Journal of Computer Applications ISSN 1001-9081 2019-01-10
计算机应用,2019,39(1):192 - 198 CODEN JYIIDU http://www.joca.cn

文章编号:1001-9081(2019)01-0192-07 DOI:10.11772/j.issn.1001-9081.2018051134

基于小型 Zynq SoC 硬件加速的改进 TINY YOLO 实时车辆检测算法实现

张雲轲,刘 丹*

(电子科技大学 电子科学技术研究院,成都 611731)

(* 通信作者电子邮箱 yunke.zhang@outlook.com)

摘 要: 针对 TINY YOLO 车辆检测算法计算量过大,且在小型嵌入式系统中难以达到实时检测要求的问题。利用小型 Zynq SoC 系统的架构优势以及 TINY YOLO 的网络权值中存在大量接近零的权值参数这一特点,提出硬件并行加速的改进算法,称为浓缩小型深度网络(Xerantic-TINY YOLO, X-TINY YOLO)车辆检测算法。首先对 TINY YOLO 中网络结构进行压缩;其次采用高效多级流水线、流水线内全并行的方式对卷积计算部分进行算法加速;最后提出与网络结构相配合的数据切割和传输方案。实验结果表明,X-TINY YOLO 仅消耗 50% 的片内硬件资源,可在相对于 GPU 和 CPU 性价比更高更适合嵌入式场景的 Zynq SoC 系统上实现,且其检测速度达到 24 帧/s,满足车辆检测的实时性要求。

关键词: 车辆检测;机器视觉;TINY YOLO;Zynq-7020;硬件加速

中图分类号: TP389.1; TP391.413 **文献标志码:** A

Real-time implementation of improved TINY YOLO vehicle detection algorithm based on Zynq SoC hardware acceleration

ZHANG Yunke, LIU Dan*

(*Research Institute of Electronic Science and Technology, University of Electronic Science and Technology of China, Chengdu Sichuan* 611731, *China*)

Abstract: TINY YOLO (TINY You Only Look Once) vehicle detection algorithm requires much amount of calculation which makes it difficult to achieve real-time detection in small embedded systems. Because plenty of zero values exist in a network weight matrix which makes the network a sparse structure, an improved version of TINY YOLO vehicle detection algorithm, called Xerantic-TINY YOLO (X-TINY YOLO), was proposed and accelerated in parallel way using architectural advantages of small Zynq SoC system. Original network structure of TINY YOLO was compressed and the operations of convolution steps were accelerated in parallel by using high efficient multistage pipeline. All multiply-add operations were concurrently executed within each stage of pipeline. By matching network structure, a method of data segmentation and transfer was also proposed. The experimental results show that, X-TINY YOLO only consumes 50% hardware resources on chip, and it can be implemented on small Zynq SoC systems which have higher performance-price ratio than GPU and CPU and is suitable for embedded implementation scenes. Its detection speed reaches 24 frames per second, which meets the requirement of real-time vehicle detection.

Key words: vehicle detection; machine vision; TINY You Only Look Once (TINY YOLO); Zynq-7020; hardware acceleration

0 引言

实时车辆检测算法主要分为三类:基于先验知识的车辆检测算法[1-2]、基于浅层机器的学习车辆检测算法以及基于深度学习的车辆检测算法[3]。

基于先验知识的检测算法是根据车身所带有的线条、阴影或边缘特征对前方车辆进行识别。文献[4]通过对车辆尾部特定水平直线(如保险杠、后车窗下边沿等)的检测达到车辆检测目的。该算法易将车身周边环境中存在的类似直线误判为车辆,且对处于转弯状态的车辆会出现检测丢失现象。文献[5]中提出对刹车灯进行检测的方法实现车辆夜间检测,效果较好但无法应用于外部光强较好的白天或地下停车场等环境。行驶中车辆周边环境的多样性常常会以噪声形式干扰此类检测方法准确率,故此类方法难以在环境变化剧烈的使用场景中适用。

