高技术通讯 2022 年 第32 卷 第3 期:227-238

doi:10.3772/j.issn.1002-0470.2022.03.002

面向目标检测的卷积神经网络优化方法①

张志超②[*,**,***,****] 王 剑[*,**,***] 章隆兵[*,**,***] 肖俊华[*,**,***]

([*]计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

([**]中国科学院计算技术研究所 北京 100190)

([***]中国科学院大学 北京 100049)

([****]中国电子科技集团公司第十五研究所 北京 100083)

摘 要 针对星载等功耗受限平台下遥感影像目标检测存在的高准确率、低功耗以及高吞吐量等要求,本文提出了一种面向目标检测的现场可编程门阵列(FPGA)卷积神经网络(CNN)优化方法。采用数据流调度技术以及基于乘法矩阵与前向加法链的卷积计算阵列设计对浮点卷积神经网络模型进行加速。利用该方法在FPGA开发板上实现了浮点卷积目标检测网络,在应用中达到了与原模型一致的准确率,平均准确率为97.59%,吞吐量达到了Titan X的22倍。与同类的FPGA加速浮点卷积方法对比,该方法的吞吐量以及能效比达到了最优。实验数据表明,该方案突破了浮点卷积加速的线速吞吐难点,解决了应用中存在的功耗、准确率以及吞吐量三者制衡的问题。

关键词 卷积神经网络(CNN);现场可编程门阵列(FPGA);数据流调度;目标检测;加速

0 引 言

以卷积神经网络(convolution neural network, CNN)为代表的深度学习技术广泛应用于图像分类、目标检测等应用中。遥感影像领域目标检测与识别应用也广泛应用卷积神经网络技术。近年来深度学习的发展经历了由小模型[1]到大模型[2-3]、再由大模型到轻量型模型[4-6]的发展。然而深度卷积神经网络算法存在计算复杂度高、参数存储量大等特点,导致其在星载、机载等功耗受限环境下进行目标检测处理运算时间长、功耗大等问题,并且面向领域应用的算法需要较高的目标检测准确率。现有的卷积神经网络的加速研究包括参数量化[7-10]、轻量型网络结构设计[4-6]、硬件结构设计[11-12]等方法。

基于轻量型网络结构设计[4-6]的方法展现了较高的准确率,在嵌入式处理器上能够达到近实时的处理性能,在图形处理器(graphics processing unit, GPU)平台处理性能高但功耗大。在星载、机载等数据产生终端,存在数据量大、平台处理功耗受限等情况,需要结合现场可编程门阵列(field programmable gate array, FPGA)方法进行浮点精度模型加速设计,提升处理能力。

一般的目标检测模型采用浮点训练推理的方式使用,参数量化[7-9]的方法存在精度降低问题,需要特定的硬件结构支撑量化计算来提升性能,并且对于紧凑小模型的量化收效甚微[10]。通用的基于FPGA加速浮点CNN模型[11,13]受限于通用模型复杂度高,基于乘累加的卷积计算方法存在达不到线速吞吐问题,造成加速效果有限,需要突破浮点乘累加造成的线速吞吐瓶颈,提升CNN加速处理能力。

本文在研究了大量CNN加速方法后,针对遥感

① 国家自然科学基金(61432016)和国家重点研发计划(2018YFC0832306,2018YFC0831203,2018YFC0831206)资助项目。

② 男,1990年生,博士生;研究方向:计算机系统结构;联系人,E-mail: zhangzhichao@ict.ac.cn。

(收稿日期:2021-02-04)

影像广域场景筛选机场目标区域的垂直应用需求，采取面向领域应用定制轻量化 CNN 目标检测模型，结合基于数据流调度的 FPGA 加速浮点卷积神经网络的方法以及基于乘法矩阵与前向加法链的卷积计算阵列设计，构建软硬件 CNN 模型一致性的加速方案，解决了卷积计算中浮点加速线速吞吐的问题，达到了功耗受限条件下实时计算的需求，符合应用要求的高准确率，并且使得软硬件开发去耦合，软件人员能够进一步优化模型以及研究模型的认知不确定性[14]等问题，以提供更可信任的深度学习模型。

1　卷积神经网络模型设计

针对星载、机载等功耗受限数据产生终端进行遥感影像机场目标检测的垂直应用需求，需要在大尺度区域中选出感兴趣区域对目标进行识别，这种热点区域的识别能够显著降低细粒度目标识别的尺度范围，进一步缩小目标识别的运算开销。面向机场目标检测的卷积网络模型，采用基于 LeNet-5[1]与 Yolo-v2[15]启发的网络骨干结构及目标检测策略设计目标检测网络模型，利用当前的小卷积核紧致模型的运算量低的特点，在解决目标检测识别准确率基础上，尽可能设计小的卷积网络降低模型计算量需求，便于后续的基于 FPGA 的处理加速。

1.1　基于 CNN 的目标区域筛选算法

针对从大区域遥感影像中筛选机场目标区域的技术需求，分析出机场目标于周围环境特征明显，该问题类似于 MNIST 手写数字识别问题。本文受 LeNet-5 模型和 Yolo-v2 目标检测模型的启发，自定义了一个 10 层的轻量型目标识别网络用于筛选机场区域目标，进而降低大区域卫星影像中飞机目标识别的计算量。该目标识别网络的检测层利用了 Yolo-v2 的候选款筛选算法，将这 10 层卷积的轻量型命名为 Conv10 _ Yolo。该目标区域筛选算法的网络结构如图 1 所示，里面包含 10 层卷积层和 2 层池化层，该卷积网络的输入为 128 ×128 像素的 3 通道卫星影像，输出为 4 ×4 ×30 维的目标检测向量。

