{m,1,KH,p})+\cdots+\end{aligned}$$

$$\sum_{p=1}^{KW}(I_{i,C,u,v+p-1}\times F_{m,C,1,p})+\cdots+\sum_{p=1}^{KW}(I_{i,C,u+KH-1,v+p-1}\times F_{m,C,KH,p})=\sum_{n=1}^{KH}\sum_{p=1}^{KW}(I_{i,1,u+n-1,v+p-1}\times F_{m,1,n,p})+\cdots+\sum_{n=1}^{KH}\sum_{p=1}^{KW}(I_{i,C,u+n-1,v+p-1}\times F_{m,C,n,p})=\sum_{j=1}^{C}\sum_{n=1}^{KH}\sum_{p=1}^{KW}(I_{i,j,u+n-1,v+p-1}\times F_{m,j,n,p})。\quad(9)$$

(2)MEC 算法

MEC 算法计算多通道卷积过程中,二维分块矩阵 $\boldsymbol{L}_u$ 的索引为 $v-1+(i-1)\cdot OW$ 的行乘以卷积核矩阵的索引为 $m-1$ 列,得到卷积输出的任意元素 $O_{i,m,u,v}$。观察图1(b)可知,分块矩阵与 $\boldsymbol{K}$ 相乘,再经过变换,得到卷积输出在 OH 维度的值。例如 $\boldsymbol{L}_1$ 与 $\boldsymbol{K}$ 计算分块矩阵乘法得到输出 O 的第一行的值。这表明卷积输出的索引 u 决定了取 $\boldsymbol{L}$ 的第几个分块矩阵。$\boldsymbol{L}[v-1+(i-1)\cdot OW,:]$存储了扫描 I_i 得到的第 v 个小矩阵的元素。因此,索引 i 和 v 决定了 $\boldsymbol{L}$ 的第一个维度的索引。索引 m 决定了 $\boldsymbol{K}$ 的第二个维度的索引,表示取第 m 个卷积核参与计算。

$$\begin{aligned}O_{i,m,u,v}&=I_i\odot F_m[u-1,v-1]=\\&\boldsymbol{L}_u[v-1+(i-1)\cdot OW,:]\times\boldsymbol{K}[:,m-1]=\\&I_{i,1,u,v}\times F_{m,1,1,1}+\cdots+I_{i,C,u,v}\times F_{m,C,1,1}+\cdots\\&+I_{i,1,u,v+KW-1}\times F_{m,1,1,KW}+\cdots+I_{i,C,u,v+KW-1}\times\\&F_{m,C,1,KW}+\cdots+I_{i,1,u+KH-1,v}\times F_{m,1,KH,1}+\cdots+\\&I_{i,C,u+KH-1,v}\times F_{m,C,KH,1}+\cdots+I_{i,1,u+KH-1,v+KW-1}\\&\times F_{m,1,KH,KW}+\cdots+I_{i,C,u+KH-1,v+KW-1}\times\\&F_{m,C,KH,KW}=\\&\sum_{j=1}^{C}(I_{i,j,u,v}\times F_{m,j,1,1})+\cdots+\sum_{j=1}^{C}(I_{i,j,u,v+KW-1}\times\\&F_{m,j,1,KW})+\cdots+\sum_{j=1}^{C}(I_{i,j,u+KH-1,v}\times F_{m,j,KH,1})+\\&\cdots+\sum_{j=1}^{C}(I_{i,j,u+KH-1,v+KW-1}\times F_{m,j,KH,KW})=\\&\sum_{p=1}^{KW}\sum_{j=1}^{C}(I_{i,j,u,v+p-1}\times F_{m,j,1,p})+\cdots+\\&\sum_{p=1}^{KW}\sum_{j=1}^{C}(I_{i,j,u+KH-1,v+p-1}\times F_{m,j,KH,p})=\\&\sum_{n=1}^{KH}\sum_{p=1}^{KW}\sum_{j=1}^{C}(I_{i,j,u+n-1,v+p-1}\times F_{m,j,n,p})。\end{aligned}\quad(10)$$

