·人工智能及识别技术· 文章编号：1000-3428(2019)07-0217-05 文献标志码：A 中图分类号：TP391

基于矩阵转换的卷积计算优化方法

方玉玲[a]，陈庆奎[a,b]

（上海理工大学 a. 管理学院；b. 光电信息与计算机工程学院，上海 200093）

摘 要： 提出一种基于矩阵转换的高效卷积计算优化方法 MCFA。根据输出矩阵的宽度和卷积核大小对输入矩阵进行分块，通过 im2col 方法转换输入矩阵子块和核函数矩阵，利用计算统一设备架构中封装的矩阵-矩阵乘法加速库提升卷积计算的速度。在此基础上，将输出子块按序排列，最终得到完整的输出矩阵。实验结果证明，该方法相比 im2col 方法能节省 61.25% 的计算空间，相比 MEC 方法能提高 20.57% 的计算速度，且在分块情况下可以缓解大输入矩阵引起的缓存压力，提高缓存利用率。

关键词： 深度学习；卷积计算；直接卷积；矩阵分块；计算统一设备架构；卷积优化

中文引用格式： 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)：217-221，228.

英文引用格式： FANG Yuling，CHEN Qingkui. Convolution calculation optimization method based on matrix transformation[J]. Computer Engineering，2019，45(7)：217-221，228.

Convolution Calculation Optimization Method Based on Matrix Transformation

FANG Yuling[a]，CHEN Qingkui[a,b]

（a. Business School；b. School of Optical-Electrical and Computer Engineering，
University of Shanghai for Science and Technology，Shanghai 200093，China）

【Abstract】 An efficient convolution calculation optimization method MCFA based on matrix transformation is proposed. The input matrix is divided into blocks according to the width and the convolution core size of the output matrix. The input matrix sub-blocks and the core function matrix are transformed by im2col method. The matrix-matrix multiplication library encapsulated in the Computing Unified Device Architecture（CUDA） is used to speed up the convolution calculation. On this basis，the output sub-blocks are arranged in order，and the complete output matrix is finally obtained. Experimental results show that this method can save 61.25% of the computing space compared with im2col method，improve 20.57% of the computing speed compared with MEC method，and relieve the cathe pressure caused by large input matrix in the case of block，thus improve the cache utilization.

【Key words】 deep learning；convolution calculation；direct convolution；matrix blocking；Computing Unified Device Architecture(CUDA)；convolution optimization

DOI： 10.19678/j.issn.1000-3428.0051507

0 概述

在计算机视觉、模式识别以及人工智能领域，深度学习得到广泛应用，其效果与性能优于传统方法，如 HOG（Histogram of Oriented Gradients）[1]、SIFT（Scale-Invariant Feature Transform）[2]等。在深度学习算法与模型中，卷积神经网络（Convolutional Neural Network，CNN）[3-5]是重要的组成部分。CNN 无需对图像进行复杂的前期预处理而直接输入原始图像，且在如下 3 个方面具有明显优势：局部感受野，有助于神经元从原始图像中提取基本信息，包括边缘和角点信息；权值共享，局部感受野通过权值共享减少了神经网络需要训练的参数数目，在不影响精度的情况下能够简化计算过程；空间或时间子采样，通过子采样可以降低特征映射的分辨率，并输出对移位和失真的敏感度[6]。与传统方法相比，CNN 在实际应用中面临的主要挑战是需要花费较多的检测时间，其中，卷积计算占据很大比例。因此，对 CNN

基金项目： 国家自然科学基金（61572325，60970012）；高等学校博士学科点专项科研博导基金（20113120110008）；上海重点科技攻关项目（14511107902，16DZ1203603）；上海市工程中心建设项目（GCZX14014）；上海智能家居大规模物联共性技术工程中心项目（GCZX14014）；上海市一流学科建设项目（XTKX2012）；沪江基金研究基地专项（C14001）。