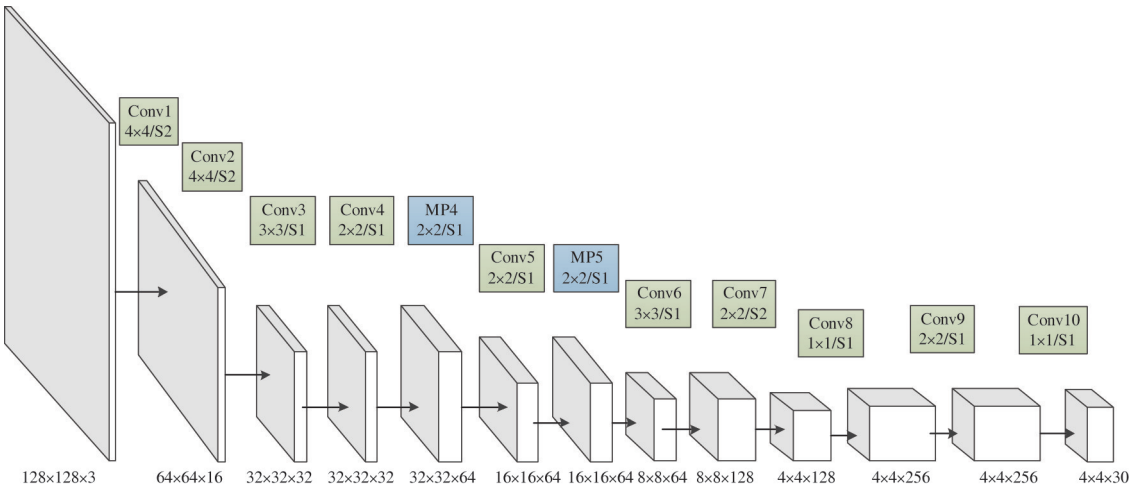

图 1　基于 CNN 的目标区域选择算法网络结构

具体的 Conv10 _ Yolo 每层的卷积或池化参数配置如表 1 所示，初始选用大卷积(4 ×4)来提取特征，利用卷积的步进降低输出维度，中间层主要采用 2 ×2 卷积以及 3 ×3 卷积进行特征提取，后续层主要采用 1 ×1 卷积与 2 ×2 卷积进行特征提取，检测输出层采用 1 ×1 卷积输出检测目标属性信息。

目标检测层采用 1 ×1 卷积实现，输出维度为 4 ×4 ×30 的检测向量。该检测层参考了 Yolo-v2 的相对坐标检测思路，包含 5 个候选框，每个候选款包含 6 个目标属性，具体如图 2 所示。该检测层输出的坐标为预测的候选款相对中心点坐标，具体坐标解算参考 Yolo-v2 的坐标解算。候选框的预设值采用 KNN 聚类方法，对训练数据中目标尺寸太小的进行维度聚类得到候选框的预设值，具体值为[1.13，

表 1　基于 CNN 的目标区域选择算法网络结构

名称	核大小	步进	输入通道数	输出通道数	输入维度
Conv1	4	2	3	16	128
Conv2	4	2	16	32	64
Conv3	3	1	32	32	32
Conv4	2	1	32	64	32
MP4	2	1	64	64	32
Conv5	2	1	64	64	16
MP5	2	1	64	64	16
Conv6	3	1	64	128	8
Conv7	2	2	128	128	8
Conv8	1	1	128	256	4
Conv9	2	1	256	256	4
Conv10	1	1	256	30	4

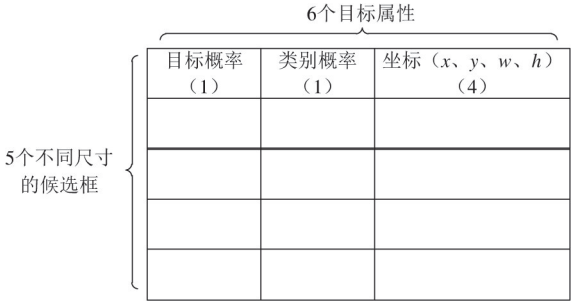

图 2　检测层输出向量维度分解

1.92, 1.70, 2.04, 1.99, 0.98, 2.28, 1.73, 2.70, 2.69]，共 5 组 x、y 坐标值。目标检测层卷积输出为相对于候选框的偏移量，目标框的解算按照 Yolo-v2 的解算思路进行计算。

1.2　模型复杂度分析

该部分分析面向目标区域筛选的卷积神经网络快速识别算法的卷积网络模型参数与计算复杂度。

通用的卷积参数计算如式(1)所示。

$$W_{conv} = IC \times OC \times k^2 \tag{1}$$

其中，IC(input channel)为输入通道大小，OC(output channel)为输出通道大小，k 为卷积核尺寸。

卷积层计算乘累加数量如式(2)所示。

$$MAC_{conv} = IC \times OC \times OW \times OH \times k^2 \tag{2}$$

其中，OW(output width)为卷积计算输出宽度，OH(output height)为卷积计算输出高度。

卷积层计算的操作数通常有 *GOPs* 指标，该指标将卷积乘法和加法分开计算，通常一个包含 *bias* 的卷积计算加法数量与乘法数量一致，具体如式(3)所示。

$$GOPs = 2 \times MAC_{conv}/10^9 \tag{3}$$

通过计算得出，该模型的权重总大小为 1.85 MB，总乘累加操作数为 0.044 GMAC，总卷积计算操作数为 0.088 GOPs。由此可见，该检测模型为一个深层轻量化的目标检测网络，适于在 FPGA 上实现全浮点精度的卷积网络运算加速。

2　基于数据流调度的浮点 CNN 加速器设计

在 FPGA 上加速目标检测模型，需要解决模型一致性问题，降低应用移植的难度，保持模型检测准确率的一致性。采用基于数据流调度的浮点卷积神经网络加速器设计方法，使得卷积网络的浮点参数权重保持在片上存储，多层卷积网络计算同时以流水线的方式在片上执行，即解决了模型加速与能效比问题，又能符合加速模型与原始模型计算一致性的需求。

2.1　基于数据流调度的卷积神经网络加速总体架构

针对基于卷积神经网络的目标区域筛选算法，设计基于数据流计算的卷积神经网络加速器。Xilinx UltraScale + 系列新品的内部物理结构分布为 3 个超级逻辑区(super logic region，SLR)，每个逻辑区覆盖芯片内部的一个 Die，整颗芯片由 3 颗 Die 通过基底层互联。

考虑逻辑分区问题，将整个数据流计算分为 3 大块。第 1 块超级逻辑区(SLR0)覆盖卷积网络的前 3 层卷积层计算，即 Conv1、Conv2、Conv3 的计算；第 2 块超级逻辑区(SLR1)覆盖中间 3 层卷积层计算和 2 层最大池化层计算，即 Conv4、Mp4、Conv5、Mp5、Conv6 的计算；第 3 块超级逻辑区(SLR2)覆盖后 4 层卷积层计算，即 Conv7、Conv8、Conv9 以及 Conv10 的计算。