基于浅层机器学习的车辆检测算法以先验知识类算法中提取的车辆特征为基础,结合机器学习算法实现车辆的检测。文献[6]中提出 Haar-like 与 Online Boosting 相结合的方式,使用大量特征集对网络进行训练使其能检测车辆,与单纯的先验知识检测相比该算法在各种环境中的鲁棒性有了显著提高。文献[7]对 Haar-like 特征进行了优化,提升了该特征对环境中光强变化的适应性。该类算法特点是检测效率高、复杂度低,但依然严重依赖特征选取,使得其在复杂且变化的路况场景下重建模工程量大。

收稿日期:2018-06-01;修回日期:2018-06-21;录用日期:2018-06-28。

作者简介:张雲轲(1993—),男,四川成都人,硕士研究生,主要研究方向:人工智能、高性能计算、深度神经网络、模式识别; 刘丹(1969—),男,四川成都人,副教授,博士,主要研究方向:网络安全、Web 数据挖掘、图像处理。

基于深度学习的车辆检测算法多以卷积神经网络为基础加以改进,文献[8]中提出的 Faster R-CNN(Faster Region proposal Convolutional Neural Network)通过聚类方式对车辆尺寸进行分类,随后使用高分辨率图片作为网络输入,最后在 KITTI 数据集上获得了较高的车辆检测精度,但其卷积层数过多导致网络计算复杂度过大而难以实现实时性检测。在网络模型的小型化方向上,Facebook 的人工智能实验室设计了更轻巧的模型 YOLO(You Only Look Once)[9],其最快版本的 TINY YOLO 在 GPU 上的测试速度可以达到每秒 200 帧。各版本 YOLO 算法的出现为深度学习类的目标检测算法在实际运用场景的实时性检测提供了可能。

深度学习算法的硬件加速方案主要有以下四种: 图形处理器(Graphic Processing Unit, GPU)、专用集成电路(Application-Specific Integrated Circuit, ASIC)、粒度可重构阵列和现场可编程门阵列(Field-Programmable Gate Array, FPGA)。

利用 GPU 数千个计算核心并行计算的优势,文献[10-12]中提出的 GPU 加速方案,相比 CPU 上实现的深度学习算法在运算速度和功耗方面均有较大提升,然而中低端 GPU 多核之间共享存储(share memory)资源少难以完成算法移植,而高端 GPU 芯片昂贵的价格难以在短时间内为工业级应用所接受。文献[13]基于深度神经网络结构特点设计的由 64 块 ASIC 芯片组成的超级加速器 DaDianNao 在加速性能方面是 GPU 的 450.65 倍,功耗仅为 CPU 的 0.67%,性能达到极致; 然而各版本神经网络的网络结构间存在较大差异,而 ASIC 专用芯片的可变性较差,故其较适用于作为某些结构固定的网络的专用加速器。文献[14]基于单指令多数据流(Single Instruction Multiple Data, SIMD)架构,引入 FPGA 可编程逻辑电路,实现了可重构硬件的神经网络加速方案。其与文献[15-16]中提出的基于 FPGA 架构的加速器均可提供多种神经网络的实现方案,然而其缺少核心处理器,在实现较复杂的分支任务繁多的算法上有一定难度。

使用 Zynq SoC 架构对深度神经网络算法进行加速的研究还较少。Zynq SoC 在架构方面结合了 ARM(Acorn RISC Machine)嵌入式处理核心和 FPGA 可编程逻辑电路两方面的优势,提供较强的流程控制的同时可实现多个高速并行的乘加运算,而卷积神经网络的主要计算量集中于卷积层中的矩阵乘加运算,故其非常适用于卷积神经网络的检测算法加速,且该系列芯片在价格上相比 FPGA 和 GPU 均有较大优势。

综上所述,本文针对实际应用场景中车辆检测算法对实时性要求高这一特点,选择 YOLO 算法的快速版本 TINY YOLO 作为算法基础,并对网络作出裁剪优化进一步降低算法复杂度; 再选择相比 GPU 等硬件平台性价比更高的小型 Zynq SoC 平台作为加速算法的硬件平台; 通过 Zynq SoC 平台内的数字信号处理(Digital Signal Processing, DSP)并行计算资源和流水线技术,最终在千元以内的硬件平台上实现车辆实时检测算法。