作者简介： 方玉玲（1990—），女，博士研究生，主研方向为计算机视觉、并行计算、GPU 集群可靠性分析；陈庆奎，教授、博士、博士生导师。

收稿日期： 2018-05-09 **修回日期：** 2018-06-11 **E-mail：** forwardfyl@163.com

中的卷积计算过程进行优化具有现实意义。

目前,学者们主要从内存使用效率与处理速度 2 个方面对 CNN 进行优化,提出快速 CNN[7]、RCNN[8]等方法。这些方法在很大程度上提高了前馈过程的速度,但是并未解决内存受限的问题。随着深度学习应用范围的扩大,设备的内存受限问题越来越突出,提升卷积中的内存占用效率对于深度学习在各种设备和平台上的应用至关重要。文献[9]提出将输入矩阵转换为列向量组合(im2col)的方法,该方法虽然产生了中间矩阵转换开销,但是转换后的矩阵与核函数卷积可以利用相关加速库(如 cuBLAS、OpenBLAS 中封装的矩阵-向量乘)进行加速。在此基础上,文献[10]提出一种内存高效的卷积优化算法 MEC,该算法根据输出矩阵的行、列对输入矩阵分段读取并将其转换到一个中间矩阵中,然后每次取转换矩阵的一段与核函数转换向量进行卷积。虽然 MEC 算法降低了额外的内存开销,但是分段卷积控制逻辑复杂,且分段矩阵地址不对齐,不利于使用 GPU 进行并行控制。

针对上述问题,本文提出一种基于计算统一设备架构(Computing Unified Device Architecture,CUDA)加速库的卷积计算优化算法 MFCA。根据输出矩阵的大小对输入矩阵进行分块,采用 im2col 方法转换各子块和相应的核函数矩阵,利用 cuBLAS 中矩阵-矩阵乘法加速库对卷积运算实现加速,在此基础上,对输出结果按序存储以得到输出矩阵。

1 相关工作

目前主流的 CNN 及相关计算架构有 Caffe[9]、Theano[11]、cuDNN[12]等。他们使用的卷积计算加速算法可以分为以下 3 种:

1)im2col + GEMM:将输入的原始图像(image)信息转换为多个小的列向量(column)并重组为中间转换矩阵,然后利用高度优化并封装好的 BLAS[13]中的矩阵乘法进行加速。这类方法的典型代表有 OpenBLAS[14]和文档处理[15]中的应用。

2)FFT(Fast Fourier Transform):基于 FFT 的卷积变换主要用于时域和频域之间的转换[16],在频域中卷积可以作为乘法进行计算,但这种计算会因为必须将卷积核填充到与输入图像相同大小而增加内存开销。因此,FFT 不适用于卷积核较小的情况。

3)Winograd:基于 Winograd 的卷积来源于 Coppersmith-Winograd 算法[17]。该方法以增加中间计算开销为代价来减少乘法计算量,目前,其通常与 FFT 相结合被应用于 NNPACK 库中[18]。