超级逻辑区之间通过 Xilinx 堆叠硅片互联(stacked silicon interconnect，SSI)技术互联，在超级逻辑区边缘有特定的专用寄存器进行跨逻辑区通信，基底层有超长线(super long line，SLL)来连接专

用传输寄存器。超级逻辑区的内部访存和跨逻辑区通信需求,均通过 SLL 互联实现。具体的基于数据流计算的卷积神经网络加速总体架构如图 3 所示。

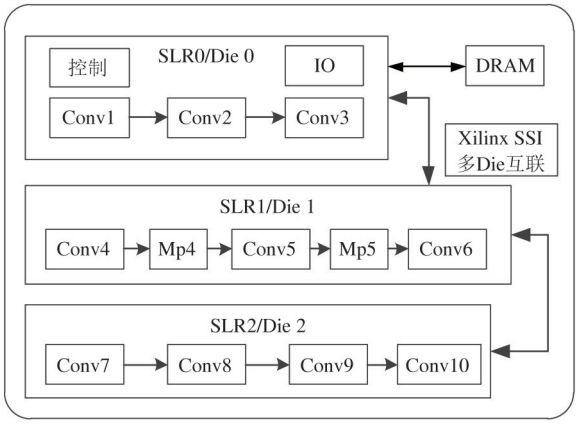

图 3　基于数据流计算的 CNN 加速总体架构

2.2　基于数据流调度的卷积层设计

通常的卷积层计算由卷积乘加计算、批归一化处理以及非线性响应等部分组成。在基于 FPGA 的卷积加速设计中,设计一个大的卷积乘加矩阵是卷积计算性能的必要条件。在没有硬核浮点支持的 FPGA 中,如何设计一个保持线速处理的乘加矩阵也是一个难点。在基于数据流的 FPGA 卷积计算单元里,卷积层的输入调度、权重调度也必须达到卷积计算矩阵的线速输入条件,这样才能避免卷积计算矩阵空转,浪费计算性能。本部分阐述基于数据流的卷积层计算,主要包括卷积层权重输入、权重缓存、卷积输入调度、卷积计算阵列、批归一化处理、非线性响应等部分。具体的基于数据流的卷积层初始化与计算流程如图 4 所示,其中权重初始化只在权重缓存初始化的时候运行一次,后续的卷积计算不再需要从动态随机存取存储器(dynamic random access memory, DRAM)中调度缓存数据,降低了存储带宽的开销,可避免卷积计算阵列需要等待权重数据。

(1)卷积特征图内存表示

卷积输入及输出特征图在内存中维持“$H \times W \times C$”的内存分布,卷积输入及输出特征图在数据流中按照“$H \times W \times C$”的方式传输,即通道优先(channel first)的方式传输。图像与特征图本质上是一个三维矩阵,图像一般具有RGB三维通道,特征图的

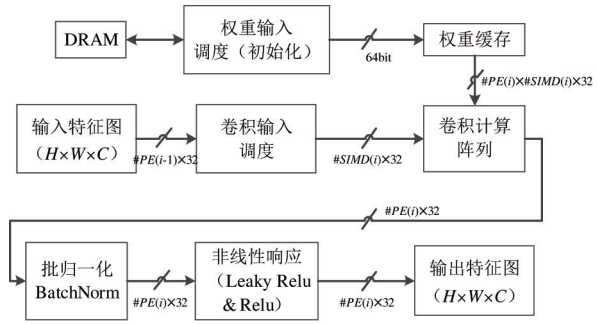

图 4　卷积层初始化与计算流程

通道数为 16 ~ 2048,如果把图像或者特征图的通道数看成一个像素点的属性,则图像的抽象表示可以维持一个“二维抽象”,具备宽 × 高的像素点集合,像素的属性个数是可变的。后面的卷积计算就可以基于像素处理,图像或者特征图可以抽象为一个二维的处理过程。

进一步把图像或者特征图的“$H \times W$”个像素平铺,则图像或者特征图可以进一步维持一个“一维抽象”,即“$H \times W$”行像素点,这种表示不仅适合内存的线性存储,也适用于在数据流中传输图像或者特征图。具体的输入图像与特征图的内存表示如图 5所示。

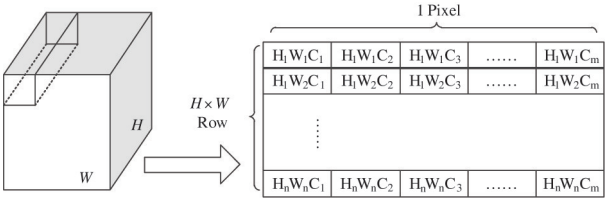

图 5　卷积层输入图像与特征图的内存表示

后续的计算流程在输入输出上都维持这样的“$H \times W \times C$”的内存表示。基于统一的内存表示,多个卷积层计算可以通过先入先出队列(first input first output, FIFO)互联并协同计算,避免中间结果经由内存导入导出,消耗内存带宽,同时也浪费一定量的卷积层计算周期。

(2)卷积层数据流调度

基于数据流的卷积计算要求输入数据及权重都能够达到线速吞吐,这样卷积计算单元的流水周期不会出现空转,计算效率达到最高。基于线速吞吐的数据流调度要求,设计多端口的权重缓存,能够提

供处理单元(process element, PE)组权重,每组权重有单指令多数据(single instruction multiple data, SIMD)个权重32位浮点数;设计多端口卷积输入缓存,能够提供 *SIMD* 个32位浮点数特征图输入;同时引入与卷积输入缓存 *Stride*(卷积滑窗步进)行同等大小的输入FIFO,与卷积输入缓存构成异构"乒乓"缓存,去耦合上个阶段的卷积计算输出调度和本阶段的卷积计算输入调度。输入特征图的输入宽度为上一层 *PE* 个数量的32位浮点数,通过输入FIFO转换宽度为 *SIMD* 个32位浮点数据。卷积计算单元的输出为 *PE* 个32位浮点数据计算输出。具体的卷积层数据流调度如图6所示,实现了一组数据计算多个卷积输出,利用了输入数据的局部性。

