计算机工程
Computer Engineering
ISSN 1000-3428,CN 31-1289/TP

《计算机工程》网络首发论文

题目：基于矩阵转换的卷积 CUDA 计算优化方法
作者：方玉玲，陈庆奎
DOI：10.19678/j.issn.1000-3428.0051507
网络首发日期：2018-11-02
引用格式：方玉玲，陈庆奎．基于矩阵转换的卷积 CUDA 计算优化方法[J/OL]．计算机工程. https://doi.org/10.19678/j.issn.1000-3428.0051507

基于矩阵转换的卷积 CUDA 计算优化方法

方玉玲[1]，陈庆奎[1,2]
（1. 上海理工大学 管理学院，上海 200093；2. 上海理工大学 光电信息与计算机工程学院，上海 200093）

摘 要：卷积计算作为深度学习中重要的内容得到了广泛的研究与优化。其中，图像转为列（im2col）的方法应用最为广泛。基于此，本文提出了一种高效的基于矩阵转换的卷积计算优化方法。首先根据输出矩阵宽度和卷积核大小对输入矩阵进行分块，然后通过 im2col 的思想对输入矩阵子块和核函数矩阵进行转换以方便利用计算统一设备架构（Compute Unified Device Architecture，CUDA）中封装好的矩阵-矩阵乘法加速库提升卷积计算速度，最后把输出的子块按序排列得到完整的输出矩阵。理论分析和实验结果证明本文方法既能减少额外的内存开销同时充分利用乘法库加速计算，又能在分块的情况下减少大输入矩阵带来的缓存压力提高缓存利用率。该方法比 im2col 节省了 61.25%的计算空间，比 MEC 方法提高了 20.57%的性能，同时也降低了算法复杂度。

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

A Convolution CUDA Calculation Optimization Method Based on Matrix Transformation

Fang Yuling[1]，Chen Qingkui[1,2]
（1. Business School，University of Shanghai for Science and Technology，Shanghai 200093，China；2. College of Optical-Electrical and Computer Engineering，University of Shanghai for Science and Technology，Shanghai 200093，China）

【Abstract】As an important part of deep learning, convolution calculation has been extensively studied and optimized. Among them, the image-to-column (im2col) convolution method is the most widely used. This paper presents an efficient convolution optimization method based on matrix transformation. First, the input matrix is partitioned according to the width of the output matrix and the size of the convolution kernel. Then the input matrix sub-block and the kernel function matrix are transformed by the idea of im2col, and exploit the matrix-matrix multiplication encapsulated in the Compute Unified Device Architecture (CUDA) to speed up the convolution calculation. Finally, the output matrix is made up of these output sub-blocks. Theoretical analysis and experimental results show that this method can not only reduce the extra memory overhead, but also make full use of the matrix-matrix multiplication library to speed up the calculation. Moreover, it can improve cache utilization in the case of blocking. This method saves 61.25% of the computational space than im2col, and it also improves the computational speed by 20.57% over the MEC. Meanwhile, it also reduces the algorithm complex.

【Key words】deep learning; convolution calculation; direct convolution; matrix blocking; CUDA; convolution optimization
DOI:10.19678/j.issn.1000-3428.0051507

1 概述

为满足计算机视觉，模式识别以及人工智能的发展需求，人们对深度学习的研究也越来越深入。在这些领域中，深度学习方法通常能获得比传统方法，如 HOG（Histogram of Oriented Gradients）[1]，SIFT（Scale-Invariant Feature Transform）[2]更好的效果和性能。在所有深度学习算法和模型中，卷积神经网络（convolutional neural network, CNN）[3]都是必不可少且至关重要的部分，被广泛应用并取得了很好的效果[4][5]。它的优点不仅在于该网络能够避免对图像进行的复杂前

基金项目：国家自然科学基金项目(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，主要研究方向为并行计算、GPU 集群可靠性分析，Email: forwardfyl@163.com；陈庆奎(1966－)，男，博士，教授，博士生导师，主要研究领域为网络计算、物联网技术、并行计算和 GPU 集群，CCF 会员，会员号：E20000931S，Email: chenqingkui@tom.com。