(3)CMEC 算法

使用 CMEC 算法计算多通道卷积的过程中,同 MEC 的分析可知,对于卷积输出的任意元素 $O_{i,m,u,v}$,取输入矩阵 $\boldsymbol{L}$ 的第 u 个三维分块矩,即 $\boldsymbol{L}_u$,再取 $\boldsymbol{L}_u$ 的第二个维度下索引为 $v-1+(i-1)\cdot OW$ 的元素,卷积核矩阵 $\boldsymbol{K}$ 的第三个维度下索引为 $m-1$ 的元素,执行三维分块矩阵乘法,再将每个通道的结果相加,得到卷积输出的任意一个元素 $O_{i,m,u,v}$的值。

$$\begin{aligned}O_{i,m,u,v}&=I_i\odot F_m[u-1,v-1]=\\&SUM(\boldsymbol{L}_u[:,v-1+(i-1)\cdot OW,:]\times\\&\boldsymbol{K}[:,:,m-1])=\\&\boldsymbol{L}_u[0,v-1+(i-1)\cdot OW,:]\times\boldsymbol{K}[0,:,\\&m-1]+\cdots+\boldsymbol{L}_u[C-1,v-1+(i-1)\cdot\\&OW,:]\times\boldsymbol{K}[C-1,:,m-1]=\\&I_{i,1,u,v}\times F_{m,1,1,1}+\cdots+I_{i,1,u,v+KW-1}\times\\&F_{m,1,1,KW}+\cdots+I_{i,1,u+KH-1,v}\times F_{m,1,KH,1}+\cdots+\\&I_{i,1,u+KH-1,v+KW-1}\times F_{m,1,KH,KW}+\cdots+I_{i,C,u,v}\times\\&F_{m,C,1,1}+\cdots+I_{i,C,u,v+KW-1}\times F_{m,C,1,KW}+\cdots+\\&I_{i,C,u+KH-1,v}\times F_{m,C,KH,1}+\cdots+\\&I_{i,C,u+KH-1,v+KW-1}\times F_{m,C,KH,KW}=\\&\sum_{p=1}^{KW}(I_{i,1,u,v+p-1}\times F_{m,1,1,p}+\cdots+I_{i,1,u+KH-1,v+p-1}\times\\&F_{m,1,KH,p})+\cdots+\sum_{p=1}^{KW}(I_{i,C,u,v+p-1}\times F_{m,C,1,p}+\cdots+\\&I_{i,C,u+KH-1,v+p-1}\times F_{m,C,KH,p})=\\&\sum_{n=1}^{KH}\sum_{p=1}^{KW}(I_{i,1,u+n-1,v+p-1}\times F_{m,1,n,p})+\cdots+\\&\sum_{n=1}^{KH}\sum_{p=1}^{KW}(I_{i,C,u+n-1,v+p-1}\times F_{m,C,n,p})=\\&\sum_{j=1}^{C}\sum_{n=1}^{KH}\sum_{p=1}^{KW}(I_{i,j,u+n-1,v+p-1}\times F_{m,j,n,p})。\end{aligned}\quad(11)$$

由公式(6)~(8)可知,直接卷积、MEC 算法和 CMEC 算法计算输出同一个特定的元素,三者的计算公式是等价的。进一步地,由公式(9)~(11)可知,对于卷积输出的任意元素,虽然3种算法数据的组织模式不同,但是计算结果相同,从而说明了 CMEC 算法使用三维分块矩阵乘法计算卷积是正确的。

3.3 复杂度分析

CMEC 算法实现卷积的伪代码如算法1所示。单个卷积层的时间复杂度由卷积的浮点运算次数(Floating-point Operations,FLOPs)决定。在不考虑偏置项的情况下,对于一张输入图像(1,C,H,W),卷积核为(FN,C,KH,KW),此时获得的卷积输出为(1,FN,OH,OW)。计算一次点积所需的浮

点运算次数为 $2 \cdot KH \cdot KW \cdot C - 1$,共需要计算 $OH \cdot OW \cdot FN$ 次点积。卷积过程所需的浮点运算次数为$(2 \cdot KH \cdot KW \cdot C - 1)(OH \cdot OW \cdot FN)$,因此,卷积的时间复杂度为 $O(OH \cdot OW \cdot KH \cdot KW \cdot C \cdot FN)$,可见运算的时间复杂度与输入图像和卷积核的尺寸有关。而 im2col 算法、CMEC 算法和 MEC 算法具有相同的浮点运算次数,因此,两者具有相同的时间复杂度。

3 种不同算法获取的中间矩阵的规模如表 2 所示,输入图像的形状为(N,C,H,W),卷积核的形状为(FN,C,KH,KW)。MEC 算法表明当卷积核的高度大于步长时,即 $KH > s_h$,由于中间矩阵读取 MEC 算法中间矩阵占用的内存比 im2col 算法减少了大约 KH 倍[14]。由表 2 可知,CMEC 算法获得的输入矩阵 $\boldsymbol{L}$ 和卷积核矩阵 $\boldsymbol{K}$ 的规模与 MEC 算法相同。因此,相比于 im2col 算法,CMEC 算法的中间矩阵内存减少了 KH 倍。