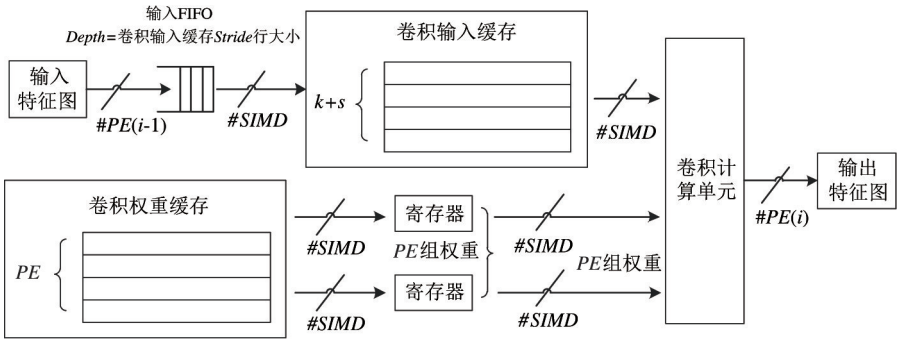

图6　卷积层数据流调度

(3)卷积层输入缓存设计

卷积层输入缓存与输入FIFO的大小一致,结构不同,卷积层输入缓存将"$H \times W \times C$"的输入特征图转换为卷积核所需的"$k \times k \times C$"的输入,通过在多组寄存器滑窗,为卷积计算阵列提供计算卷积所需的线性输入。具体的卷积输入缓存设计如图7所示。

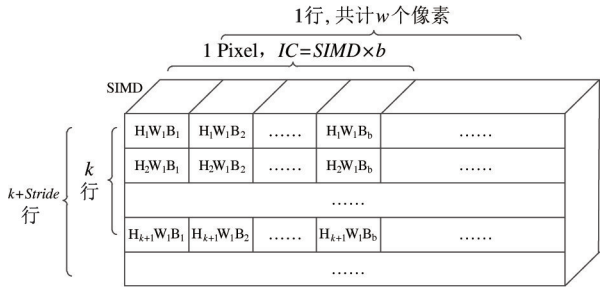

图7　卷积层输入缓存设计

卷积层输入缓存包括 k + *Stride* 行寄存器组,其中 k 为卷积核大小,*Stride* 为卷积滑窗的步进大小。每行寄存器缓存输入特征图多个通道一行的数据,即"$W \times C$"个数据,W 为输入特征图的宽度,*IC* 为输入特征图的通道数。每个寄存器行都可以按照 *SIMD* 个浮点数的宽度输出数据,一个像素的多个通道数据按照 *SIMD* 的宽度分为 b 个Block,每个Block按照顺序放在寄存器行里。由于二维卷积的空间访存特性,需要从多个寄存器行里顺序读取数据,按照卷积核行列计算的要求顺序输出 *SIMD* 宽度的输入特征图数据流。

多个寄存器组分开存放,可以将卷积输入特征图的输出调度和输入调度分开,除了初始化读入 k 行输入特征图像素,后续可以按照 *Stride* 行补充输入特征图数据。虽然访问的同一个卷积输入缓存,利用多行卷积输入缓存寄存器,由此可以将输入数据与输出数据调取去耦合,避免数据依赖和数据污染,实现无锁的数据输入输出同时调度,满足卷积计算单元 *SIMD* 个浮点数据的输出线速调度要求。

(4)卷积层权重缓存设计

针对在片上缓存该层卷积计算所需的全部卷积,并且权重缓存的输出能按照卷积计算阵列"*PE* × *SIMD*"的计算模式输出面向多个处理单元(*PE*)输出每组 *SIMD* 个浮点权重数据等要求,设计卷积层权重缓存。

卷积层参数按照式(4)维度顺序进行存放,其中 *PE* 为卷积计算单元的处理单元个数,*SIMD* 为每个卷积处理单元输入数据的宽度,每个数据均为32位浮点数据。

$$W = PE \times T \times SIMD \tag{4}$$

卷积权重缓存 T 的计算如下：

$$T = \frac{IC}{SIMD} \times k^2 \times \frac{OC}{PE} \tag{5}$$

其中，IC 为卷积核输入通道数量，OC 为卷积核输出通道数量，k 为卷积核大小。

卷积层权重缓存具有 PE 组独立的缓存行，每个行的输出宽度为 $SIMD$ 个浮点数，由于卷积计算阵列的多个 PE 输出为输出特征图的一个像素的连续属性，故需要在权重缓存里将卷积权重按照多个 PE 行寄存器交叉存储，保证输出的 PE 组权重通过卷积计算阵列计算得到的是连续的输出特征图一个像素内部的连续属性值。具体的存放顺序如图 8 所示。

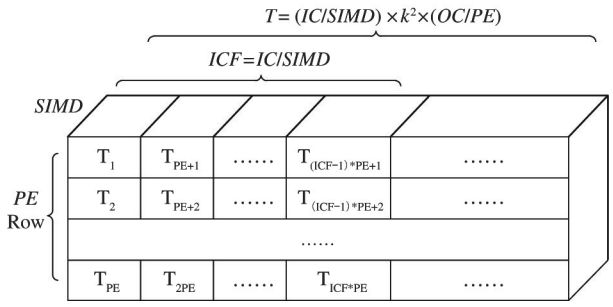

图 8　卷积层权重缓存设计

(5)卷积阵列计算单元设计

针对卷积计算高吞吐量的计算需求，设计卷积计算阵列，每个时钟周期能够处理“$PE \times SIMD$”个乘累加计算，其中 PE 为卷积阵列的行数，$SIMD$ 为卷积输入特征图数据的个数，输出为 PE 个卷积计算输出，PE 个浮点数据属于同一个输出特征图像素的连续属性值。

通常卷积计算存在一个乘累加计算矩阵，基于定点数的累加计算，通过高级综合工具调度出来的累加器延迟为1个clk，累加计算可以达到线速吞吐；而基于浮点数的累加计算，通过高级综合工具调度出来的累加器延迟为 4 个 clk，吞吐达不到线速，严重影响整个卷积计算阵列的吞吐速率。具体的累加器性能分析如图 9 所示。