1　X-TINY YOLO 检测算法描述

1.1　网络结构压缩

TINY YOLO 检测算法是 YOLO V2 目标检测算法的简化网络,相比原 YOLO V2 算法而言,其运算量减少,伴随着精度的降低,但作为在系统内部硬件资源极度有限的小型 Zynq SoC 中实现的嵌入式算法,TINY YOLO 依然存在着网络规模过大导致计算量过大的缺点,从而使得其在实际应用场景下检测的实时性无法得到满足,故算法中首先需考虑对 TINY YOLO 网络进行裁剪。

TINY YOLO 网络中包含三种传递结构: 卷积层、全连接层和最大池化层。卷积层的计算过程如图 1 所示。

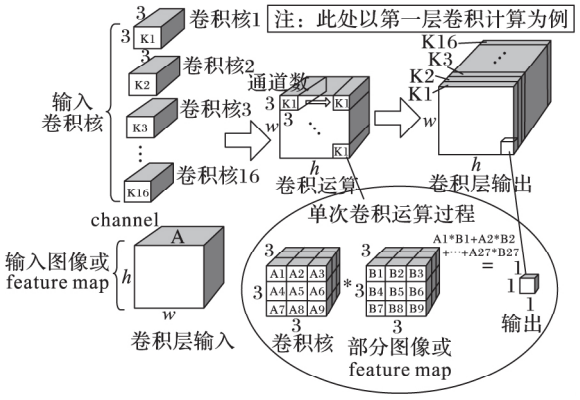


图 1　单层卷积计算过程

Fig. 1　Calculation processing of single layer convolution

feature map 是每一层的输入特征矩阵,由图 1 可见卷积运算中存在大量的矩阵乘加运算,所以耗时最多。又因为 Zynq SoC 系统中有可用于乘加运算加速的专用数字信号处理单元 DSP48E 硬 IP 核资源,可对乘加运算进行并行计算加速,故考虑利用 FPGA(简称为 PL 端)资源对卷积层进行并行计算加速。最大池化层是滑动地取 2×2 网格中的最大值,可以紧接在卷积乘加运算之后而无需耗费更多的硬件资源。对于全连接层,虽然计算量较大,但在 TINY YOLO 的 16 层网络结构中只有最后一层是全连接层,若移植至 FPGA 端(PL 端)将占用较多资源且对速度提升不明显,故选择将全连接层直接放在 ARM 端(简称为 PS 端)串行运行。

通过以上分析发现,对 TINY YOLO 网络进行移植时的主要计算量源于卷积层中的大量乘加运算,故考虑裁剪卷积层从而减少计算规模。通过观察 TINY YOLO 网络训练后的卷积核中的权值发现,该网络第 3 到第 9 层卷积层中存在着大量接近 0 的权值。通过对比实验发现,这样的网络稀疏性会带来少量的网络性能提升,但伴随着成倍的计算量的增加(表 1)。故尝试在性能不会有过大减小的情况下,在网络训练前对 TINY YOLO 第 3 到第 9 层卷积层的输入卷积核个数进行裁剪,从而使得重新训练后得到的新网络(X-TINY YOLO)中接近于 0 的权值个数减少,最后得出缩小规模后的网络权值用于嵌入式移植。

裁剪后 X-TINY YOLO 网络结构下新网络权值的训练方法是先用 ImageNet 数据集和 MS COCO2007+2012 数据集对网络进行预训练,待网络参数稳定后再用收集和标定的2 000 张图片集(行驶车辆前方的路况图片),对网络后 3 层负责分类的网络进行再训练,从而进一步提升网络对具体的车辆检测任务数据的拟合效果。训练后得出的 X-TINY YOLO 网络