期预处理，直接输入原始图像。同时，它的优势还体现在以下三个方面：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个数，在不影响精度的情况下简化了计算过程；空间或时间子采样，通过子采样能够降低特征映射的分辨率，同时降低输出对移位和失真的敏感度[6]。与传统方法相比，CNN 在实际应用中的挑战主要是需要花费更多的检测时间，而起主导作用的是卷积计算，因此，我们要对 CNN 中的卷积计算过程进行优化。

目前，已有的研究和实现主要从内存使用效率和处理速度提升两个方面对 CNN 进行优化。例如快速 CNN[7]，RCNN[8]，这些方法在很大程度上提高了前馈过程的速度，但是并没有解决内存受限问题。随着深度学习应用的越来越广泛，设备的内存受限问题也越来越突出。因此，提升卷积中的内存占用效率对于在各种设备和平台上的深度学习应用至关重要。文献[9]通过把输入矩阵转换为列向量组合（im2col）的方式虽然引入了多余的中间矩阵转换开销，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 中封装好的矩阵-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（memory-efficient convolution algorithm, MEC），它根据输出矩阵的行、列对输入矩阵分段读取并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做卷积。虽然节省了额外的内存开销，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。

针对上述情况，本文提出一种新的基于 CUDA 加速库的节省内存且计算速度快的卷积算法（memory-saved and fast convolution algorithm, MFCA）。根据输出矩阵大小对输入矩阵进行分块，然后采用 im2col 的思想对各子块和相应的核函数矩阵进行转换，利用 cuBLAS 中矩阵-矩阵乘加速库对卷积运算进行加速，最后对输出结果按序存储得到输出矩阵。实验结果表明本文方法在保证卷积结果正确的情况下，不仅节省了额外的内存转换开销而且能够在很大程度上提升卷积计算性能。

2 相关工作

目前流行的 CNN 及相关计算架构主要但不限于 Caffe[9]，Theano[11]，cuDNN[12]。它们使用的通过对输入矩阵进行优化的卷积加速算法可以总结为以下三种：

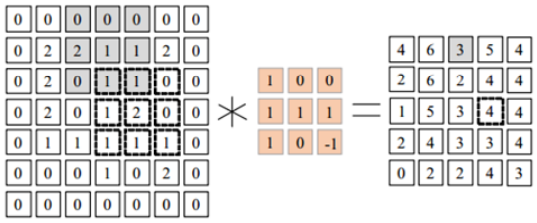
1. im2col+GEMM：把输入的原始图像（image）信息转换为多个小的列向量(column)重组为中间转换矩阵，然后利用高度优化的封装好的 BLAS[13]中矩阵乘法进行加速，例如 OpenBLAS[14]和文档处理[15]中的应用；
2. FFT（Fast Fourier Transform）：基于 FFT 的卷积变换主要用于时域和频域之间的转换[16]，在频域中卷积可以作为乘法进行计算。但是，这种计算会因为必须把卷积核填充到与输入图像相同大小而增加内存开销。因此，不适用于卷积核较小的情况；
3. Winograd：基于 Winograd 的卷积源自于 Coppersmith-Winograd 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，目前结合 FFT 被应用于 NNPACK 库中[18]。

与上述方案相比，本文方法在保证结果准确性的基础上对内存使用和计算速度都进行了优化。

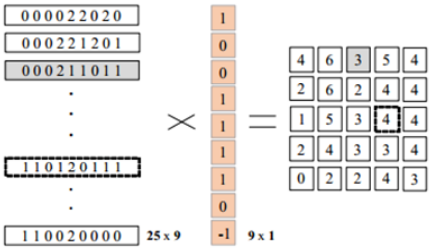
3 算法

3.1 问题描述

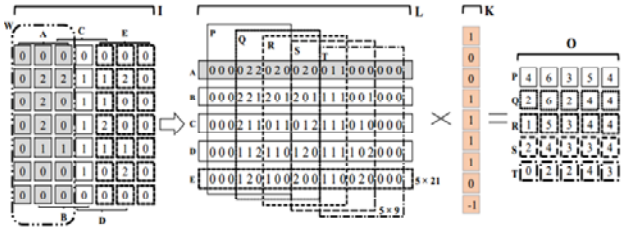
我们首先对现有的内存优化的卷积方法进行简单分析：包括直接卷积、基于 im2col 的卷积方法和 MEC 的卷积方法，如图 1 所示。