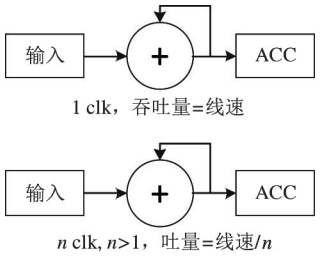

图 9　累加器性能分析

考虑浮点累加器调度延迟问题主要在与累加器输出要反馈到累加器输入上，这个延迟决定了累加器的吞吐速率，如果能够打破累加器的后向反馈，则可以使累加器性能达到线速，进而卷积计算阵列达到线速。

由于卷积的累加次数是固定的，且随着每个卷积层参数而累加次数不同。基于数据流的卷积计算阵列支持面向不同的卷积层分别配置卷积计算参数，包括卷积阵列大小、卷积累加的次数，为此设计了一个基于前向累加链的卷积累加器。卷积计算单元的每个 PE 输出 $SIMD$ 个数据，通过加法树 1 汇集成一个数据输出，通过加法树 2、3 分别汇聚 k 个数累加，后续可以通过个加法器进行后续累加(图 10)。基于前向加法链的卷积累加器共需 N 个加法器，计算公式如式(6)所示，ICF 的计算公式如式(7)所示。

$$N = 2(k-1) + \log_2 ICF \tag{6}$$

$$ICF = IC/SIMD \tag{7}$$

由于累加矩阵可以在后续完成累加计算，故卷积阵列只需要实现为乘法阵列，具体的基于乘法矩阵与前向加法链的卷积计算阵列如图 11 所示，包含

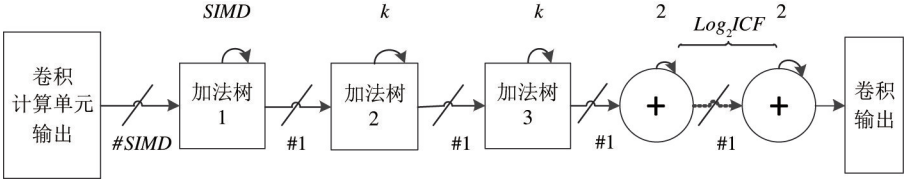

图 10　基于前向加法链的卷积累加器设计

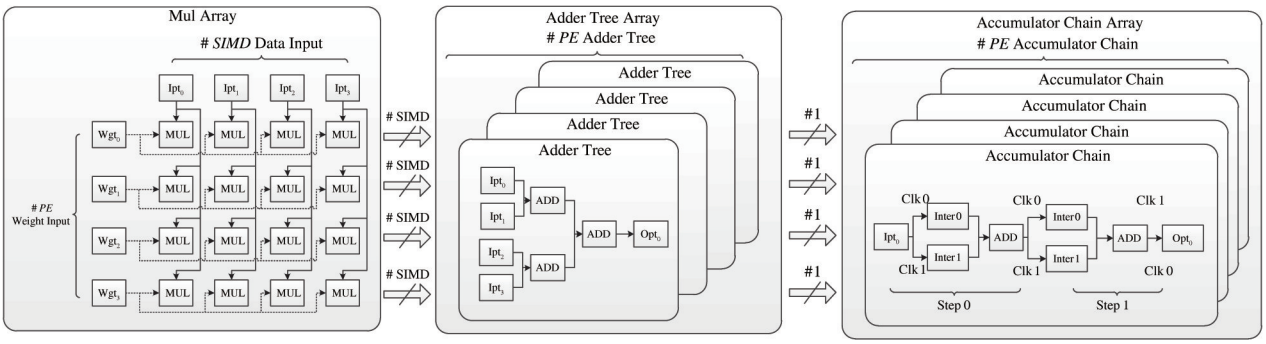

图 11　基于乘法矩阵与前向加法链的卷积计算阵列

3 部分:(1)乘法矩阵,完成“$PE \times SIMD$”个数的乘法运算;(2)加法树矩阵,将 $SIMD$ 个数汇聚成一个数;(3)加法链,完成同一个卷积计算的累加功能。基于前向加法链的卷积阵列设计,打破了浮点数累加的后向反馈,不依赖于加法器的具体延迟,通过延长流水线的方式达到了卷积计算的线速累加和输出。

(6)批归一化及非线性响应设计

批归一化及非线性响应均为面向每个处理元素单独处理,比较容易实现并行处理,多个并行处理之间不需要交互数据,具体的批归一化及非线性响应设计如图 12 所示。

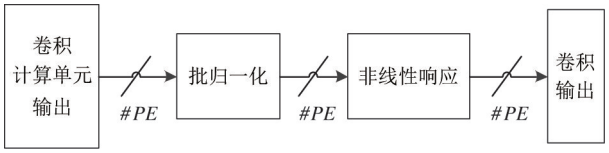

图 12　批归一化与非线性响应流程

批归一化的计算如式(8)所示。

$$y = alpha \times \frac{x - mean}{std} + gamma \tag{8}$$

其中,$alpha$、$gamma$、$mean$ 以及 std 均为批归一化参数,可以从模型权重中获取。

非线性响应部分支持 Leak ReLU 和 ReLU 函数,具体的计算公式如式(9)所示,Leak ReLU 的参数 $alpha$ 一般为 0.1,ReLU 的参数 $alpha$ 为 0。

$$y = \begin{cases} x & x \geqslant 0 \\ alpha \times x & x < 0 \end{cases} \tag{9}$$

2.3　基于数据流调度的池化层设计

池化层的输入一般为上一层的卷积计算输出,池化层的输出为下一层卷积计算的输入。基于数据流调度的池化层计算流程包括池化层输入调度以及池化层计算单元两部分。池化层计算的并行数 PE 与上一层卷积计算的 PE 保持一致。每个阶段的数据流均为 PE 个浮点数据。输入特征图与输出特征图仍然维持“$H \times W \times C$”的数据抽象,便于后续通过流数据接口输出给下一个卷积计算层。具体的池化层计算流程如图 13 所示。