与原 TINY YOLO 网络的各卷积层输入 feature map 的结构的对比如图 2。

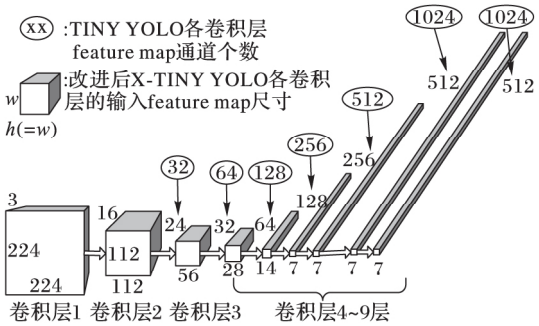


图 2 TINY YOLO 网络结构与 X-TINY YOLO 网络各卷积层的输入尺寸结构对比

Fig. 2 Comparison of input size structure of each layer between TINY TOLO and X-TINY YOLO

如图 2 所示，在训练时对上述层的输入卷积核数量进行了控制，从而减少了对相应卷积层输入的 feature map 通道个数，其中第二卷积层的训练所用卷积核个数由原来的 32 个减小为 24 个，之后每一个卷积层训练所用卷积核个数减半。网络理论总计算量如式(1)：

$$total\_calculation = \sum_{layers}(w \cdot h \cdot k\_size^2 \cdot in\_c \cdot out\_c) \tag{1}$$

其中：w、h 和 in_c 分别表示单层卷积层输入 feature map 的宽、高和通道数，k_size^2 表示卷积核二维尺寸面积，out_c 表示卷积核数量。

训练后的网络性能及计算量对比统计如表 1，其中 mAP(mean Average Precision)为平均精度指标。

表 1 网络裁剪和重新训练后性能对比

Tab. 1 Performance comparison after network cutting and retraining

检测网络	mAP/%	计算量/10^6
TINY YOLO 网络	67	592.372
X-TINY YOLO 网络	62	304.217

TINY YOLO 网络中引入了 Batch Normalization(BN)的操作来解决训练中的梯度爆炸和收敛速度慢等问题，故在 Zynq SoC 平台实现检测网络的前向传播时，每一层的卷积运算之后还需要进行 BN 操作，其输出 *output* 的具体公式如式(2)所示：

$$\boldsymbol{output} = \frac{\boldsymbol{w} \otimes \boldsymbol{d} - m}{v} \cdot \boldsymbol{s} + \boldsymbol{b} \tag{2}$$

其中：$\boldsymbol{w}$ 为该层的权值矩阵，$\boldsymbol{d}$ 为该层输入的 feature map 矩阵，m 为单张 feature map 均值，v 为单张 feature map 方差，$\boldsymbol{b}$ 为偏移向量(bias)，$\boldsymbol{s}$ 为缩放系数向量(scale)。若每一次的结果都进行这样的运算，将在检测环节耗费过多时间，而这些参数中，$\boldsymbol{w}$、m、v、$\boldsymbol{s}$、$\boldsymbol{b}$ 通过网络训练后都成为已知参数，即在网络初始化时已知且不会再改变的数值，仅 $\boldsymbol{d}$ 需要用到上一层的计算结果。故对式(2)进行化简得到式(3)：

$$\boldsymbol{output} = \frac{\boldsymbol{w} \cdot \boldsymbol{s}}{v} \otimes \boldsymbol{d} - \frac{m \cdot \boldsymbol{s}}{v} + \boldsymbol{b} = \bar{\boldsymbol{w}} \otimes \boldsymbol{d} - \bar{\boldsymbol{b}} \tag{3}$$

按照式(3)在网络初始化时提前对参数进行处理，之后网络前向传播时用新生成的 $\bar{\boldsymbol{w}}$ 和 $\bar{\boldsymbol{b}}$ 替代原权值矩阵 $\boldsymbol{w}$ 和偏移 $\boldsymbol{b}$ 进行运算。通过这种方法，用网络初始化时间增加的成本换取了实时检测计算时间的缩减。

