利用参数稀疏性的卷积神经网络计算优化及其 FPGA 加速器设计

刘勤让 刘崇阳*
(国家数字交换系统工程技术研究中心 郑州 450002)

摘 要：针对卷积神经网络(CNN)在嵌入式端的应用受实时性限制的问题，以及 CNN 卷积计算中存在较大程度的稀疏性的特性，该文提出一种基于 FPGA 的 CNN 加速器实现方法来提高计算速度。首先，挖掘出 CNN 卷积计算的稀疏性特点；其次，为了用好参数稀疏性，把 CNN 卷积计算转换为矩阵相乘；最后，提出基于 FPGA 的并行矩阵乘法器的实现方案。在 Virtex-7 VC707 FPGA 上的仿真结果表明，相比于传统的 CNN 加速器，该设计缩短了 19%的计算时间。通过稀疏性来简化 CNN 计算过程的方式，不仅能在 FPGA 实现，也能迁移到其他嵌入式端。

关键词：卷积神经网络；稀疏性；计算优化；矩阵乘法器；FPGA
中图分类号： TP183 **文献标识码**： A **文章编号**：1009-5896(2018)06-1368-07
DOI: 10.11999/JEIT170819

Calculation Optimization for Convolutional Neural Networks and FPGA-based Accelerator Design Using the Parameters Sparsity

LIU Qinrang LIU Chongyang
(*National Digital Switching System Engineering and Technological Research Center*, *Zhengzhou* 450002, *China*)

Abstract: Concerning the problem of real-time restriction on the application of Convolution Neural Network (CNN) in embedded field, and the large degree of sparsity in CNN convolution calculations, this paper proposes an implement method of CNN accelerator based on FPGA to improve computation speed. Firstly, the sparseness characteristics of CNN convolution calculation are seeked out. Secondly, in order to use the parameters sparseness, CNN convolution calculations are converted to matrix multiplication. Finally, the implementation method of parallel matrix multiplier based on FPGA is proposed. Simulation results on the Virtex-7 VC707 FPGA show that the design shortens the calculation time by 19% compared to the traditional CNN accelerator. The method of simplifying the CNN calculation process by sparseness not only can be implemented on FPGA, but also can migrate to other embedded ends.

Key words: Convolution Neural Network (CNN); Sparseness; Computational optimization; Matrix multiplier; FPGA

1 引言

随着大数据时代的到来、计算能力的不断提升以及层出不穷的算法，人工智能应用面越来越广泛，影响也越来越显著。霍金于 2016 年曾说：“人工智能的成功将会是人类历史上最重大的事件之一”，雷鸣认为人工智能将与工业革命有同等影响力。现实生活中，网上各种推荐推送服务、智能手机上智能语言交流，以及软件上的智能机器人，都涉及了人工智能的内容。现在人工智能领域中深度学习发展非常迅速，特别是深度学习中卷积神经网络(CNN)。

提到 CNN，不得不提类脑智能，类脑智能是以计算建模为手段，受脑神经机制和认知行为机制启发，并通过软硬件协调实现的机器智能[1]，CNN 正是从生物学上获得启发的，具体来说是受到动物视觉神经的感知行为的启发。CNN 主要用在图像识别领域，也就是对图像的特征进行提取然后判别的过程[2,3]。但 CNN 的层递结构非常复杂，且连接权重数量众多，要把 CNN 用在实时性要求很高(比如自动驾驶)的实际生活中，得突破速度这一瓶颈并对 CNN 计算进行加速设计。CNN 的加速可以在 GPU, FPGA 以及 ASIC 上实现[4,5]，对于嵌入式端，FPGA 因其成本及灵活性优势独占市场鳌头[6,7]，例如现在诞生了各种 CNN 加速器的优化方法：文献[8]采用

收稿日期：2017-08-21；改回日期：2018-01-05；网络出版：2018-03-15
*通信作者：刘崇阳 983760127@qq.com
基金项目：国家科技重大专项(2016ZX01012101)，国家自然科学基金(61572520, 61521003)
Foundation Items: The National Science and Technology Major Project of the Ministry of Science and Technology of China (2016ZX01012101), The National Natural Science Foundation of China (61572520, 61521003)

定点的优化方法逐个量化每一层来降低网络参数的数量，从而降低计算复杂度。文献[9]中采用分块、存储器存取优化以及复用等措施。文献[10]提出CLP(卷积层处理器)的概念，把 FPGA 上可利用的硬件资源优化分配来提高整体计算效率。文献[11]通过把 CNN 加速器放置在电荷耦合元件附件，并设计 CNN 内部特定的数据访问模式来进行加速。文献[12]优化了硬件内存、一二级缓存以及卷积计算的迭代次序，同时对乘加计算的并行执行、sigmoid 函数分段近似表示等进行了一系列的改动来给计算进行加速。