（a） 直接卷积



（b） im2col 卷积



（c） MEC 卷积方法

图 1 不同卷积方案

我们以实际应用为例，从图 1 中可看出，对于相同的输入矩阵 $I(i_h \times i_w \times i_c=7\times7\times1)$、卷积核 $K(k_h \times k_w \times k_c=3\times3\times1)$、步长 $s_h = s_w = 1$ 且 pad 为 0 时，三种不同方法的输出矩阵大小均为 $O(o_h \times o_w \times o_c=5\times5\times1)$。其中，直接卷积是根据卷积计算的定义进行计算，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。图 2（b）中 im2col 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相乘最终得到相同的输出矩阵。该方案虽然因为中间转换矩阵增加了内存开销但是可以利用矩阵向量乘库对卷积进行加速，提升计算性能。图 3（b）中 MEC 方法在前一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，该方案对性能的改进与 im2col 效果相同。相比于 im2col 的优点 MEC 虽然也引入了中间矩阵但是其比前者节省了 51.2%的内存空间。同时，该方案也存在不足，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不利于使用 GPU 进行并行控制。

3.2 MFCA 算法概述

为了更好地改进卷积计算性能，本文基于 im2col 的思想提出一种新的节省内存且同时能够利用 cuBLAS[19] 中矩阵-矩阵乘加速库的方法对卷积计算进行优化。本文的 MFCA 方法把输入矩阵按行读取转换为中间矩阵的一行，即复制滑动窗口 $W(i_h \times i_w = 3\times7)$ 所覆盖的部分成为中间矩阵 L 的一行，滑动窗口步长为 1，具体过程如图 2 所示。

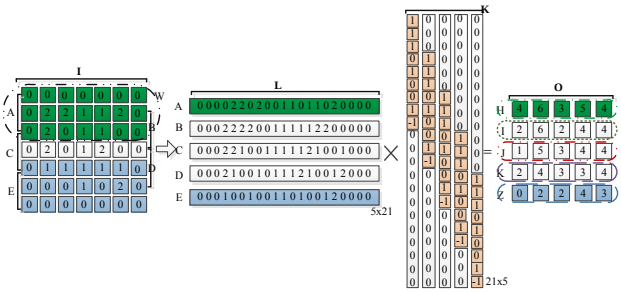


图 2 MFCA 卷积演示

如图所示，A 是输入矩阵 I 的第一部分，$A = I[1{:}3,1{:}7]$。然后，我们以步长 $s_h = 1$ 滑动窗口 W 覆盖范围并形成第二部分 $B = I[2{:}4,1{:}7]$，我们继续滑动子窗口 W 直至覆盖到输入矩阵 I 的最后一行 $E = I[5{:}7,1{:}7]$。得到的五个子块分别对应中间矩阵 L 的五行 A,B,C,D,E。核函数矩阵 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 K'，如图所示。其中，K' 中非零元素为 $K_W \times K_H \times O_W$ 个，其余均是为了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 $O = K' \times L$ 的输出。与 MEC 方法相比，在当前实例中 MFCA 的中间转换矩阵较大，但控制逻辑简单易于 GPU 并行实现而且具有更好的计算性能。

若实际应用中输入矩阵的大小为 $I(I_H \times I_W \times I_C)$，卷积核大小为 $K(K_H \times K_W \times K_C)$，输入矩阵的中间矩阵为 $L(L_H \times L_W \times L_C)$，输出矩阵为 $O(O_H \times O_W \times O_C)$。则使用 MFCA 方法后它们存在如表 1 所示的关系（在计算过程中不考虑图像通道数和卷积和个数）：

表 1 基本信息

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

在表 1 中，核函数矩阵 K' 的行（高度）为中间矩阵 L 的列（宽度）$I_W \times K_H$ 决定，但在实际计算中 K' 只有 $K_H \times K_W \times O_W$ 个非零值，其它的 $(I_W\text{-}K_W) \times K_H \times O_W$ 均为 0，非零值所占比例越小计算所需的内存开销越大。因此，为了减少 K' 中值为 0 的元素个数，我们对 MFCA 方法进行了改性。