1.2 数据分割算法

由于 X-TINY YOLO 算法中存在大量卷积运算，即矩阵乘法运算(如图 1)，若将大量乘加运算放置在 Zynq SoC 平台的 FPGA 端(PL 端)进行，就可利用多片数字信号处理单元——DSP48E 内核对多个乘加运算并行从而提升计算效率，但小型 SoC 平台包含的 DSP48E 和块随机存储单元(Block RAM，BRAM)的数量有限，远远小于 X-TINY YOLO 每一层的计算及存储所需要的资源数量，例如图 2 中第一层卷积层的输入图片尺寸是 $224 \times 224 \times 3$，输入权值规模是 $3 \times 3 \times 3 \times 16$，若一次性传入 PL 端则共需要 150 960 个 BRAM 存储单元，之后进行 $224 \times 224 \times 16 \times 3 \times 3 \times 3$(共 21 676 032)次乘加运算，而以 Xilinx 公司旗下 Zynq-7020 系列中的 XC7Z020 芯片为例，其 PL 端仅仅有 4.9 MB(最多可分为 280 片)BRAM 存储单元和 220 个 DSP48E 片上 IP 核，只可同时并行 220 个 25×18 位的乘加运算，远远小于该层卷积计算全并行所需资源。

故考虑根据每一层的网络结构，将单层卷积分解为多次进行，即将无法在 PL 端一次性完成的大矩阵乘法运算进行运算拆分后分多步完成。该方法首先需要对存储于 PS 端的第三代双倍数据率同步动态随机存取存储器 DDR3 SDRAM(Double-Data-Rate Three Synchronous Dynamic Random Access Memory)中的单层网络的权值矩阵和该层图片像素矩阵(第二层后称为 feature map 矩阵)进行拆分。拆分后的单个小矩阵尺寸由 X-TINY YOLO 网络结构和 PL 端硬件资源允许的单次最大并行计算量共同决定。单层卷积中上述二矩阵未拆分前在 PS 端 DDR3 中的存储示意图如图 3～4 所示。

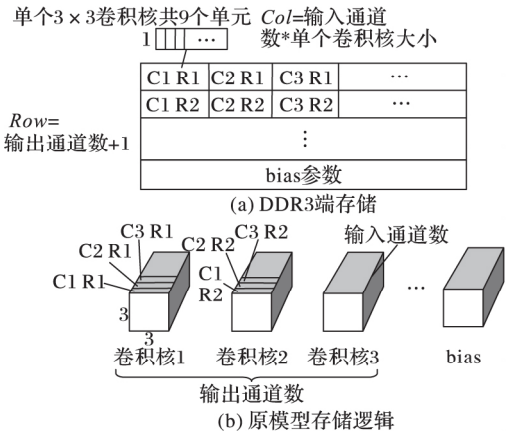


图 3 DDR3 中权重矩阵的存储与原模型逻辑的比较

Fig. 3 Comparison of storage of weight matrix between DDR3 and original model logic

图 3 和图 4 中的(a)图分别表示两矩阵在 PS 端 DDR3 中的存储方式，而图(b)分别展示两矩阵在 X-TINY YOLO 中的逻辑对照模型。可以看出，单层卷积运算中权重矩阵需要传入 PL 端的数据总量 *weights_trans* 计算如式(4)：

$$weights\_trans = out\_c \cdot in\_c \cdot k\_size^2 \tag{4}$$

由式(3)知，还需要传入偏移向量(bias)，它只用保存在权值矩阵的最后一行并随其一同传输即可，其传输大小为

bias_trans 计算如式(5):

$$bias\_trans = out\_c \cdot k\_size^2 \tag{5}$$

同时需要传入的 feature map 矩阵的传入总量 *feature_map_trans* 计算如式(6):

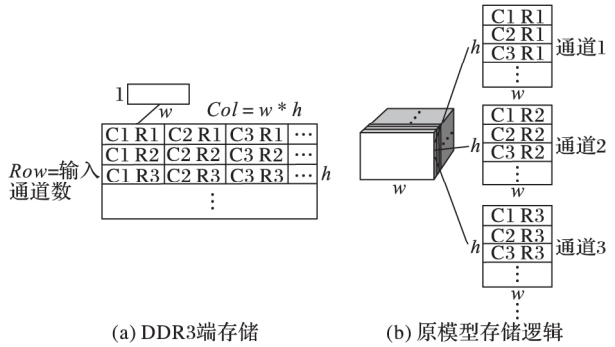
$$feature\_map\_trans = w \cdot h \cdot in\_c \tag{6}$$