2 MFCA 算法

2.1 问题描述

现有典型内存优化的卷积方法包括直接卷积、im2col 卷积和 MEC 卷积,三者的卷积过程如图 1 所示。

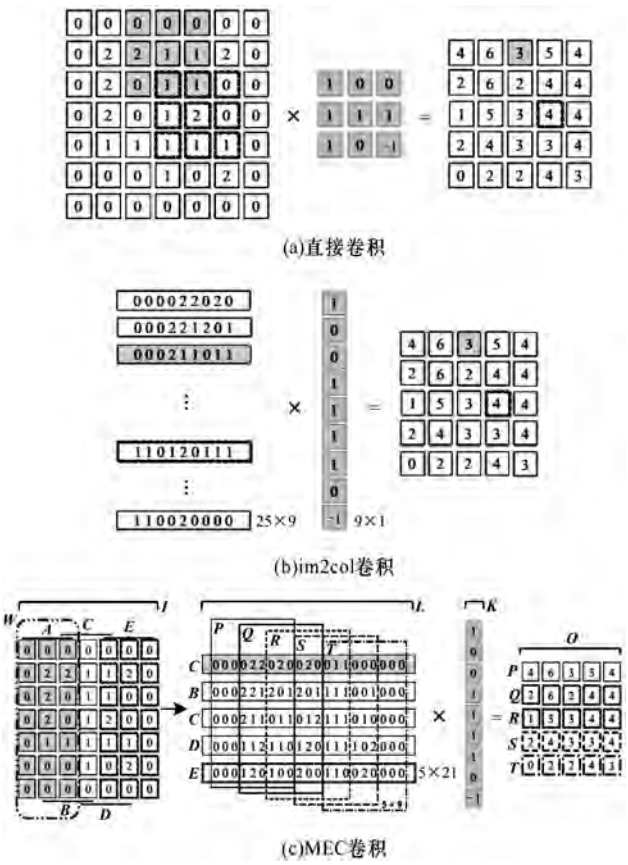

图 1 3 种卷积过程

从图 1 可以看出,在相同输入矩阵 $\boldsymbol{I}(i_h \times i_w \times i_c = 7 \times 7 \times 1)$、卷积核 $\boldsymbol{K}(k_h \times k_w \times k_c = 3 \times 3 \times 1)$、步长 $s_h = s_w = 1$ 且 pad 为 0 时,3 种卷积方法的输出矩阵均为 $\boldsymbol{O}(o_h \times o_w \times o_c = 5 \times 5 \times 1)$。其中,直接卷积根据卷积的定义进行计算,如图 1(a)中灰色部分与卷积核的内积即为输出矩阵的一个元素,随后按步长滑动卷积窗口直至得到整个输出。im2col 卷积方法对输入矩阵和核函数进行转换,得到中间矩阵和核函数的列向量并对两者进行相乘,最终得到相同的输出矩阵。该卷积方法虽然因为中间转换矩阵增加了内存开销,但是其可以利用矩阵向量乘库对卷积进行加速,最终提升计算性能。MEC 方法在 im2col 方法的基础上,按输出矩阵大小分段读取输入矩阵并得到对应的中间矩阵,该方法对计算性能的改进与 im2col 相同。MEC 方法虽然也引入了中间矩阵,但是其比 im2col 节省了 51.2% 的内存空间。但是,MEC 方法也存在不足,即在矩阵转换时矩阵分块逻辑复杂,且得到的小矩阵地址空间不对齐。

2.2 MFCA 算法实现

为进一步提升卷积的计算性能,本文基于 im2col 方法,提出一种节省内存空间并能够利用 cuBLAS[19]中矩阵-矩阵乘法加速库的 MFCA 算法。

该算法对输入矩阵按行读取并转换为中间矩阵的一行,即复制滑动窗口 $\boldsymbol{W}(i_h \times i_w = 3 \times 7)$所覆盖的部分作为中间矩阵 $\boldsymbol{L}$ 的一行,滑动窗口步长设为 1。MFCA 算法具体卷积过程如图 2 所示。