3.3 算法优化

我们把原始输入矩阵 I 按列分为 m 块分别进行中间矩阵转换与核函数矩阵进行卷积，最终由得到的 m 个输出分量按序组成输出矩阵。以图 3 为例，输入矩阵为 8×8，假设在计算过程中平均分为两块，即 m=2，则

输出矩阵也为对应的两块按序存放得到。

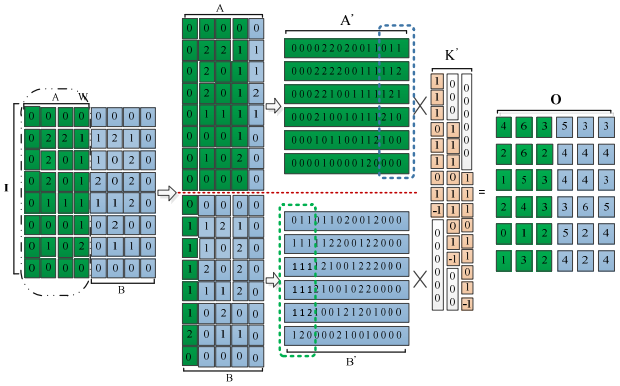


图 3 分块卷积过程

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 A 和 B 进行填充，填充元素个数为 $n=(O_W/m+K_W-1)-I_W/m$。元素来源为与该块相邻的上/下一个块的按列存储的前 n 个元素，例如图中子矩阵 A 中的蓝色部分和子矩阵 B 中的绿色部分，则转换矩阵中为 $A^{'}$ 中的蓝色虚线框和 $B^{'}$ 中的绿色虚线框部分。两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 O。

已知输出矩阵宽度 O_W，每块的输出宽度为 $\frac{O_W}{m}$，则分割后的每块输入矩阵宽度为 $\frac{O_W}{m}+K_W-1$，对应的中间矩阵的宽度（列）为 $\left(\frac{O_W}{m}+K_W-1\right)\cdot K_H$，具体矩阵信息见表 2。

表 2 O_W 能被 m 整除时

矩阵	行	列
中间矩阵	O_H	$\left(\frac{O_W}{m}+K_W-1\right)\times K_H$
核函数矩阵	$\left(\frac{O_W}{m}+K_W-1\right)\times K_H$	$\frac{O_W}{m}$
输出矩阵	O_H	$\frac{O_W}{m}$

表 2 中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 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 $\left(\frac{O_W}{m}+K_W-1\right)\times K_H$ 为核函数矩阵的行（高度），同时也是分块矩阵的列（宽度）。此时，转换矩阵总内存开销为：

$$\begin{aligned}M&=M_L+M_K\\&=\left(\frac{O_W}{m}+K_W-1\right)\times K_H\times(m\times O_H+O_W)\end{aligned} \tag{3}$$

上述转换需要满足三个条件：

1. 当 M 最小时确定分块个数 m；
2. $M_{min}<M_{im2col}=O_W\times O_H\times K_W\times K_H+K_W\times K_H$
3. $K_W\le\frac{I_W}{m}$ 且 $1\le m$。

结合上述条件可得

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

上式对 m 求导可得：

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

则

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

得到的 m 值又分为两种情况：

1. m 为整数时，输入矩阵的第 m 个子块矩阵不需要进行 0 值填充（padding）；
2. m 不是整数时，则有两个可能值：$\lfloor m\rfloor$ 或 $\lceil m\rceil$，比较它们对应的总内存大小，选择内存开销较小的那一个并对第 m 个子块进行 0 值填充 padding。

把得到的 m 带入公式（4）得到最小的总内存开销 $M_{MFCA_{min}}$，并与 im2col 和 MEC 的内存开销进行比较，其中 im2col 的总内存可由公式（6）获得：

$$M_{im2col}=O_W\times O_H\times K_W\times K_H+K_W\times K_H \tag{6}$$

MEC 的总内存由公式（7）获得：

$$M_{MEC}=O_H\times I_H\times K_W+K_H\times K_W \tag{7}$$

比较公式（4）、（6）和（7）可知 $M_{MCE}<M_{im2col}$，$M_{MFCA_{min}}<M_{im2col}$，而 M_{MCE} 与 $M_{MFCA_{min}}$ 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 都有关，具体比较结果见 4.2 节。