算法 1 CMEC 算法计算卷积

输入: Input tensor(N,C,H,W), Kernel tensor (FN,C,KH,KW), stride(s_h,s_w)

$O \leftarrow tensor(N,FN,OH,OW)$;
$\boldsymbol{L} \leftarrow tensor(C,N \cdot OW,H \cdot KW)$;
for $n = 1$ to N do
　for $w = 1$ to OW do
　$\boldsymbol{L}[:,w+n \cdot OW,:] \leftarrow Input[n,:,:,w \cdot s_w + KW].reshape(C,-1)$
　end for
end for
$\boldsymbol{K} \leftarrow Kernel.reshape(C,KH \cdot KW,FN)$
for $h = 1$ to OH do
　$O[h] \leftarrow SUM(\boldsymbol{L}[:,:,h \cdot KW \cdot s_w : h \cdot KW \cdot s_w + KH \cdot KW] \times \boldsymbol{K}).view(1,N,OW,FN).permute(0,1,3,2)$;
end for

输出:O

表 2　3 种算法的中间矩阵大小

Table 2　Intermediate matrix sizes for the three algorithms

算法	输入矩阵 $\boldsymbol{L}$	卷积核矩阵 $\boldsymbol{K}$	分块矩阵
im2col	$(N \cdot OH \cdot OW, C \cdot KH \cdot KW)$	$(C \cdot KH \cdot KW, FN)$	–
MEC	(N, OW, H, KW, C)	$(C \cdot KH \cdot KW, FN)$	$(N \cdot OW, C \cdot KH \cdot KW)$
CMEC	$(C, N \cdot OW, H \cdot KW)$	$(C, KH \cdot KW, FN)$	$(C, N \cdot OW, KH \cdot KW)$

4　实验与分析

4.1　实验平台与测试集

本文在 CPU 和 GPU 上基于 Pytorch 开源框架实现 CMEC 算法,CPU 型号为 Intel(R) Core(TM) i5 – 8300H CPU2. 30GHz,GPU 为 NVIDIA GeForce RTX 3080,本文使用 Pytorch 提供的 API 计算矩阵乘法,该 API 集成了 BLAS 等线性代数库,可以在 CPU 和 GPU 上便捷高效地计算 CMEC 算法中的分块矩阵乘法。

为了全面验证和比较算法的性能,设置单个卷积层实验和 3 个常见的卷积神经网络推理实验。在单个卷积层实验中,如表 3 所示,选取了 12 个卷积层作为单个卷积层实验的测试集,这些卷积层都取自常用的神经网络。取 im2col 方法和 MEC 算法作为实验的比较对象,测试 3 种算法在 CPU 和 GPU 上计算单个卷积层的效率。实验方案为在上述实验环境下实现 3 种算法,每种方法运行 10 次,取平均运行时间,再使用公式计算每种算法的运行时间占比。

$$\frac{T_{\text{algorithm}}}{T_{\text{im2col}} + T_{\text{MEC}} + T_{\text{CMEC}}} \times 100\%, \tag{12}$$

其中,algorithm 分别取 im2col、MEC 和 CMEC,$T_{\text{algorithm}}$表示不同的算法的运行时间。

表 3　测试集

Table 3　Test sets

卷积层	输入 (C,H,W)	卷积核 (FN,KH,KW)	步长 (s_h,s_w)
CV1	(3,227,227)	(96,11,11)	4
CV2	(3,231,231)	(96,11,11)	4
CV3	(3,227,227)	(64,7,7)	2
CV4	(64,224,224)	(64,7,7)	2
CV5	(96,24,24)	(256,5,5)	1
CV6	(256,12,12)	(512,3,3)	1
CV7	(3,224,224)	(64,3,3)	1
CV8	(64,112,112)	(128,3,3)	1
CV9	(64,56,56)	(64,3,3)	1
CV10	(128,28,28)	(128,3,3)	1
CV11	(256,14,14)	(256,3,3)	1
CV12	(512,7,7)	(512,3,3)	1