其中: in_c、w、h 分别为输入的 feature map 的通道数量、宽度和高度; out_c 为输出 feature map 的通道数量,也是输入的权重矩阵中卷积核的个数; k_size 为卷积核尺寸,除最后一层卷积使用 1 × 1 卷积核之外,其他各层卷积核尺寸都是 3 × 3。

以下根据网络结构核硬件平台资源设计权值矩阵的分割方法,观察式(4)发现,k_size 除第 9 层外始终是 3,而 in_c 除第 1,3 层外,随层数增加以 16 为基数成倍增加,$weights\_trans$ 都有公约数 3 × 3 × 16,即 144。故决定对 DDR3 中权值矩阵(图 4(a))进行按列切割,切割后每列宽度为 144 或 144 的倍数(取决于硬件资源限制),单次传入的分割后权值矩阵大小(*Weights_trans_once*)为如式(7)所示:

$$Weights\_trans\_once = 144 \cdot \lfloor m/144 \rfloor \cdot out\_c \tag{7}$$

其中: m 为 Zynq SoC 平台的 PL 端允许的单次最大并行乘加运算个数,n 为双端口 BRAM 存储单元个数,实际小型 Zynq SoC 平台各系列芯片的 n 是略大于 m 的,m 约为 n 的 66% ~ 90%,故 m 成为计算瓶颈。



(a) DDR3端存储　　(b) 原模型存储逻辑

图 4　DDR3 中 feature map 矩阵的存储与原模型逻辑的比较

Fig. 4　Comparison of storage of feature map matrix between DDR3 and original model logic

对应模型中的逻辑意义为: 在本层所有卷积核的相同位置截断,每次运算同时向 PL 端传入 out_c 个卷积核在相同位置的 144 · ⌊m/144⌋ 个数据,即截取该层所有卷积核的连续 16 · ⌊m/144⌋ 个通道的所有权值。其中的特殊情况为第 1,3,9 层,由于通道数小于 16 或 k_size 为 1 导致列数不足 144。采取的措施是在初始化时,依然按单次传入 144 × out_c 个值,不足 144 按的列数在 DDR3 端存储时先补 0。例如,图 4 中若 DDR3 端存储的是第一层卷积层的权值矩阵,即 16 行 3 × 3 × 3 列的数据,则在每一行末尾补 144 − 3 × 3 × 3 个 0,使得每行的个数为 144。

feature map 矩阵的切割方式需要与权值矩阵的切割方式相匹配以适应计算需要。在权值矩阵切割时选择按列切割,即对 in_c 切割,而计算时需要传入将会与切割后的权值矩阵作卷积运算的 feature map 对应部分,故对其 in_c 按行切割,在 DDR3(图 5(a))中表现为按行切割,切割后每份的行数为 $144/k\_size^2 \cdot \lfloor m/144 \rfloor$。单次传入的分割后 feature map 矩阵大小(*feature_map_trans_once*)如式(8)所示:

$$feature\_map\_trans\_once = 144/k\_size^2 \cdot \lfloor m/144 \rfloor \cdot w \cdot h \tag{8}$$