4 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，并通过多组实验对本文算法进行验证。

4.1 实验条件

我们在 CPU+GPU 的实验平台上使用 C++结合 CUDA 架构实验本文方法，CPU 为 Intel i7-4790 CPU@3.60GHz，GPU 为 NVIDIA GTX 970。其中 NVIDIA 提供的封装好的矩阵运算库 cuBLAS 对本文方法的卷积运算进行加速。为了更全面的对本文方法进行验证与比较，我们同时也在该实验环境下执行了 im2col 和 MEC 两种方法，实验的测试集如表 3 所示。

表 3 测试集

实验	输入矩阵 $I_H \times I_W$	核函数 $K_H \times K_W$
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	1920×1080	5×5
CV9	7×7	3×3

4.2 内存使用率分析

为比较不同方法对内存的使用情况，我们分别记录使用不同方法时上述 5 组实验的内存开销，如表 4 所示。

表 4 不同卷积方法的内存开销

实验	M_{im2col}	M_{MEC}	m	M_{MFCA}	最优 m 值
CV1	2359305	789513	16	887808	16
CV2	5697890	541970	4.6	764794	5
CV3	2393258	351218	6.1	463842	6
CV4	4365	1593	3.3	2464	3
CV5	108909	36969	7.4	46765	7
CV6	26253	9081	5.2	12441	5
CV7	8520425	2577625	8.9	1968355	9
CV8	51540425	18393625	12.3	11271846	12
CV9	234	114	1.6	202	2

为了简化分析我们对表 4 中的 M_{im2col}、M_{MEC} 和 M_{MFCA} 进行详细比较，如图 4 所示。

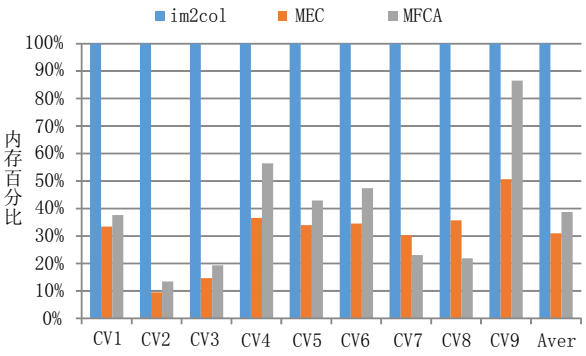


图 4 不同卷积方法的内存开销

图 4 中显示了三种不同方法对对组卷积计算时的内存使用情况，为方便分析，我们以 im2col 的内存开销为基准。从图中可以看到本文方法 MFCA 产生的中间转换矩阵的内存开销明显优于 im2col 的内存开销，平均内存使用效率提升了 61%。尤其是对于大输入矩阵效果更好，如 CV2，节省了 86.57%内存空间。与 MEC 相比本文方法在大输入矩阵中优势突出，例如 CV7 和 CV8，在输入矩阵较小时不占优势。

4.3 性能分析

为了验证本文方法对卷积计算的性能改进效果，我们对不同方法的执行时间也进行了统计，如图 5 所示。

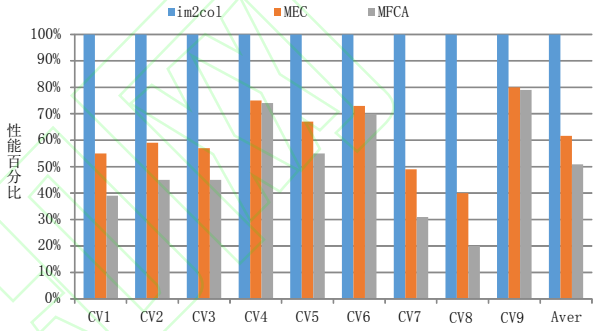


图 5 不同卷积方法的运行时间

图 5 中以 im2col 的计算性能为基准分析本文方法 MFCA 与 MEC 的性能。从图中可以看出针对不同的输入矩阵大小，后两种方法计算性能的提升效果也各不相同。当输入矩阵较小时 MFCA 与 MEC 对性能的改进相差不大，但是当输入矩阵较大时，本文方法比 MEC 效果更好，最高时卷积计算速度为 MEC 的 2 倍，如 CV8 所示。图中 9 组不同的实验中，MFCA 比 MEC 的卷积计算速度平均提高了 20.57%。