在卷积神经网络推理实验中,随机选取 ImageNet 测试集中的2张图片为测试序列,再取 Pytorch 的预训练模型 AlexNet、ResNet18 和 DenseNet121,将3个预训练网络模型中的所有卷积层分别使用 MEC 算法和 CMEC 算法实现,测试替换后的模型向前推理所耗费的时间,从而分析 CMEC 算法在模型中的性能表现。公式(13)计算相比于 MEC 算法的推理速度提升,

$$\frac{T'_{\mathrm{MEC}} - T'_{\mathrm{CMEC}}}{T'_{\mathrm{MEC}}} \times 100\%, \tag{13}$$

其中,T'_{MEC}表示 MEC 算法替换卷积层后神经网络推理一张图片耗费的时间,T'_{CMEC}表示 CMEC 算法替换卷积层后神经网络推理一张图片耗费的时间。

王朝闻等[15]提出一种基于 TVM 平台实现的优化算法,TVM 编译器[27]是一种开源端到端的深度学习编译器,支持在 CPU、ARM 等多种硬件架构上部署和优化深度学习模型,内置自动优化算法,可以根据目标硬件和性能需求自动调整模型,以获得最佳性能。由于 TVM 平台的特殊性,本文不与文献[15]中的优化算法进行对比。

4.2 中间矩阵的效率分析

使用表3所示的12个常见的卷积层作为测试集,批次 $batch = N = 1$,使用 im2col、MEC 和 CMEC 算法获取输入矩阵,每个测试集在 CPU 上运行10次,获取中间矩阵转换的平均运行时间。3种算法在中间矩阵转换过程的开销占比如表4所示。由表4可知,在中间矩阵转换过程中,相较于 im2col 算法,CMEC 算法的中间矩阵转换性能平均提升了93%,相较于 MEC 算法,CMEC 算法的中间矩阵转换速率平均提升88%,其中,测试序列 CV1~CV3、CV7~CV8 的提升效果最明显。在这些测试序列中,输入图像的尺寸较大,那么 MEC 算法在 H 维度下一次需要读取的原图像像素数量较多,从而增加了访存用时。

表4 3种算法在中间矩阵转换过程的开销占比

Table 4 The overhead percentage of the three algorithms for the intermediate matrix transformations

算法	卷积层测试序列											
	CV1	CV2	CV3	CV4	CV5	CV6	CV7	CV8	CV9	CV10	CV11	CV12
im2col	0.039 1	0.041 9	0.123 5	0.300 6	0.011 9	0.003 6	0.462 2	0.237 8	0.059 7	0.019 0	0.003 8	0.000 8
MEC	0.133 2	0.186 4	0.253 3	0.246 7	0.009 2	0.002 0	0.400 4	0.191 7	0.050 3	0.012 2	0.001 7	0.000 3
CMEC	0.001 8	0.000 9	0.005 2	0.017 4	0.000 9	0.000 4	0.007 1	0.007 0	0.002 3	0.001 0	0.000 3	0.000 1

4.3 CPU 上单层卷积效率分析

取卷积输入的批次为 $batch = N = 1$,在 CPU 上使用3种算法计算单个卷积层,单位为秒。CPU 上3种算法计算卷积的开销占比如表5所示。由表5可知,相比于 im2col 算法,CMEC 算法计算卷积的平均性能提升了21%,相比于 MEC 算法,卷积的平均性能提升了61%。特别地,对于小通道、大尺寸的输入图像,例如 CV1~CV4、CV7~CV9,CMEC 算法计算卷积的效率至少可获得70%以上的提升,这表明 CMEC 算法更适合处理小通道、大尺寸的图像。在三维分块矩阵乘法过程中,CMEC 算法需要在输入通道维度上分割,获取大小为(C, $N \cdot OW$, $KH \cdot KW$)的分块矩阵,输入通道数 C 越小,获取分块矩阵越快,三维矩阵乘法计算越高效。相反地,对于大通道的小尺寸输入图像,例如 CV6、CV11 和 CV12,CMEC 算法的性能提升不明显,甚至低于 im2col 算法和 MEC 算法。

表5 CPU 上3种算法计算卷积的开销占比

Table 5 The overhead percentage for three algorithms to compute convolution on CPU

算法	卷积层测试序列											
	CV1	CV2	CV3	CV4	CV5	CV6	CV7	CV8	CV9	CV10	CV11	CV12
im2col	0.064	0.051	0.178	0.293	0.009	0.003	0.594	0.208	0.051	0.014	0.004	0.002
MEC	0.183	0.147	0.324	0.355	0.009	0.006	0.476	0.149	0.040	0.013	0.005	0.008
CMEC	0.008	0.00	0.012	0.036	0.005	0.006	0.031	0.039	0.007	0.005	0.003	0.007