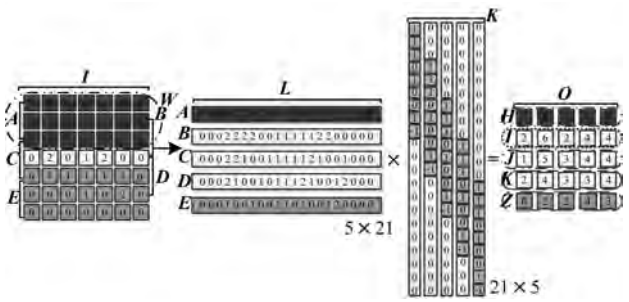

图 2　MFCA 算法卷积过程

在图 2 中,$\boldsymbol{A}$ 是输入矩阵 $\boldsymbol{I}$ 的第 1 部分,$\boldsymbol{A} = \boldsymbol{I}[1:3,1:7]$。将步长 $s_h = 1$ 的滑动窗口 $\boldsymbol{W}$ 所覆盖范围作为第 2 部分,$\boldsymbol{B} = \boldsymbol{I}[2:4,1:7]$。继续滑动子窗口 $\boldsymbol{W}$ 直至覆盖到输入矩阵 $\boldsymbol{I}$ 的最后一行 $\boldsymbol{E} = \boldsymbol{I}[5:7,1:7]$。得到的 5 个子块分别对应中间矩阵 $\boldsymbol{L}$ 的 5 行 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$、$\boldsymbol{D}$、$\boldsymbol{E}$。核函数矩阵 $\boldsymbol{K}$ 中的元素分别与原始卷积中的位置相对应,得到中间矩阵 $\boldsymbol{K}'$。其中,$\boldsymbol{K}'$中非零元素为 $K_W \times K_H \times O_W$ 个,其余均是为利用加速库而填充的 0 值。最终,通过调用 BLAS3 级库中矩阵-矩阵乘接口实现 $\boldsymbol{O} = \boldsymbol{K}' \times \boldsymbol{L}$ 的输出。与 MEC 方法相比,MFCA 的中间转换矩阵较大,但控制逻辑简单,易于 GPU 并行实现,且具有更好的计算性能。

在实际应用中,若输入矩阵为 $\boldsymbol{I}(I_H \times I_W \times I_C)$,卷积核为 $\boldsymbol{K}(K_H \times K_W \times K_C)$,输入矩阵的中间矩阵为 $\boldsymbol{L}(L_H \times L_W \times L_C)$,输出矩阵为 $\boldsymbol{O}(O_H \times O_W \times O_C)$。在使用 MFCA 算法后,他们存在如表 1 所示的关系(在计算过程中不考虑图像通道个数和卷积核个数)。

表 1　各矩阵信息对比

矩阵	行	列	大小
中间矩阵 $\boldsymbol{L}$	O_H	$I_W \times K_H$	$O_H \times I_W \times K_H$
核函数矩阵 $\boldsymbol{K}'$	$I_W \times K_H$	O_W	$I_W \times K_H \times O_W$
输出矩阵 $\boldsymbol{O}$	O_H	O_W	$O_H \times O_W$

在表 1 中,核函数矩阵 $\boldsymbol{K}'$的行(高度)由中间矩阵 $\boldsymbol{L}$ 的列(宽度)$I_W \times K_H$ 决定,但在实际计算中,$\boldsymbol{K}'$只有 $K_H \times K_W \times O_W$ 个非 0 值,其他的 $(I_W - K_W) \times K_H \times O_W$ 个均为 0。非 0 值所占比例越小,计算所需的内存开销越大。因此,为减少 $\boldsymbol{K}'$中值为 0 的元素个数,本文对 MFCA 算法进行改进。

2.3　MFCA 算法优化

将原始输入矩阵 $\boldsymbol{I}$ 按列分为 m 块,分别进行中间矩阵转换,然后与核函数矩阵做卷积,最终得到 m 个输出分量,并按序组成输出矩阵。以图 3 为例,输入矩阵大小为 8×8,在计算过程中将输入矩阵平均分为 2 块,即 $m = 2$,则 2 个分块按序存放得到对应的输出矩阵。