4.4 算法复杂度分析

已知离散二维卷积计算公式为：

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

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

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

此处为简化说明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。分析可知三种算法的上述参数完全相同，因此三种算法具有相同的时间复杂度。而三种算法所需要的存储空间包括三个部分：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和算法运行时临时占用的存储空间。根据 3.3 节相关分析可知 $M_{im2col} > M_{MEC}$，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 $M_{MEC} > M_{MFCA}$。而且 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加速计算过程。因此，MFCA 的空间复杂度低，im2col 的空间复杂度相对较高。

综上可得，本文提出的卷积优化方法 MFCA 在 im2col 的基础上能很大程度的提升内存使用效率，同时采用分块矩阵的思想能有效降低大矩阵计算时对有限的访存资源的压力，提高数据局部性以及缓存利用率。而且本文的中间转换过程可以利用 NVIDIA 提供的 cuBLAS 中的矩阵-矩阵乘法库进一步提升卷积计算性能。尤其是对大输入矩阵来说该方法的优势更加明显。

结束语

随着深度学习在模式识别、目标检测等领域的广泛应用，作为深度学习中重要组成部分的卷积计算的相关研究也越来越多。对卷积的优化主要集中在两个方面：减少卷积计算时内存开销和提高卷积计算速度。目前常用的优化方法是基于 Caffe 源码的把输入矩阵转为由多个列向量组成的矩阵（im2col）的方法，通过利用矩阵-向量乘进行卷积加速，但同时也牺牲了部分存储空间。本文对输入矩阵按列进行分块，然后采用 im2col 的思想对各子块和卷积核矩阵进行转换节省内存空间，最后利用 NVIDIA 提供的矩阵运算库 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，实验证明了本文方法的有效性。不足之处在于我们虽然提供了一种快速的卷积优化方法，但是还未推广到多通道多区域的卷积应用场景，这是我们下一步的研究内容。

参考文献：

[1] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]// IEEE Computer Society Conference on Computer Vision & Pattern Recognition. IEEE Computer Society, 2005:886-893.

[2] Zhou H, Yuan Y, Shi C. Object tracking using SIFT features and mean shift[J]. Computer Vision & Image Understanding, 2009, 113(3):345-352.

[3] Sharif Razavian A, Azizpour H, Sullivan J, et al. CNN features off-the-shelf: an astounding baseline for recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition workshops. Columbus, Ohio, USA：IEEE, 2014.

[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(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 F, Choi W, Lin Y. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA: IEEE, 2016.

[8] Wang X, Shrivastava A, Gupta A. A-fast-rcnn: Hard positive generation via adversary for object detection[J]. arXiv preprint arXiv:1704.03414, 2017, 2.

[9] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding[C]//Proceedings of the 22nd ACM international conference on Multimedia. Dallas, Texas, USA: ACM, 2014.

[10] Cho M, Brand D. MEC: memory-efficient convolution for deep neural network[J]. arXiv preprint arXiv:1706.06873, 2017.

[11] Bergstra J, Bastien F, Breuleux O, et al. Theano: Deep learning on gpus with python[C]//NIPS 2011, BigLearning Workshop, Granada, Spain: DBLP, 2011.

[12] Chetlur S, Woolley C, Vandermersch P, et al. cudnn: Efficient primitives for deep learning[J]. arXiv preprint arXiv:1410.0759, 2014.

[13] Jia Y. Learning semantic image representations at a large scale[M]. University of California, Berkeley, 2014.

[14] Van Zee F G, Van De Geijn R A. BLIS: A framework for rapidly instantiating BLAS functionality[J]. ACM Transactions on Mathematical Software (TOMS), 2015, 41(3): 14.

[15] Cireşan D C, Meier U, Masci J, et al. High-performance neural networks for visual object classification[J]. arXiv preprint arXiv:1102.0183, 2011.

[16] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[17] Winograd S. Arithmetic complexity of computations[M].

Siam: IBM Thomas J. Waston Research Center, 1980.

[18] Vasilache N, Zinenko O, Theodoridis T, et al. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions[J]. arXiv preprint arXiv:1802.04730, 2018.

[19] Nvidia C. CUBLAS library[J]. NVIDIA Corporation, Santa Clara, California, 2018, 3: 67.