对应模型中的逻辑意义为一次性传入多个大小为 $w \times h$ 的完整的通道数据,但由于前三层的 w,h 过大导致整行数据无法一次性传入,故需要对前三层按行切割后再按列切割。对于第 1、2 层,因为 in_c 很小不需要按行切割,只需对其矩阵按列切割。第 3 层需要同时按行和列切割,而之后的 4 ~ 9 层只需要按行切割,但按列切割操作会使得一次传入的单通道数据不完整,而进行卷积运算时又需要用到 feature map 中一个单元及周边的 3 × 3 个数据,使得在对边缘数据卷积时缺少 w 列数据(也缺少 w 行,但这部分是补零操作而非用到 feature map 矩阵中的数据,将放在 PL 端补零)。故为满足边缘部分卷积对数据的要求,对于 feature map 中进行过按列切割的 1,2,3 层数据进行补传操作: 若是该列切割中传入的第一块数据或最后一块数据,则在执行卷积操作时需要额外传入 w 列; 若是非首尾块,则需额外传入 2 × w 列。具体切割方式的示意图如图 5 所示,图中 $mul = \lfloor m/144 \rfloor$。

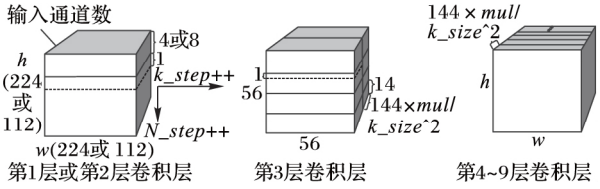


图 5　Feature map 矩阵的切割方式

Fig. 5　Slicing method of feature map matrix

按照上述方式拆分后的两个大矩阵被分割为存在计算匹配关系的若干小矩阵,之后成对传入一个权值小矩阵和一个 feature map 小矩阵供 PL 端计算单元使用,单次计算完成后立即传回 PS 端存储,所有小矩阵乘加计算结束后,结果将按照 X-TINY YOLO 该层通道顺序存储于 DDR3 中,组成下一层卷积运算的 feature map 输入。

1.3　数据传输、在 PL 端的存储和计算

数据传输时采用加速一致性接口(AXI-Accelerator Coherency Port,AXI_ACP)结合分散—集中型直接内存读取模式(Scatter-Gather Direct Memory Access,DMA_SG)进行数据搬移,将切割后小矩阵组从 PS 端的 DDR3 中分多次搬运至 PL 端的 BRAM 中。DMA_SG 方式适合传输数据量大于300 B 的连续或非连续数据块的搬运,高效且灵活。高速一致性 AXI_ACP 接口使得 PL 端可直接读取 L1 和 L2 两级 cache 中缓存的 PS 端数据,降低传输延迟。

由于 AXI_ACP 口的位宽限制,分配 4 个 8 bit 的 AXI_ACP 端口给权值矩阵的传输,保证在同一传输周期 PL 端可同时从 PS 端 DDR3 内搬移 4 组 char 类型的权值矩阵数据到 BRAM,并存入 4 个不同的 BRAM 中,直至单次权值小矩阵传输完成。单次小矩阵共需传输数据量计算公式如式(7),结合以上分析知,单次权值小矩阵传输消耗时钟周期(*weight_trans_clock*)如式(9):

$$weight\_trans\_clock = 144 \cdot out\_c/4 \cdot \lfloor m/144 \rfloor \tag{9}$$

分配 1 个 8 bit 的 AXI_ACP 对 feature map 数据传输,单个

feature map 小矩阵共需传输数据量如式(8),单次 feature map 小矩阵传输消耗时钟周期(*weight_trans_clock*)如式(10):

$$weight\_trans\_clock = 144/k\_size^2 \cdot w \cdot h \cdot \lfloor m/144 \rfloor \quad (10)$$

但对于第一到第三层卷积,传入计算的是非完整通道数据(如图 5),计算时还需要用到上一次硬件函数输出的部分卷积运算结果,故分配 4 个 8 bit 的 AXI_ACP 用于传入上一个小矩阵运算后的部分结果。最后剩下 1 个 8 bit 位宽 AXI_ACP 用于在 PL 端单次计算得到的新 feature map 矩阵传回 PS 端使用,因为结果的传输过程是紧跟在计算完成之后放在计算流水线之内的,每完成一次计算传回一个结果,只会增加单级流水线的延迟,并不会增加流水线级数,而当启动间隔(Initiation Interval, II)为 1 且流水线级数远远大于单级延迟时,因此增加的少量额外时间可忽略。