4.4 GPU 上3种算法计算卷积的开销占比

为了更好地测试 GPU 上算法的性能,取输入图像的批次 $batch = N = 32$。利用统一计算设备架构(Compute Unified Device Architecture,CUDA)并行实现3种算法,在12个测试序列上获取3种算法计算卷积的所耗的时间,单位为 s。使用 Stream

管理和分配 CUDA 上的资源和任务,更有效地利用 GPU 的并行处理能力。GPU 上 3 种算法计算卷积的开销占比见表 6。由表 6 可知,相比于 im2col 算法,CMEC 算法在 GPU 上计算单层卷积的平均性能提升了 75%,相比于 MEC 算法,CMEC 算法的性能最高提升 71%。MEC 算法和 CMEC 算法具有更高的并行性,能更好地发挥 GPU 的性能,从而拥有比 im2col 更好的效率表现。同样的,当卷积输入的通道数较大时,如 CV5、CV12,CMEC 算法的表现欠佳。但是 CMEC 算法在 GPU 上的总体性能还是优于 im2col 算法和 MEC 算法。

表 6　GPU 上 3 种算法计算卷积的开销占比

Table 6　The overhead percentage for three algorithms to compute convolution on GPU

算法	卷积层测试序列											
	CV1	CV2	CV3	CV4	CV5	CV6	CV7	CV8	CV9	CV10	CV11	CV12
Im2col	0.455	0.200	0.448	2.398	0.100	0.033	1.446	2.471	0.496	0.135	0.035	0.015
MEC	0.084	0.083	0.231	0.345	0.025	0.013	0.682	0.463	0.049	0.022	0.010	0.004
CMEC	0.033	0.034	0.065	0.113	0.027	0.008	0.258	0.184	0.023	0.009	0.006	0.021

4.5　在卷积神经网络模型中的效率分析

表 7　不同网络模型的效率分析

Table 7　Efficiency analysis of different network models

图片	框架	AlexNet /s	ResNet18 /s	DenseNet121 /s
Fig1	MEC	0.196	0.667	1.426
	CMEC	0.036	0.149	0.646
Fig2	MEC	0.192	0.622	1.428
	CMEC	0.030	0.138	0.591
Average Speedup		83%	77%	56%

如表 7 所示,将 3 种预训练模型的卷积层分别替换为 MEC 和 CMEC,统计网络推理一张图片所耗费的时间,单位为 s。相比于 MEC 算法,CMEC 算法在 AlexNet 网络的平均性能提升了 83%,对于更深层的 ResNet18 网络,平均性能提升了 75%,而在密集连接的卷积神经网络 DenseNet121 中,CMEC 算法平均性能提升只有 50%。可见,随着网络深度加深,CMEC 算法的性能提升速度变慢。这是因为 DenseNet101 中包含多个大通道的 1×1 卷积和 3×3 卷积,而 CMEC 算法在计算三维分块矩阵乘法时,多个小尺寸的卷积核意味着输入矩阵的分块矩阵增多,增加了计算量。另外,通道数越大,在 C 方向分割输入矩阵时获取分块矩阵的效率越慢,从而导致算法的性能下降。但是在卷积神经网络模型推理中,CMEC 算法的整体性能还是优于 MEC 算法。

5　总　结

本文提出了一种内存和计算高效的多通道卷积优化算法 CMEC,将输入图像和卷积核重新组织为三维矩阵,计算输入矩阵与卷积核矩阵的三维分块矩阵乘法,最后将计算结果的内存布局转换为默认的 NCHW 格式。在内存开销上,CMEC 算法的中间矩阵占用内存比 im2col 算法减少了 KH 倍。在计算效率上,相较于 MEC 算法,CMEC 算法的中间矩阵转换效率平均提升 88%。在 CPU 上计算单层卷积的平均性能提升了 61%,在 GPU 上性能最高提升 71%,在多层卷积神经网络中至少获得 56% 的性能提升。对小通道、大尺寸输入,CMEC 算法在单个卷积层和常用神经网络中的平均性能表现都优于 MEC 算法。进一步优化 CMEC 算法,提高其处理大通道、小尺寸卷积的效率是未来的研究方向。此外,本文主要针对 NCHW 数据布局下卷积算子的前向传播进行优化,对于其他数据布局以及卷积算子的反向传播、转置卷积等具有有限的适用性。

参考文献:

[1] Khan A, Sohail A, Zahoora U, et al. A survey of the recent architectures of deep convolutional neural networks[J]. Artificial Intelligence Review, 2020, 53(8): 5455-5516.