CNN 有很多像 GoogleNet 这样著名模型，本文发现训练好的模型处理数据的过程中有很多 0 参与中间的乘法计算。为了实现对 CNN 计算加速的设计，本文充分利用了 0 元素的参数稀疏性。传统的卷积计算是 6 层循环，本文中对此进行改进，并在 FPGA 进行了实现。本文的对比对象是文献[9]，先从理论证明了该方式的可行性，之后在 FPGA 上进行了试验仿真，结果显示相比文献[9]缩短了 19%的计算时间。

2　问题与模型描述

2.1 架构与数据

CNN 在 2012 年暂露头角，到现在已经出现了许多著名结构，本文使用了 AxexNet, GoogleNet, CaffeNet 以及 VGG_ILSVRC_16_layers 这些架构来对卷积计算过程进行分析，借助的深度学习框架是 Caffe[13]。本文以训练好的模型的 caffemodel 为权重参数集，对图片数据进行测试。

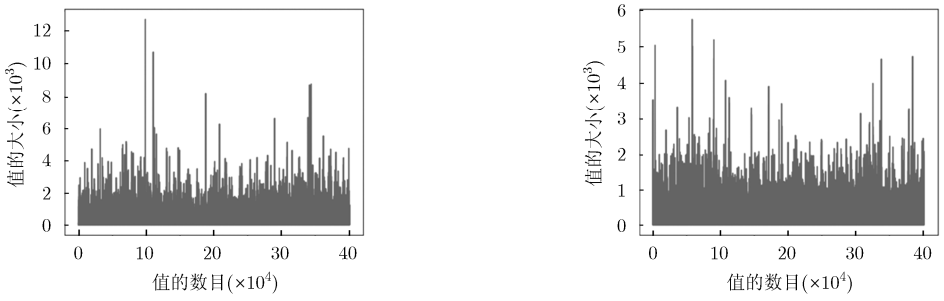
首先对 Caffe 自带的图片进行卷积计算分析。VGG 模型中 conv4_2 生成的特征图以及 conv4_2 输入的特征图(也就是 conv4_1 的输出特征图)中的值的大小及分布图如图 1，图 2。

conv4_2 生成的特征图和 conv4_2 输入的特征图值的分布有些相似，下面以 conv4_2 生成的特征图为例来进行说明。conv4_2 的输出特征图大约有 4×10^4 多的值，大小分布在 0 到 6000，而为 0 的值就接近 3×10^4，占比达到 70%，所以对各个值进行统计时，除了 0 值感觉不到其他值的存在。

CNN 卷积计算是乘法和加法的计算。CNN 中卷积计算除了输入特征图外，还有卷积核要参与其中。本文提取了已经训练好的 caffemodel 的卷积核的参数并进行了分析，发现卷积核权值中的 0 数据的占比非常小，例如 GoogleNet, VGG 中的占比不足 1/10000，所以忽略卷积核中的 0，只考虑特征图中的 0 对卷积计算的影响。