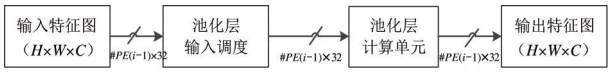

图 13　池化层计算流程

(1)池化层计算单元设计

根据卷积模型计算的需要,本部分池化层的设计只支持最大值池化功能。池化层计算单元输入为 PE 个卷积计算输出,通过与池化层的中间结果寄存器进行比较,如果比中间结果寄存器数据值大,则更新中间结果寄存器的存储值。经过一定次数的迭代,最终从中间结果寄存器输出到输出寄存器。具体的池化层计算单元如图 14 所示。

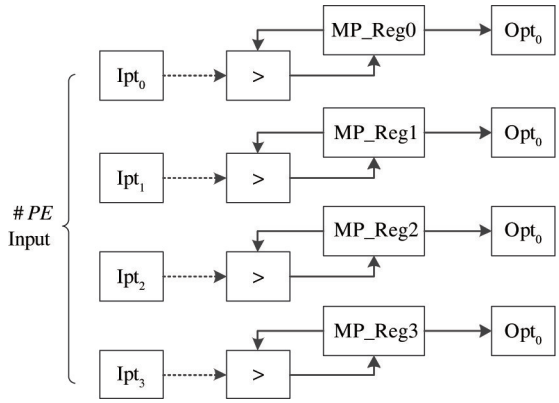

图 14　池化层计算单元

(2)池化层中间结果缓存设计

池化层中间结果缓存面向池化层的数据调度和中间结果缓存以及最终的池化输出,需要设计一个面向输入数据流的池化计算缓存。输出调度不能中断池化单元的正常计算流程,并且输入与输出同时维持统一的特征图数据流格式。

池化计算一般为"$k \times k$"的二维计算,而输入特征图数据是按照"$H \times W \times C$"的统一数据流格式输出的,这就需要设计一个多行缓存,能够无缝切换不同行的池化计算中间结果,且无需更改输入数据的"通道优先"的格式。

具体的池化层包含 *PE* 组缓存行,每行缓存有 *Pool _ Dim* 个缓存块,每个缓存块有 *PE _ Blocks* 个中间结果寄存器(见图 15)。*PE _ Blocks* 的计算如式(10)所示。计算不同行的池化过程,需要切换池化层的中间缓存,使得中间结果能够存储在合适的位置,*PE* 行池化数据并行处理,处理完后输出,宽度为 *PE* 个浮点数据流。

$$PE\_Blocks = IC/PE \tag{10}$$

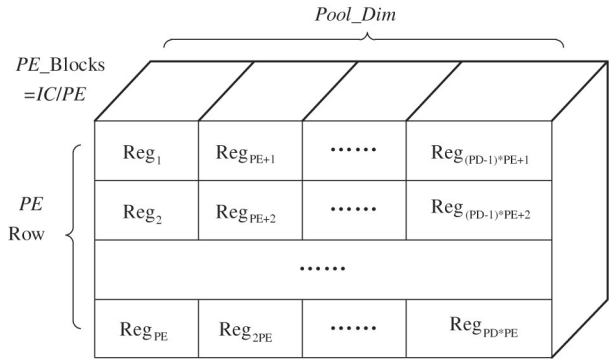

图 15　池化层中间结果缓存

3 实验结果和分析

3.1　加速器性能设计与分析

由于该加速器采用了基于数据流调度的多层卷积网络协同加速的模式构建长流水线,获得性能提升,因此每一个卷积模块的配置,即 *PE*、*SIMD* 的选值,影响该层的计算能力。同时多层的计算能力要相匹配,使得多层的处理能力大致相当而又避免过大的处理核由于流水线吞吐的限制,发挥不出应有的性能。本部分通过分析模型计算硬件预估性能,结合 HLS 高级综合资源占用分析,选取性能最高的 Conv10 _ Yolo 卷积加速器配置进行分阶段设计,以适应 FPGA 芯片的分超级逻辑区放置资源。

Conv10 _ Yolo _ FPS 2902 卷积网络加速器配置如表 2 所示,基于高级语言综合(HLS)设计的处理速度达到了 2902.56 fps,基于 HLS 设计的硬件卷积处理资源性能达到了 279.056 GFLOPS。

表 2　Conv10 _ Yolo _ FPS 2902 卷积网络加速器配置

名称	*PE*	*SIMD*	卷积循环次数	FPS (214 MHz)	HW MFLOPS (214 MHz)
cnv1	16	3	65 536	3265.38	20 544
cnv2	8	16	65 536	3265.38	54 784
cnv3	8	16	73 728	2902.56	54 784
cnv4	8	16	65 536	3265.38	54 784
mp4					0
cnv5	4	16	65 536	3265.38	27 392
mp5					0
cnv6	4	16	73 728	2902.56	27 392
cnv7	4	4	65 536	3265.38	6848
cnv8	4	2	65 536	3265.38	3424
cnv9	4	16	65 536	3265.38	27 392
conv _ detect	1	4	30 720	6966.14	1712
总计				2902.56	279 056

3.2　CNN 加速器资源占用与功耗分析

Conv10 _ Yolo 卷积网络加速器使用 HLS 形成处理核 IP 以及 Vivado 板卡仿真结合的方式获得最终可执行的比特流,Vivado _ HLS 版本为 2019.1。由于加速器核过大,超出了一个逻辑分区的可用资源总量,导致 Vivado 后续的综合实现的时序不能收敛,需要对加速器核进行切分,分成 3 个独立的 IP 核分别进行仿真分析,具体的资源占用对比如图 16 所示。3 个阶段的 IP 资源占用均不超过一个逻辑分区的资源总量,其中资源占用最多的是 Step1 的 DSP 资源,达到了 77%,各个阶段 IP 资源平均占用不超过 52%,有利于后续的综合实现时序收敛。

Vivado 版本为 2019.1,基于 Vivado 仿真工具的 Conv10 _ Yolo 卷积网络加速器资源占用分析如表 3 所示。资源占用量最高的为 DSP 资源,达到了55.92%,限制了加速器规模进一步扩大,具体的资源占比柱