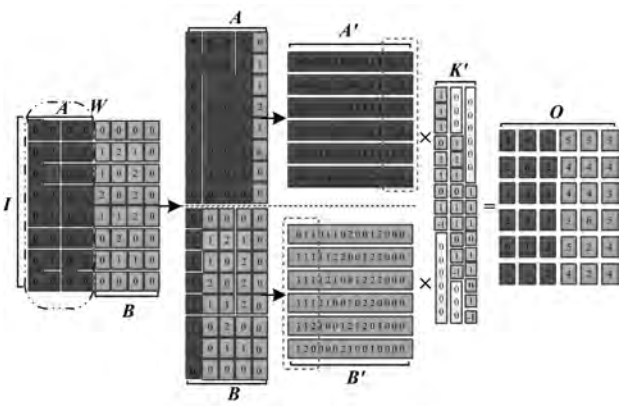

图 3　分块卷积过程

在图 3 中,根据输出矩阵的大小决定分块矩阵是否需要进行填充。若 $I_W/m < O_W/m + K_W - 1$,则需要对子矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 进行填充,填充元素个数为 $n = (O_W/m + K_W - 1) - I_W/m$,填充元素来自与该块相邻的上/下一个块的按列存储的前 n 个元素,例如图 3 中子矩阵 $\boldsymbol{A}$ 中的右侧阴影部分和子矩阵 $\boldsymbol{B}$ 中的左侧阴影部分,则转换矩阵为 $\boldsymbol{A}'$中的右侧虚线框部分和 $\boldsymbol{B}'$中的左侧虚线框部分。2 个子矩阵分别与核函数矩阵进行卷积,得到最终的输出矩阵 $\boldsymbol{O}$。

已知输出矩阵宽度为 O_W,每块的输出宽度为$\dfrac{O_W}{m}$(假设 O_W 能被 m 整除),则分割后的每块输入矩阵宽度为$\dfrac{O_W}{m} + K_W - 1$,对应的中间矩阵的宽度(列)为$\left(\dfrac{O_W}{m} + K_W - 1\right) \times K_H$。各矩阵具体信息如表 2 所示。

表 2　O_W 能被 m 整除时的矩阵信息

矩阵	行	列
中间矩阵 $\boldsymbol{L}$	O_H	$\left(\dfrac{O_W}{m} + K_W - 1\right) \times K_H$
核函数矩阵 $\boldsymbol{K}'$	$\left(\dfrac{O_W}{m} + K_W - 1\right) \times K_H$	$\dfrac{O_W}{m}$
输出矩阵 $\boldsymbol{O}$	O_H	$\dfrac{O_W}{m}$

表 2 显示了被分割后的一块输入矩阵所对应的中间矩阵、核函数矩阵及输出矩阵的行列信息,则中间矩阵 $\boldsymbol{L}$ 的总内存开销为:

$$M_L = \left(\frac{O_W}{m} + K_W - 1\right) \times K_H \times O_H \times m \tag{1}$$

核函数矩阵的内存开销为:

$$M_K = \left(\frac{O_W}{m} + K_W - 1\right) \times K_H \times O_W \tag{2}$$

其中,$\left(\dfrac{O_W}{m} + K_W - 1\right) \times K_H$ 为核函数矩阵的行(高度),

同时也是分块矩阵的列(宽度)。此时,转换矩阵的总内存开销为:

$$M = M_L + M_K = \left(\frac{O_W}{m} + K_W - 1\right) \times K_H \times (m \times O_H + O_W) \quad (3)$$

上述转换需要满足 3 个条件:

1)当 M 最小时确定分块个数 m。

2)$M_{\min} < M_{\text{im2col}} = O_W \times O_H \times K_W \times K_H + K_W \times K_H$。

3)$K_W \leqslant \frac{I_W}{m}$ 且 $1 \leqslant m$。

结合上述条件可得:

$$M_{\text{MFCA}} = \left(\frac{O_W}{m} + K_W - 1\right) \times K_H \times (O_H \times m + O_W) \quad (4)$$

将式(4)对 m 求导可得:

$$M'_{\text{MFCA}} = (K_W - 1) \times K_H \times O_H - \frac{O_W}{m^2} \times K_H \times O_W$$

则有:

$$m = \sqrt{\frac{O_W^2}{(K_W - 1) \times O_H}} \quad (5)$$

得到的 m 值分为 2 种情况:

1)当 m 为整数时,输入矩阵的第 m 个子块矩阵无需进行 0 值填充。

2)当 m 不是整数时,则有 2 个可能值:$\lfloor m \rfloor$,$\lceil m \rceil$,比较他们对应的总内存大小,选择内存开销较小的 m 值,并对第 m 个子块进行 0 值填充。