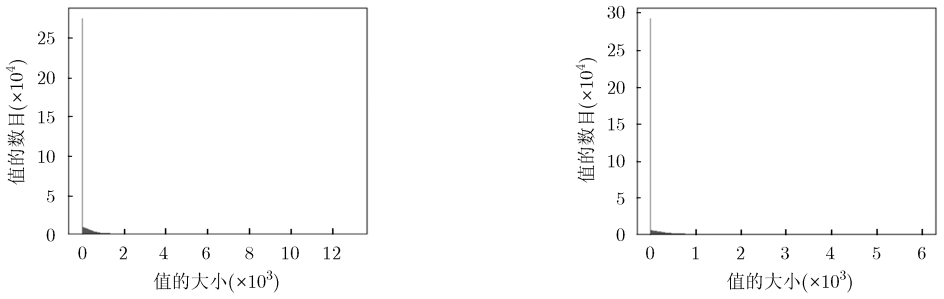
本文用多张图片测试，最终结果取平均。结果如图 3~图 6 所示。

以上各图中的最后一栏“平均”代表输入特征图中参与卷积的所有数中 0 数据的占比。可以看出，



(a) conv4_2输入特征图各值大小　　(b) conv4_2输出特征图各值大小

图 1　conv4_2 输入特征图和输出特征图的值大小



(a) conv4_2输入特征图值分布　　(b) conv4_2输出特征图值分布

图 2　conv4_2 输入特征图和输出特征图每个值的数量统计

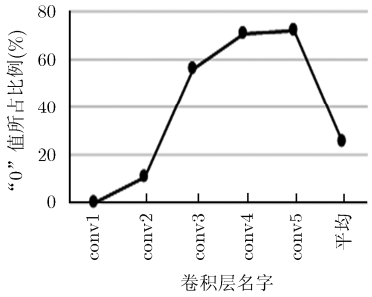


图 3 AlexNet 各层特征图中 0 所占比例

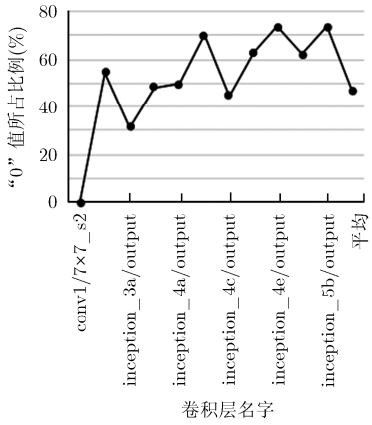


图 4 GoogleNet 各层特征图中 0 所占比例

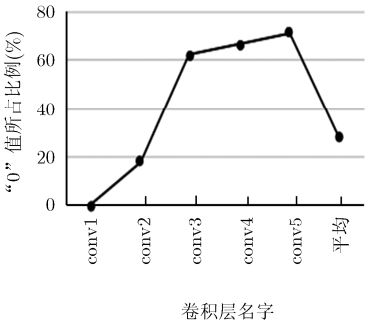


图 5 CaffeNet 各层特征图中 0 所占比例

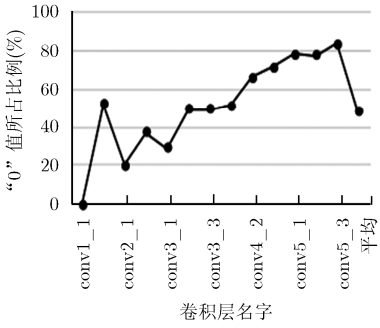


图 6 VGG_16 各层特征图中 0 所占比例

4 种模型的 0 元素占比从 20%～50%不等，稀疏性程度较大，稀疏性对于简化运算效果非常显著[14]。如果能够做到足够优化，当乘法操作中输入为 0 元素时，不进行乘法操作，直接把输出结果置为 0，那么整体计算量会降低很多，计算速度会提高很多。

2.2 CNN 卷积计算优化

CNN 中卷积计算用图 7 来阐述，卷积计算的输入对应图 7 输入特征图部分，首先有长和宽，在具体计算中对应 2 维图像像素点，图 7 的深度要对应到 3 维，也就是输入特征图的数目(不要等同于输入图片的数目)，对于第 1 次卷积计算(即输入图片被卷积)深度代表 RGB 三通道，而对于后面的卷积计算其深度从几十到几百不等(可以理解为不同特征的集合图)。卷积核尺寸(长和宽)一般较小，其深度一般情况下与输入特征图的深度一致。

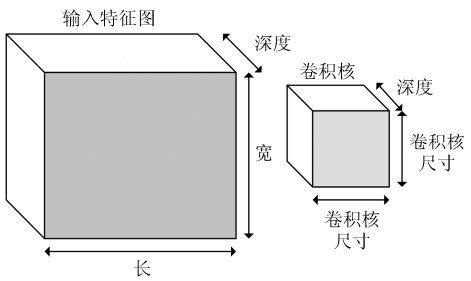


图 7 CNN 卷积方式示意图

CNN 卷积计算过程如图 8 所示，卷积计算时，卷积核在输入特征图上移动，例如图 8 中在 X 方向上移动了 4 次，则生成的特征图的长度就是 4。具体来说，生成特征图长度与移动次数密切相关，可表示为

$$C=(X-K)/S+1 \tag{1}$$

同理可得输出特征图的宽度：

$$R=(Y-K)/S+1 \tag{2}$$

式(1)中 C 代表输出特征图的长度大小，X 是输入特征图的长度大小，K 是卷积核的长度大小，S 是卷积核每次移动的步长。每次移动到一个位置，卷积核与输入特征图对应的位置就会相乘，最后相加

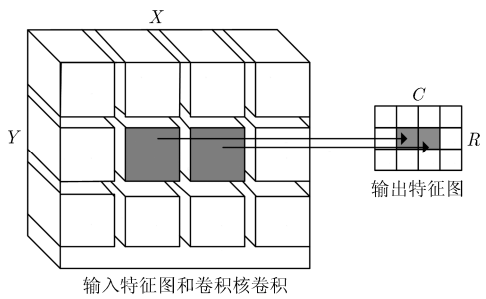


图 8 CNN 卷积计算过程示意图

得到一个输出，也就是从 3 维数组(数值数目 = $K \times K \times$深度)变成一个点(只有一个数值)。卷积过程的伪代码如表 1 所示。

表 1 卷积过程的伪代码

算法 1 常规卷积计算过程伪代码

```
for (row=0; row<R; row++){
  for (col=0; col<C; col++){
    for (to=0; to<M; to++){
      for (ti=0; ti<N; ti++){
        for (i=0; i<K; i++){
          for (j=0; j<K; j++){
output_fm[to][row][col]+=weights[to][ti][i][j]*
  imput_fm[ti][S*row+i][S*col+j]
}}}}}}
```