状图如图 17 所示。

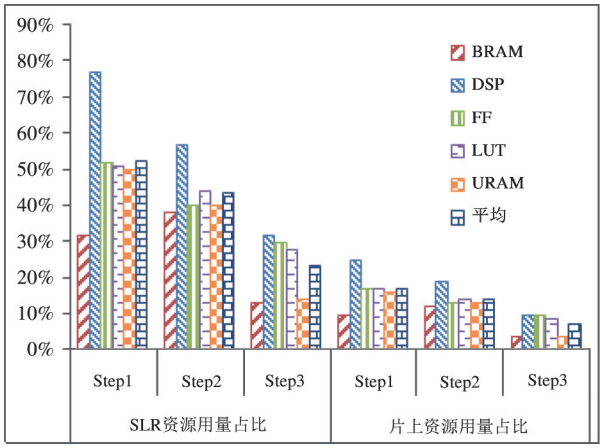

图 16　IP 核资源占用分析

表 3　基于 Vivado 的 Conv10 _ Yolo _ FPS 2902 卷积网络加速器资源占用表

名称	用量	资源总量	占比
LUT	491 666	1 182 240	41.59%
LUTRAM	69 475	591 840	11.74%
FF	871 826	2 364 480	36.87%
BRAM	355	2160	16.44%
URAM	334	960	34.79%
DSP	3825	6840	55.92%
IO	144	676	21.30%
GT	1	76	1.32%
BUFG	108	1800	6.00%
MMCM	3	30	10.00%
PLL	3	60	5.00%

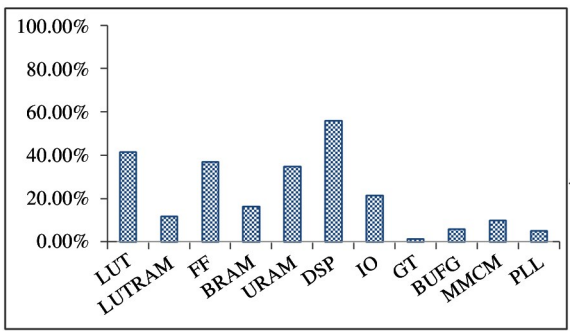

图 17　加速器资源占用分析

基于 Vivado 仿真工具的 Conv10 _ Yolo 卷积网络加速器功耗分析如表 4 所示,功耗为 19.078 W。

表 4　基于 Vivado 的 Conv10 _ Yolo _ FPS 2902 卷积网络加速器功耗分析

名称	片上功耗	结温
Conv10 _ Yolo	19.078 W	34.9 ℃

3.3　模型训练与性能结果

基于开源 Google Earth 卫星影像数据,本文收集了全球多个国家多时相的机场影像作为训练数据,采用人工标注的方式在机场图片中获得机场目标标注信息,总共标注了 506 张 128 ×128 像素尺寸的分辨率 8 m 左右的机场目标标注数据集。基于该数据集展开机场目标筛选实验。

实验 FPGA 硬件采用 Xilinx VCU 1525 开发板卡进行性能测试,在 FPGA 平台上达到了 2802.08 fps 的处理速率,折合有效的吞吐量为 247.496 GFLOPS,机场目标识别平均准确率达到 97.59%。具体的测试结果如图 18 和图 19 所示,其中灰色框为目标真实值,黑色框为检测结果,图示结果显示检测机场的准确率较高,达到了应用的需求。

3.4　与 GPU 处理卷积计算性能对比

实验硬件配置为 E5-2650-v4 CPU 以及 Nvidia Titan X GPU,软件配置为 Ubuntu14.04,Cuda 版本为 9.0.176,TensorFlow 版本为 1.5.1,Keras 版本为2.2.4。训练参数 Batch 为 16,迭代次数为 20 000 次,IOU 阈值设置为 0.3,框的概率阈值设置为 0.3。基于 GPU 平台测试机场目标识别能力,平均准确率达到 97.59%,基于 GPU 的处理速度为 126.01 fps。通过

图 18　机场目标识别结果 1

图 19　机场目标识别结果 2

FPGA 平台和 GPU 平台分别对 Conv10 _ Yolo 卷积网络加速器进行性能测试,结果如表 5 所示,该设计在 FPGA 平台上达到了 2802.08 fps 的处理速率,处理速度达到了 GPU 的 22 倍,功耗低于 GPU 处理平台。与大模型[15-16]相比,该设计处理速度更快,能耗更低。

3.5　与通用 FPGA 加速卷积计算性能对比

本文将Conv10_Yolo的性能与之前的工作对比,本文提出的 Conv10 _ Yolo 加速器吞吐量达到了 247.49 GFLOPS。文献[12,17]的工作采用了基于硬核浮点计算单元的 Intel Arria 10 器件,吞吐量较高,除了采用浮点硬核加速的工作,本文方法的运算吞吐量达到了最高值,为 247.49 GFLOPS;运算效能也达到了最高值,为 12.97 GFLOPS/W。相对于文献[11]、文献[18]、文献[19]、文献[20]的工作,本文方法使用更少的 DSP 资源达到了更高的运算吞吐量,有效解决了浮点卷积加速的线速瓶颈。实验数据表明,基于乘法矩阵与前向加法链的卷积计算阵列设计,解决了浮点卷积乘累加造成的线速吞吐瓶颈,提升了浮点卷积网络加速的处理速度与能效比,满足了星载、机载等功耗受限条件下进行高准确率目标检测的功耗、加速吞吐量等要求。

表 5　与 GPU 处理卷积计算性能对比

模型	平台	准确率	速度/fps	功耗/W
SSD[16]	GPU	77%	46	~150
YOLOv2[15]	GPU	76.8%	67	~150
Conv10 _ Yolo	GPU	97.59%	126.01	~150
Conv10 _ Yolo	FPGA	97.59%	2802.08	19.078

表 6　相关 FPGA 浮点加速卷积网络工作对比

名称	年份	位宽	描述语言	设备	主频/MHz	吞吐量/GOPs	功耗/W	LUT/K	DSP	DSP/GOPs	GOPS/W
文献[11]	2016	Float 32	HLS	Virtex 7 VX 485T	100	75.16	33.93	28	2695	35.85	2.22
文献[12]	2017	Float 32	OpenCL	Arria 10 GX 1150	370	866	41.7	437	1320	1.48	20.77
文献[13]	2016	Float 32	OpenCL	Stratix-V GXA 7	181	33.9	27.3	-	162	4.77	1.24
文献[14]	2017	Float 32	OpenCL	Arria 10 GX 1150	270.8	360.4	-	350	1290	3.57	-
文献[16]	2019	Float 32	-	ZC 706	150	22.74	-	145	596	27.09	-
文献[17]	2020	Float 32	Verilog	XCZU 9EG	100	42.13	24	124	784	18.60	1.75
文献[18]	2019	Float 32	HLS	ZYNQ 7100	100	17.11	4.083	142	1926	112.56	4.19
Conv10 _ Yolo (本文算法)	-	Float 32	HLS	VU + VU9P	214	247.49	19.078	491	3825	15.45	12.97