[2] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[J]. Communications of the ACM, 2012, 60(6): 84-90.

[3] Sze V, Chen Y H, Yang T J, et al. Efficient processing of deep neural networks: A tutorial and survey[J]. Proceedings of the IEEE, 2017, 105(105): 2295-2329.
[4] Mannino M, Peccerillo B, Mondelli A, et al. Analysis and optimization of direct convolution execution on multi-core processors[J]. IEEE Access, 2023, 11(11): 57514-57528.
[5] Santana A D, Armejach A, Casas M. Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, February 25,2023[C]. New York: ACM, 2023.
[6] Lu S, Chu J, Guo LZ, et al. Processing of European Conference on Parallel Processing 2023, August 24,2023[C]. Berlin: Springer, 2023.
[7] Korostelev I. De Carvalho J P L, Moreira J, et al. YaConv: Convolution with low cache footprint[J]. ACM Transactions on Architecture and Code Optimization, 2022, 20(1): 1-18.
[8] Amir O, Ben-Artzi G. Processing of 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 18-24, 2022[C]. Piscataway: IEEE, 2022.
[9] Zhang J Y, Franchetti F, Low T M. Proceedings of the 35th International Conference on Machine Learning, July 10-15, 2018[C]. New York:ACM, 2018.
[10] Vasudevan A, Anderson A, Gregg D. Proceedings of 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors, July 10-12,2017[C]. Piscataway: IEEE, 2017.
[11] Paszke A, Gross S, Massa F, et al. Proceedings of 33rd Conference on Neural Information Processing Systems, May 16-23, 2019[C]. New York: ACM, 2019.
[12] Jia Y Q, Shelhamer E, Donahue J. Proceedings of the 22nd ACM international conference on Multimedia, November 03-07, 2014[C]. New York: ACM, 2014.
[13] Wang Q L, Song Z M, Liu J et al. Proceedings of 2019 International Joint Conference on Neural Networks, July 14-19,2019 [C]. Piscataway: IEEE, 2019.
[14] Minsik C, Brand D. Proceedings of the 34th International Conference on Machine Learning, August 06-11,2017[C]. New York: ACM, 2017.
[15] 王朝闻, 蒋林, 李远成, 等. 基于TVM平台的MEC卷积算法优化[J]. 计算机工程与应用, 2023, 59(1): 180-186.
[16] 方玉玲, 陈庆奎. 基于矩阵转换的卷积计算优化方法[J]. 计算机工程, 2019, 45(7): 217- 221.
[17] Gural A, Boris M. Proceedings of the 36th International Conference on Machine Learning, June 10-15,2019[C]. New York: ACM, 2019.
[18] Georganas E, Avancha S, Banerjee K, et al. Proceedings of SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, November, 11-16,2018[C]. Piscataway: IEEE, 2018.
[19] Anderson A, Vasudevan A, Keane C, et al. Proceedings of 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing, September 09-11,2020[C]. Piscataway: IEEE, 2020.
[20] Dukhan M. Proceedings of 2020 IEEE International Parallel and Distributed Processing Symposium Workshops, May 18-22, 2020[C]. Piscataway: IEEE, 2020.
[21] Mathieu M, Henaff M, LeCun Y. Proceedings of 2nd International Conference on Learning Representations, April 14-16, 2014[C]. New York: ACM, 2013.
[22] Vasilache N, Johnson J, Mathieu M, et al. Proceedings of 3rd International Conference on Learning Representations, May 07-09,2015[C]. New York: ACM, 2015.
[23] Wang Q L, Li D S, Huang X D, et al. Proceedings of 26th International Conference on Parallel and Distributed Computing, August 24-28,2020[C]. New York: ACM, 2020.
[24] Lavin A, Gray S. Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, June 27-30,2016 [C]. Piscataway: IEEE, 2016.
[25] Zhen J, Aleksandar Z, Durand F, et al. Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, February 24-28,2018[C]. New York: ACM, 2018.
[26] Barabasz B, David G. Proceedings of 2019 International Conference of the Italian Association for Artificial Intelligence, November 19-22,2019[C]. Berlin: Springer, 2019.
[27] Chen T Q, Moreau T, Jiang Z H. Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, October 08-10,2018[C]. New York: ACM, 2018.

【责任编辑: 陈 钢】