上述代码有 6 层循环，其中 N，M 分别代表输入和输出特征图的深度，C，R，K 和 S 的含义同式(1)、式(2)，imput_fm，output_fm 和 weights 分别是输入特征图，输出特征图和卷积核。上述计算方式不好利用稀疏性进行简化，就算用文献[9]中推荐的优化算法也不好利用。本文首先对卷积计算的方式进行转换，即使用 im2col 的方法将卷积转换为矩阵相乘，具体方式可见文献[15,16]。转化为矩阵后输入特征图的 0 数据的比例并不会有太大变化，但此时在 FPGA 上就能很方便利用稀疏性进行简化[17]，FPGA 相比于 CPU 擅长并行处理[18,19]，简单的矩阵相乘 FPGA 实现起来并不复杂，下面一小节介绍本文采用 FPGA 实现方法。

2.3 充分利用稀疏性的 FPGA 实现方式

本文用 im2col 转换后得到 2 个矩阵，分别对应输入特征图和卷积核。例如 AlexNet 的第 1 个卷积层输入转换为 $(55\times55)\times(11\times11\times3)$ 的矩阵，相应的卷积核转换为 $(11\times11\times3)\times(96)$ 的矩阵。假设矩阵 $\boldsymbol{A}$，$\boldsymbol{B}$ 分别是输入特征图、卷积核对应矩阵，则卷积转化为矩阵乘法 $\boldsymbol{C}=\boldsymbol{A}\bullet\boldsymbol{B}$，其中 $\boldsymbol{A}$，$\boldsymbol{B}$ 和 $\boldsymbol{C}$ 分别为 $M\times L$，$L\times N$ 和 $M\times N$ 维矩阵。为了充分利用稀疏性来降低计算复杂度，本文设计 S 个并行矩阵乘法器，按行把矩阵 $\boldsymbol{A}$ 分成 S 个块，每个并行矩阵乘法器单独运行，并包含 P 个处理单元(Processing Element, PE)。并行矩阵乘法器所用的数据格式为 8 位定点，用 8 位来做矩阵乘法是因为 Xilinx 白皮书里面的测试表明，在诸如图像检测及分类的 CNN 应用中，INT8 定点计算可以和 32 位浮点计算结果相当。另外，文献[9]证明了 CNN 卷积计算中浮点相比于定点会多用很多 LUT 和 FF，占用更多 FPGA 上的硬件资源，所以本文用 8 位定点来完成计算。每个 PE 中包含一个定点乘加单元和一个用于存储中间结果的存储单元，其结构如图 9 所示。

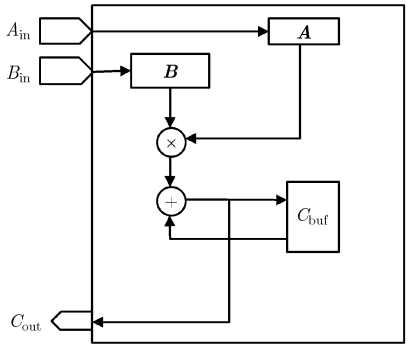


图 9　PE 内部结构示意图

图中 A_{in} 和 B_{in} 是定点乘法器的 2 个 8 位输入，C_{buf} 是一个 8 位存储单元，用于储存当前乘加操作的中间结果；当得到最终结果时输出到外面的存储单元 C_{out}。对于第 S_i ($S_i=1,2,\cdots,S$)个矩阵乘法器，在进行 $\boldsymbol{C}_{S_i}=\boldsymbol{A}_{S_i}\times\boldsymbol{B}$ 的矩阵乘法时，乘法器中的 P 个 PE 单元排列成一行(如图 10)，$\boldsymbol{A}_{S_i}$ 每次送入一个元素进行乘加运算，并且被 P 个 PE 共享，$\boldsymbol{A}_{S_i}$ 按照先行后列(例如第 1 次送第 1 行第 1 列的元素，第 2 次送第 1 行第 2 列的元素)的顺序依次送入矩阵乘法器中；$\boldsymbol{B}$ 按行的形式每次输入 P 个数据(例如第 1 次送第 1 行的前 P 个元素，第 2 次送第 1 行接下来的 P 个元素)到矩阵乘，并与 $\boldsymbol{A}_{S_i}$ 中同一个数据进行乘加操作。