4　结 论

本文面向遥感影像机场目标识别应用,定制了一个轻量化的目标检测网络,提出了基于乘法矩阵与前向加法链的卷积计算阵列设计,解决了卷积加速的乘累加线速吞吐计算瓶颈。并通过基于 FPGA

的数据流调度技术,对该检测网络的浮点模型进行加速,达到了功耗受限环境下实时高准确率的计算需求,同时解耦合了软硬件协同加速的开发流程,降低了应用开发人员利用 FPGA 进行加速计算的门槛。通过在 Xilinx VCU 1525 开发板上构建实现了浮点卷积目标检测网络,在机场目标区域检测应用中平均准确率为 97.59%,目标检测处理速度为 2802 fps,功耗为 19.07 W,运算性能为 247.49 GFLOPS。实验结果表明,该方案达到了应用需求的高准确率,解决了浮点卷积加速的线速吞吐难点,计算吞吐量以及计算能效比达到了最优,同时功耗满足应用环境的需求。下一步可以就相关场景的细粒度目标识别加速问题进行研究。

参考文献

[1] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J]. *Proceedings of the IEEE*, 1998, 86(11): 2278-2324

[2] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[EB/OL]. http://arxiv.org/pdf/1409.1556.pdf: arXiv, (2014-09-04), [2021-01-05]

[3] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, USA, 2015: 1-9

[4] HOWARD A G, ZHU M, CHEN B, et al. Mobile nets: efficient convolutional neural networks for mobile vision applications [EB/OL]. http:// arxiv.org/pdf/1704.04861.pdf:arXiv, (2017-04-17), [2021-01-05]

[5] SANDLER M, HOWARD A, ZHU M, et al. Mobilenetv2: inverted residuals and linear bottlenecks[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, USA, 2018: 4510-4520

[6] HOWARD A, SANDLER M, CHU G, et al. Searching for mobilenet v3[C]// Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 2019: 1314-1324

[7] HUBARA I, COURBARIAUX M, SOUDRY D, et al. Quantized neural networks: training neural networks with low precision weights and activations[J]. *The Journal of Machine Learning Research*, 2017, 18(1): 6869-6898

[8] CAI Z, HE X, SUN J, et al. Deep learning with low precision by half-wave Gaussian quantization[C]// Proceedings of the IEEE Conference on ComputerVision and Pattern Recognition, Honolulu, USA, 2017: 5918-5926

[9] 陈朋, 陈庆清, 王海霞,等. 基于改进动态配置的 FPGA 卷积神经网络加速器的优化方法[J]. 高技术通讯, 2020, 30(3):32-39

[10] JACOB B, KLIGYS S, CHEN B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, USA, 2018: 2704-2713

[11] RAHMAN A, LEE J, CHOI K. Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array[C]//2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2016: 1393-1398

[12] ZHANG J, LI J. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network [C]// Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, USA, 2017: 25-34

[13] WANG D, AN J, XU K. PipeCNN: an OpenCL-based FPGA accelerator for large-scale convolution neuron networks[EB/OL]. http:// arxiv.org/pdf/ 1611.02450.pdf:arXiv, (2016-11-08), [2021-01-05]

[14] KENDALL A, GAL Y. What Uncertainties do we need in Bayesian deep learning for computer vision? [EB/OL]. http:// arxiv.org/pdf/ 1703.04977v1.pdf: arXiv, (2017-03-15), [2021-01-05]

[15] REDMON J, FARHADI A. YOLO9000: better, faster, stronger[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, USA, 2017: 7263-7271

[16] LIU W, ANGUELOV D, ERHAN D, et al. SSD: single shot multibox detector[C]// European Conference on Computer Vision, Amsterdam, Netherlands, 2016: 21-37

[17] WEI X, YU C H, ZHANG P, et al. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs[C]// Proceedings of the 54th Annual Design Automation Conference, Austin, USA, 2017: 1-6

[18] 蹇强, 张培勇, 王雪洁. 一种可配置的 CNN 协加速器的 FPGA 实现方法[J]. 电子学报, 2019, 47(7): 1525-1531

[19] 赵烁, 范军, 何虎. 基于 FPGA 的 CNN 加速 SoC 系统设计[J]. 计算机工程与设计, 2020, 41(4):939-944

[20] LIU B, ZOU D, FENG L, et al. An FPGA-based CNN accelerator integrating depthwise separable convolution [J]. *Electronics*, 2019, 8(3): 281

Convolutional neural network optimization method for object detection

ZHANG Zhichao[*,**,***,****], WANG Jian[*,**,***], ZHANG Longbing[*,**,***], XIAO Junhua[*,**,***]

([*] State Key Laboratory of Computer Architecture, Institute of Computer Technology, Chinese Academy of Science, Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] University of Chinese Academy of Sciences, Beijing 100049)

([****] The 15th Research Institute of China Electronics Technology Group Corporation, Beijing 100083)

Abstract

Aiming at the requirements of high accuracy, low power consumption and high throughput in remote sensing image object detection on space-borne and other power-constrained platforms, this paper proposes an optimization method of field programmable gate array (FPGA) convolutional neural network (CNN) for object detection. Data stream scheduling technology and convolutional computing array design based on multiplication matrix and forward addition chain are used to accelerate the floating point convolutional neural network model. A floating point convolution object detection network is implemented on FPGA development board by using this method. In application, the accuracy is consistent with the original model, the average accuracy is 97.59%, and the throughput is 22 times that of Titan X. Compared with similar FPGA accelerated floating-point convolution methods, the throughput and energy efficiency ratio of the proposed method are optimal. The experimental results show that the proposed method breaks through the line-speed throughput difficulty of floating-point convolution acceleration, and solves the balance of power consumption, accuracy and throughput in the application.

Key words: convolution neural network (CNN), field programmable gate array (FPGA), data stream scheduling, object detection, acceleration