将得到的 m 值带入式(4),得到最小的总内存开销 $M_{\text{MFCA}_{\min}}$,并与 im2col 和 MEC 的内存开销进行比较。其中,im2col 和 MEC 的总内存开销可分别由式(6)、式(7)计算:

$$M_{\text{im2col}} = O_W \times O_H \times K_W \times K_H + K_W \times K_H \quad (6)$$

$$M_{\text{MEC}} = O_H \times I_H \times K_W + K_H \times K_W \quad (7)$$

比较式(4)、式(6)和式(7)可知,$M_{\text{MCE}} < M_{\text{im2col}}$,$M_{\text{MFCA}_{\min}} < M_{\text{im2col}}$,而 M_{MCE} 与 $M_{\text{MFCA}_{\min}}$ 间的大小关系与需要计算的输入矩阵大小、卷积核尺寸以及 m 值都有关,具体比较结果见 3.2 节。

3 实验结果与分析

本文基于 CUDA 架构对深度学习中的卷积计算进行优化,并通过多组实验来比较分析各算法的性能。

3.1 实验设置

在 CPU + GPU 的实验平台上使用 C ++ 结合 CUDA 架构实现本文算法,CPU 为 Intel i7-4790 CPU@3.60 GHz,GPU 为 NVIDIA GTX 970。其中,由 NVIDIA 提供的封装好的矩阵运算库 cuBLAS 对本文算法的卷积运算进行加速。为更全面地对算法进行验证与比较,在该实验环境下执行 im2col 和 MEC 2 种方法,实验的测试集信息如表 3 所示。

表 3 测试集信息

实验	输入矩阵大小 $I_H \times I_W$	核函数矩阵大小 $K_H \times K_W$
CV1	227 × 227	11 × 11
CV2	227 × 227	7 × 7
CV3	24 × 24	3 × 3
CV4	112 × 112	3 × 3
CV5	56 × 56	3 × 3
CV6	514 × 514	3 × 3
CV7	720 × 480	5 × 5
CV8	1 920 × 1 080	5 × 5
CV9	7 × 7	3 × 3

3.2 内存使用率分析

3 种算法的内存开销比较如表 4 所示。为简化分析,对表 4 中的 M_{im2col}、M_{MEC} 和 M_{MFCA} 内存开销进行柱状图比较,结果如图 4 所示。其中,为方便对比,以 im2col 算法的内存开销为基准。

表 4 3 种卷积算法的内存开销对比

实验	M_{im2col}	M_{MEC}	m 值	M_{MFCA}	最优 m 值
CV1	2 359 305	789 513	16.0	887 808	16
CV2	5 697 890	541 970	4.6	764 794	5
CV3	2 393 258	351 218	6.1	463 842	6
CV4	4 365	1 593	3.3	2 464	3
CV5	108 909	36 969	7.4	46 765	7
CV6	26 253	9 081	5.2	12 441	5
CV7	8 520 425	2 577 625	8.9	1 968 355	9
CV8	51 540 425	18 393 625	12.3	11 271 846	12
CV9	234	114	1.6	202	2

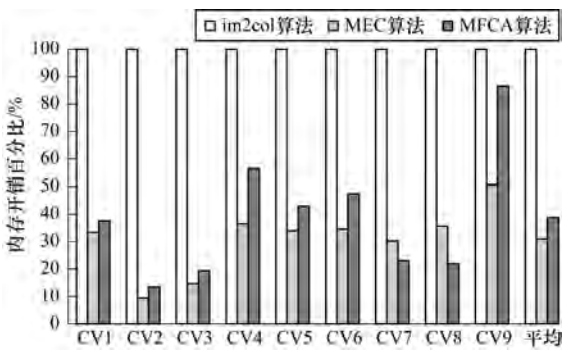

图 4 3 种卷积算法的内存开销柱状图比较

从图 4 可以看出,本文 MFCA 算法产生的中间转换矩阵的内存开销明显优于 im2col 算法,平均内存使用率提升 61.25% 。尤其对于大输入矩阵,如 CV2,效果提升更高,达到 86.57% 。与 MEC 算法相比,本文算法在大输入矩阵中优势明显,如 CV7 和 CV8,在输入矩阵较小时不占优势。