A_{input} 中有个 0 值比较器，通过排列成一行的结构，首先对 $\boldsymbol{A}_{S_i}$ 中的数据进行比较操作，若输入特征图矩阵输入一个 0 值，则在比较后直接不传入 PE 中进行乘加运算，即减少了 P 次乘加操作。对于 $\boldsymbol{A}$，$\boldsymbol{B}$ 分别为 $M\times L$，$L\times N$ 维矩阵，$\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$ 可以由 S 个矩阵乘法器并行计算得到。第 S_i 个矩阵乘法器计算第 S_i 个输出 $\boldsymbol{C}_{S_i}=\boldsymbol{A}_{S_i}\times\boldsymbol{B}$ 的算法示于表 2。

从算法 2 可以看出，使用并行矩阵乘法器进行计算时，在第 2 层循环下面有个判断，$\boldsymbol{A}$ 中输入 0 数据时省去相应的后续操作。同时第 3 层循环表示

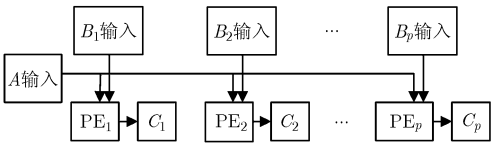


图 10 矩阵乘法器结构示意图

表 2 并行矩阵算法

算法 2 并行矩阵乘法

输入特征图矩阵：$\boldsymbol{A}_{S_i}(S_i=1,2,\cdots,S)$

输入卷积核矩阵：$\boldsymbol{B}$

预先把输出矩阵 $\boldsymbol{C}_{S_i}$ 所有元素置 0

for $i=1$ to M_{S_i} loop//遍历输入特征图矩阵 $\boldsymbol{A}_{S_i}$ 的所有行（M_{S_i} 表示第 S_i 个矩阵乘法器所要计算的行数，即 $M_{S_i}=\boldsymbol{A}_{S_i}$ 的行数）

for $j=1$ to L loop //遍历输入特征图矩阵 $\boldsymbol{A}_{S_i}$ 的所有列(同时也是卷积核矩阵的行)

if $\boldsymbol{A}_{S_i}[i][j]!=0$ //判断不是 0 后再继续下面运算，如果为 0 直接跳过，进行 j 加 1 后的判断

{

for $k=0$ to $\lceil N/P-1\rceil$ loop //每次以 P 个元素的方式遍历卷积核矩阵 $\boldsymbol{B}$ 的所有列

all PEx ($x=1,2,\cdots,P$) do in parallel

$\boldsymbol{C}_{S_i}[i][k*P+x]=\boldsymbol{A}_{S_i}[i][j]\times\boldsymbol{B}[j][k*P+x]$

End loop

}

End loop

End loop

P 个 PE 同时并行处理 P 个乘加运算，本来完全串行算法需要的 $M_{S_i}\times L\times N$ 次循环变成了 $M_{S_i}\times L\times(N/P)$ 次，即利用了 FPGA 的并行处理特性来降低计算复杂度，$\boldsymbol{C}$ 的最终结果把 S 个矩阵乘法器的结果合并就能得到。

3 性能分析

3.1 理论分析

首先从理论上进行分析，与此进行对照的是文献[9]。文献[9]中的设计主要对 6 层卷积循环进行优化，包括重新对循环进行排序，对循环单元进行分块，对内层循环的展开和流水线操作，以及增加计算量的同时力求减少数据传输带宽的要求。其可得到的性能用浮点计算性能来衡量，具体用式(3)，式(4)表示。

$$\mathrm{AP}=\min\{\mathrm{CR},\mathrm{CTC}\times\mathrm{BW}\} \tag{3}$$

$$\begin{aligned}\mathrm{CR}&=\frac{2\times R\times C\times M\times N\times K\times K}{\left\lceil\frac{M}{T_m}\right\rceil\times\left\lceil\frac{N}{T_n}\right\rceil\times\frac{R}{T_r}\times\frac{C}{T_c}\times\left(T_r\times T_c\times K\times K+P-1\right)}\\&\approx\frac{2\times R\times C\times M\times N\times K\times K}{\left\lceil\frac{M}{T_m}\right\rceil\times\left\lceil\frac{N}{T_n}\right\rceil\times R\times C\times K\times K}\end{aligned} \tag{4}$$

式(3)中，AP 指可得到的最大性能，CR 指系统中所有可用计算资源提供的峰值计算性能，CTC 指计算与通信的比例(即每数据搬移可以完成的计算操作量)，BW 指带宽。式(4)中 R，C，M，N 和 K 具体所指同算法 1，T_m，T_n，T_r 和 T_c 分别是对输出特征图深度，输入特征图深度，输出特征图宽和长的分块，P 指流水线深度。对式(4)进行简化即可得到的最大计算性能为 $2T_mT_n$。本文中对计算单元的硬件设计方式如图 11。根据图 11，$2T_mT_n$ 约等于所有乘法单元和加法单元的总和，即文献[9]可得到的性能为硬件单元中乘法单元和加法单元的总和。本文的设计中，每个 PE 中有一个乘法单元和一个加法单元，理想情况下一个操作周期(不是时钟周期)内可以完成一次乘法一次加法(共 2 次操作)，可以得到的性能能达到 $2SP$。当本文和文献[9]使用数量相等的硬件资源时，即 $2T_mT_n$ (文献[9]中所使用的乘加单元总和)等于 $2SP$ (本文的矩阵乘法器所使用的乘加单元总和)时，本文设计的矩阵乘法器可得到性能可以达到文献[9]所说的最大性能。虽然本文比文献[9]多了一些加法器和比较器，同时还要在处理器上进行 im2col 的数据转化操作，但 CNN 中大约 90% 的时间都花在卷积层操作中[20,21]，同时卷积操作中最耗时的又是乘法运算，本文利用稀疏性理论上可以减少至少 20%的乘加操作，从整体来看明显减少了计算时间，加速了 CNN 计算过程。

3.2 实验结果及分析

本文矩阵乘法器的设计在 Vivado HLS (V2015.4)上实现。这个工具能实现 C 代码到 RTL (寄存器传输级)级别的转换，本文设计的矩阵乘法的C代码添加HLS定义的编译指令来达到并行化处理，其并行性通过时序分析工具来进行验证。其具有的C代码的仿真和C / RTL协同仿真能实现综合之前的仿真，综合之前的资源利用报告可以帮助改进设计以及提供性能估计。为了和文献[9]中形成对比，也在 Virtex-7 VC707 Evaluation Platform 上进行了试验，模型选择也是 AlexNet，同时把工作频

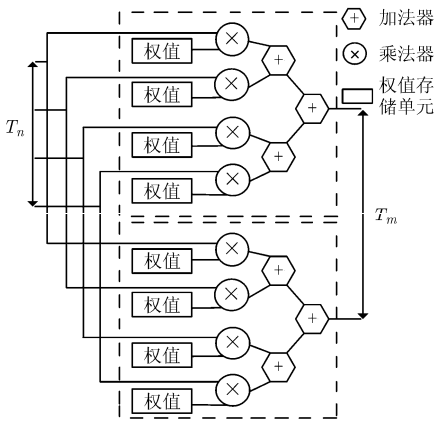


图 11 文献[9]卷积计算单元硬件结构

率设置为 100 MHz。文献[9]最终确定的T_m为 64, T_n为 7，为了更好地比较，本文最终选定的参数$S\times P$要等于 448(64×7)。

下面对S和P的选定进行一般性说明，首先对于S，希望越小越好。因为S对应并行矩阵乘法器的数量，而每个并行矩阵乘法器都有一个 0 值比较器，S越多，则 0 值比较器的数量就越多，占用资源也就越多。最好的极限情况下S为 1，即只用一个比较器。与此同时，P也不能太大。因为P越大，对卷积核矩阵$\boldsymbol{B}$的列分段程度也就越大，在计算最后一段时，$\boldsymbol{B}$的输入数据量为P除N的余数，有可能矩阵乘法器中很多计算单元都处于闲置状态，也浪费了资源。

对于不同资源的 FPGA，为充分利用资源以及最大化性能，希望SP乘积尽量大，当然SP乘积也有一个限额，本文根据 HLS 的资源利用报告进行综合考虑。之后结合一般性说明以及所要加速的模型来调节参数S和P的具体值。本文中为充分对比，已经确定SP乘积为 448，具体到S和P时，依据前面的选定规则并结合 AlexNet 模型参数，本文最终选定S为 14，而P为 32。在理想情况下，每个 PE 在一个计算周期内可以完成一次乘法和一次加法操作，整个 FPGA 上的矩阵乘法器的计算性能如式(5)：

$$\mathrm{CR}=2\times S\times P\times f \tag{5}$$

式中，CR 代表S个乘法器峰值计算性能总和(单位为每秒百万次计算操作)，S,P以及f分别为 14,32 以及 100 MHz，即该设计在去掉 0 元素参与计算后的最大计算性能达到每秒 89.6 G 次计算操作。