3.3　性能分析

为验证本文算法对卷积计算的性能改进效果,对不同算法的执行时间进行统计,结果如图 5 所示,同样以 im2col 算法的运行时间为基准。

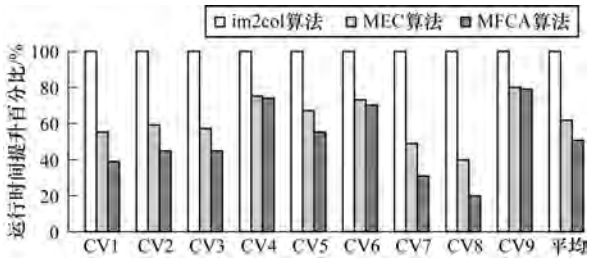

图 5　不同卷积算法的运行时间对比

从图 5 可以看出,当输入矩阵较小时,MFCA 与 MEC 对运行时间性能改进相差不大,但是当输入矩阵较大时,本文算法比 MEC 效果好,情况最佳时其卷积计算速度为 MEC 的 2 倍,如 CV8。在图 5 的 9 组不同实验中,MFCA 比 MEC 的卷积计算速度平均提高 20.57% 。

3.4　算法复杂度分析

离散二维卷积计算公式为:

$$\boldsymbol{B}(i,j)=\sum_{m=0}\sum_{n=0}\boldsymbol{K}(m,n)*\boldsymbol{A}(i-m,j-n) \tag{8}$$

其中,$\boldsymbol{A}$ 为输入矩阵,$\boldsymbol{K}$ 为卷积核,$\boldsymbol{B}$ 为卷积结果,即输出矩阵。卷积计算时间复杂度计算公式为:

$$Time\sim O(\boldsymbol{B}^2\cdot\boldsymbol{K}^2\cdot C_{\text{in}}\cdot C_{\text{out}}) \tag{9}$$

为简化说明,此处统一假设输入矩阵和卷积核都是正方形,C_{in}为每个卷积核通道数,C_{out}为输出通道数。通过分析可知,3 种算法的上述参数完全相同,因此,3 种算法具有相同的时间复杂度。算法所需的存储空间包括 3 个部分:算法本身占用的存储空间,算法的输入输出数据所占用的存储空间,算法运行时临时占用的存储空间。根据 2.3 节相关分析可知,$M_{\text{im2col}}>M_{\text{MEC}}$,而 M_{MEC}与 M_{MFCA}的关系与输入数据大小有关,当输入矩阵较大时,有 $M_{\text{MEC}}>M_{\text{MFCA}}$。且 MFCA 中分块输入矩阵地址空间连续能够提高缓存利用率,同时能够充分利用矩阵乘法库来加速计算过程。因此,MFCA 的空间复杂度较低,im2col 的空间复杂度较高。

综上,本文卷积优化方法 MFCA 在 im2col 的基础上,能够明显提升内存使用率,同时采用分块矩阵的思想能有效降低大矩阵计算对有限的访存资源的压力,提高数据局部性与缓存利用率。此外,MFCA 方法的中间转换过程可以利用 cuBLAS 中的矩阵-矩阵乘法库,进一步提升卷积计算性能,尤其对大输入矩阵,其优势更加明显。

4　结束语

本文提出一种基于矩阵转换的卷积计算优化方法 MFCA。对输入矩阵按列进行分块,采用 im2col 方法对各子块和卷积核矩阵实现转换,以节省内存空间,然后利用矩阵运算库 cuBLAS 对矩阵和矩阵乘的卷积运算进行加速,达到提高卷积计算性能的目的。实验结果验证了 MFCA 方法的有效性。将该方法推广到多通道多区域的卷积应用场景是下一步的研究方向。

参考文献

[1] DALAL N,TRIGGS B. Histograms of oriented gradients for human detection [C]//Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Washington D. C., USA: IEEE Computer Society,2005:886-893.