利用上述参数在 Vivado 工具中实现，其资源利用情况如表 3，通过对比文献[9]可以看出，BRAM_18K 使用量增加了，其他 3 种资源使用量都减少了，下面从实现方式的不同来进行分析。本文是直接把卷积计算所用到的输入特征图矩阵和卷积核矩阵转化过来加载到 FPGA 上进行计算的，并没有像文献[9]一样采用分块循环的方式，所以使用较多的片上硬件存储资源。本文中采用的是定点的数据格式，而定点乘法器会使用较少的 DSP48E，所以总使用量减少了。FF 的利用率减少了 5%，虽然从理论上定点计算不使用 FF，但本文中每个 PE 存储中间结果以及最终结果的存储单元都是由 FF 来担任寄存器，相比于文献[9]的流水线结构，本文中的完全并行结构需占用更多的寄存器。LUT 的使用量减少的主要原因也是定点乘加不需使用 LUT 资源，但本文中的比较器要进行相应表达还是大量需要 LUT 的，所以使用量上并没有减少太多。

表 3　FPGA 内部资源使用情况

资源类别	BRAM_18K	DSP48E	FF	LUT
总使用量	1458	1792	170757	142304
可利用总量	2060	2800	607200	303600
利用率(%)	70.78	64.00	28.12	46.87

本文和文献[9]使用不同的数据格式，一次浮点计算比定点计算要复杂，单考虑计算次数意义不大，加速效果以具体计算时间最为合适。计算时间的对比如表 4 所示。

表 4 计算时间比较

	FPGA2015[9] (32 位浮点)(ms)	本文实现 (8 位定点)(ms)	比例
Conv1	7.67	8.02	1.05
Conv2	5.35	5.14	0.96
Conv3	3.79	2.13	0.56
Conv4	2.88	1.31	0.46
Conv5	1.93	0.88	0.46
时间总计	21.61	17.48	0.81

从表 4 中可以看出，除了第 1 个卷积层计算时间增加以外，其他层的时间都有相应的减少，其减少时间与稀疏性程度有关，总体时间减少了约 19%。本文中增加了对 0 的判断减少了乘加操作，其定点操作次数从理论上分析为文献[9]的 1-25.92%=74.08%(其中 25.92%为整体上参与乘加计算中乘加 0 操作的比例)，本文计算性能与文献[9]的比例为

$$\frac{1-25.92\%}{1-19\%}=91.46\% \tag{6}$$

因此本文实际的计算性能并未达到文献[9]所优化的情况，此外本文中对卷积的转化需要的额外的时间未考虑进来，但本文最大的意义是对于稀疏性的利用让计算时间得以明显减少，因此对本文方法进行硬件结构上的进一步优化(比如本文中未充分考虑 I/0 的设计)可以实现更少的计算时间，这也是接下来需要做的工作。

4　结束语

本文针对嵌入式端对 CNN 计算时间的严格要求的现状，提出了一种 CNN 卷积计算加速方法。对卷积计算进行分析，发现其中计算数据存在较大程度的稀疏性，为利用该稀疏性，本文对卷积进行转换，转换为简单的矩阵相乘。同时，本文设计了一种并行矩阵乘法器，避免了 0 元素的无效乘加计算来提高效率。为了更好说明该方法的有效性，本文在 FPGA 上进行了试验，并与文献[9]进行了充分

比较，结果说明本文方法所需的计算时间更少，即可以有效地对 CNN 硬件加速器进行加速。本文方法的有效性主要是由 CNN 计算本身的特点所决定，只要对计算中存在的稀疏性加以利用，从理论上来说可以拓展到其它嵌入式端。

参 考 文 献

[1] 曾毅, 刘成林, 谭铁牛. 类脑智能研究的回顾与展望[J]. 计算机学报, 2016, 39(1): 212–222. doi: 10.11897/SP.J.1016.2016.00212.
ZENG Yi, LIU Chenglin, and TAN Tieniu. Retrospect and outlook of brain-inspired intelligence research[J]. *Chinese Journal of Computers*, 2016, 39(1): 212–222. doi: 10.11897/SP.J.1016.2016.00212.

[2] 常亮, 邓小明, 周明全, 等. 图像理解中的卷积神经网络[J]. 自动化学报, 2016, 42(9): 1300–1312. doi: 10.16383/j.aas.2016.c150800.
CHANG Liang, DENG Xiaoming, ZHOU Mingquan, *et al*. Convolutional neural networks in image understanding[J]. *Acta Automatica Sinica*, 2016, 42(9): 1300–1312. doi: 10.16383/j.aas.2016.c150800.

[3] JI S, XU W, YANG M, *et al*. 3D convolutional neural networks for human action recognition[J]. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2012, 35(1): 221–231. doi: 10.1109/TPAMI.2012.59.

[4] CHAKRADHAR S, SANKARADAS M, JAKKULA V, *et al*. A dynamically configurable coprocessor for convolutional neural networks[J]. *ACM Sigarch Computer Architecture News*, 2010, 38(3): 247–257. doi: 10.1145/1816038.1815993.

[5] KRIZHEVSKY A, SUTSKEVER I, and HINTON G E. ImageNet classification with deep convolutional neural networks[C]. International Conference on Neural Information Processing Systems, Lake Tahoe, Nevada, 2012: 1097–1105. doi: 10.1145/3065386.

[6] SUDA N, CHANDRA V, DASIKA G, *et al*. Throughput-optimized openCL-based FPGA accelerator for large-scale convolutional neural networks[C]. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, California, USA, 2016: 16–25. doi: 10.1145/2847263.2847276.

[7] QIU J, WANG J, YAO S, *et al*. Going deeper with embedded FPGA platform for convolutional neural network[C]. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, California, USA, 2016: 26–35. doi: 10.1145 /2847263.2847265.

[8] ANWAR S, HWANG K, and SUNG W. Fixed point optimization of deep convolutional neural networks for object recognition[C]. IEEE International Conference on Acoustics, Speech and Signal Processing, Brisbane, QLD, Australia, 2015: 1131–1135. doi: 10.1109/ICASSP.2015.7178146.

[9] ZHANG C, LI P, SUN G, *et al*. Optimizing FPGA-based accelerator design for deep convolutional neural networks[C]. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, California, USA, 2015: 161–170. doi: 10.1145/2684746.2689060.

[10] SHEN Y, FERDMAN M, and MILDER P. Maximizing CNN accelerator effciency through resource partitioning[C]. Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 2017: 535–547. doi: 10.1145/3140659.3080221.

[11] DU Z, FASTHUBER R, CHEN T, *et al*. ShiDianNao: Shifting vision processing closer to the sensor[C]. Annual International Symposium on Computer Architecture, Portland, Oregon, 2015: 92–104. doi: 10.1145/2749469.2750389.

[12] CHEN T, DU Z, SUN N, *et al*. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[C]. International Conference on Architectural Support for Programming Languages and Operating Systems, Salt Lake City, Utah, USA, 2014: 269–284. doi: 10.1145/2541940.2541967.

[13] HADJIS S, ABUZAID F, ZHANG C, *et al*. Caffe con troll: Shallow ideas to speed up deep learning[C]. Proceedings of the Fourth Workshop on Data analytics, Melbourne, VIC, Australia, 2015: 1–4. doi: 10.1145/2799562.2799641.

[14] YAVITS L, MORAD A, and GINOSAR R. Sparse matrix multiplication on an associative processor[J]. *IEEE Transactions on Parallel & Distributed Systems*, 2015, 26(11): 3175–3183. doi: 10.1109/TPDS.2014.2370055.

[15] CHELLAPILLA K, PURI S, and SIMARD P. High performance convolutional neural networks for document processing[C]. Tenth International Workshop on Frontiers in Handwriting Recognition, La Baule, France, 2006: 1–6.

[16] CHETLUR S, WOOLLEY C, VANDERMERSCH P, *et al*. CuDNN: Efficient primitives for deep learning[C]. International Conference on Neural Information Processing Systems, Montreal, Canada, 2014: 1–9.

[17] 田翔, 周凡, 陈耀武, 等. 基于 FPGA 的实时双精度浮点矩阵乘法器设计[J]. 浙江大学学报(工学版), 2008, 42(9): 1611–1615. doi: 10.3785/j.issn.1008-973X.2008.09.027.
TIAN Xiang, ZHOU Fan, CHEN Yaowu, *et al*. Design of field programmable gate array based real-time double-precision floating-point matrix multiplier[J]. *Journal of Zhejiang University* (*Engineering Science*), 2008, 42(9): 1611–1615. doi: 10.3785/j.issn.1008-973X.2008.09.027.

[18] JANG J, CHOI S B, and PRASANNA V K. Energy- and time-efficient matrix multiplication on FPGAs[J]. *IEEE Transactions on Very Large Scale Integration Systems*, 2005, 13(11): 1305–1319. doi: 10.1109/TVLSI.2005.859562.

[19] KUMAR V B Y, JOSHI S, PATKAR S B, *et al*. FPGA based high performance double-precision matrix multiplication[J]. *International Journal of Parallel Programming*, 2010, 38(3/4): 322–338. doi: 10.1109/VLSI.Design.2009.13.

[20] DONAHUE J, JIA Y, VINYALS O, *et al*. DeCAF: A deep convolutional activation feature for generic visual recognition[C]. International Conference on Machine Learning, Beijing, China, 2014: 647–655.

[21] PETE Warden. Why GEMM is at the heart of deep learning[OL]. https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/.

刘勤让： 男，1975 年生，研究员，研究方向为宽带信息网络及芯片设计.

刘崇阳： 男，1994 年生，硕士生，研究方向为人工智能芯片设计.