[2] ZHOU Huiyu,YUAN Yuan,SHI Chunmei. Object tracking using SIFT features and mean shift[J]. Computer Vision and Image Understanding,2009,113(3):345-352.

[3] SHARIF R A,AZIZPOUR H,SULLIVAN J,et al. CNN features off-the-shelf: an astounding baseline for recognition[C]//Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops. Washington D. C.,USA:IEEE Press,2014:156-163.

[4] 王晓晖,盛斌,申瑞民. 基于深度学习的深度图超分辨率采样[J]. 计算机工程,2017,43(11):252-260.

[5] 李传朋,秦品乐,张晋京. 基于深度卷积神经网络的图像去噪研究[J]. 计算机工程,2017,43(3):253-260.

[6] 周飞燕,金林鹏,董军. 卷积神经网络研究综述[J]. 计算机学报,2017,40(6):1229-1251.

[7] YANG Fan, CHOI W, LIN Yuanqing. Exploit all the layers: fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers[C]// Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. Washington D. C., USA: IEEE Press, 2016:236-243.

[8] WANG Xiaolong, SHRIVASTAVA A, GUPTA A. A-fast-RCNN: hard positive generation via adversary for object detection [EB/OL]. [2018-04-29]. https://arxiv.org/pdf/1704.03414.pdf.

[9] JIA Yangqing, SHELHAMER E, DONAHUE J, et al. Caffe: convolutional architecture for fast feature embedding [C]//Proceedings of the 22nd ACM International Conference on Multimedia. New York, USA:ACM Press,2014:269-280.

[10] CHO M, BRAND D. MEC: memory-efficient convolution for deep neural network[EB/OL]. [2018-04-25]. https://arxiv.org/pdf/1706.06873.pdf.

[11] BERGSTRA J, BASTIEN F, BREULEUX O, et al. Theano: deep learning on GPUs with Python[EB/OL]. [2018-04-25]. http://citeseerx.ist.psu.edu/viewdoc/download? doi=10.1.1.678.1889&rep=rep1&type=pdf.

[12] CHETLUR S,WOOLLEY C,VANDERMERSCH P,et al. cuDNN: efficient primitives for deep learning[EB/OL]. [2018-04-25]. https://arxiv.org/pdf/1410.0759.pdf.

[13] JIA Yangqing. Learning semantic image representations at a large scale [EB/OL]. [2018-04-26]. https://cloudfront.escholarship.org/dist/prd/content/qt64c2v6sn/qt64c2v6sn.pdf.

(下转第 228 页)

(上接第 221 页)

[14] ZEE F G V. BLIS: a framework for rapidly instantiating BLAS functionality[J]. ACM Transactions on Mathematical Software, 2013, 41(3): 1-33.

[15] CIRE AN D C, MEIER U, MASCI J, et al. High-performance neural networks for visual object classification[EB/OL]. [2018-04-26]. https://arxiv.org/pdf/1102.0183.pdf.

[16] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[EB/OL]. [2018-04-28]. https://arxiv.org/pdf/1409.1556.pdf.

[17] WINOGRAD S. Arithmetic complexity of computations[M]. [S. l.]: Society for Industrial and Applied Mathematics, 1980.

[18] VASILACHE N, ZINENKO O, THEODORIDIS T, et al. Tensor comprehensions: framework-agnostic high-performance machine learning abstractions[EB/OL]. [2018-04-25]. https://arxiv.org/pdf/1802.04730.pdf.

[19] NVIDIA C. CUBLAS library[EB/OL]. [2018-04-28]. https://arcb.csc.ncsu.edu/~mueller/cluster/nvidia/0.8/NVIDIA_CUBLAS_Library_0.8.pdf.

编辑　吴云芳