PL 端的数据存储使用 BRAM 存储单元,Zynq SoC 平台允许对 BRAM 进行分割,每一个分割后的小 BRAM 都具有双端口读写功能,故考虑传入一对乘数(即一对 feature map 和权值数据)存入分割后同一个 BRAM 单元,以保证可在同一时钟周期内同时读写多组数据,从而极大提高了数据吞吐量。BRAM 分割个数应取值为最大单次允许并行乘加个数,并留出富余的 2 个 BRAM 供最大池化操作使用。

计算时采用多级流水线、单级流水线内全并行的方式。因为该流水线的每一级均是从不同的 BRAM 中取数据并使用不同的 DSP48E 运算单元计算(或相同 DSP48E 中不同级单元),故流水线不存在结构冒险和数据冒险,故可实现流水线 $II = 1$ 的高效流水线。单个卷积层乘加运算消耗总时钟周期为:

$$calculation\_clock = in\_c \cdot out\_c \cdot k\_size^2/144 \cdot w \cdot h/\lfloor m/144 \rfloor \quad (11)$$

2 算法实现

2.1 系统总体设计

硬件结构如图 6 所示,Zynq SoC 平台主要由 ARM(简称 PS 端)和 FPGA(简称 PL 端)两部分构成,PL 和 PS 端分别有各自的存储单元 BRAM 和 DDR3。

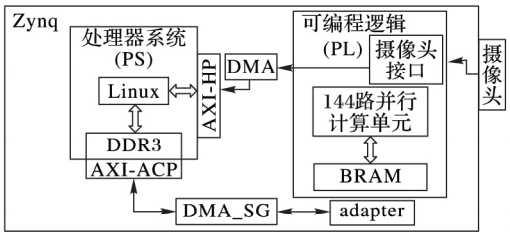


图 6 系统总体结构

Fig. 6 Overall structure of system

PL 端的并行摄像头接口逐帧接收摄像头传入的图像数据并传输给 PS 端的 DDR3 中存储。PL 端的多路并行计算单元由多片 DSP48E IP 核构成,用于并行地加速乘加运算。

算法现实的物理平台核心为 Zynq-7020 的 XC7Z020 芯片,其片内只有 220 个 DSP48E Slice,每片规格为 18×25 的乘法器。片内有 140 片双端口 BRAM,每片存储空间为 36 Kb,实际可分为 280 片 18 Kb 存储,其中每片系统预留 2 Kb 不可用空间,故每片可用 16 Kb,即 2 KB 空间。总可用空间为 280×2 KB,共可存储 280×2 048 个 8 bit 数据。

2.2 数据切割、存储、传输和计算

由第 1 章分析知,单次传入计算的矩阵规模上限将受限于 PL 端硬件存储资源(BRAM)和计算资源(DSP48E)数量。故需按照式(7)、(8)矩阵切割方法,对 feature map 矩阵和权值矩阵进行切割,由硬件资源知公式 $m = 220$,$\lfloor m/144 \rfloor = 1$ 中,单次最多并行计算 144 个乘加运算。故单次传入权值小矩阵大小为 $144 \cdot out\_c$,单次传入 feature map 小矩阵大小为 $144/k\_size^2 \cdot w \cdot h$。X-TINY YOLO 所有卷积层传入 PL 端的分割后矩阵规模及传入次数统计如表 2。

表 2 权值矩阵和 feature map 切割后规模及传输次数统计

Tab. 2 Statistics of size of block and transmission times after slicing feature map and weight matrix

卷积层标号	*k_steps*	*n_steps*	feature map 小矩阵大小	权值小矩阵大小	单层理论耗时/ms
1	1	56	576	2 688	8.15
2	1	14	864	14 336	4.20
3	2	4	1 152	12 544	2.20
4	2	1	2 304	12 544	0.96
5	4	1	4 608	3 136	0.98
6	8	1	9 216	784	1.33
7	16	1	18 432	784	5.20
8	32	1	18 432	784	10.40
9	4	1	1 080	7 056	0.26

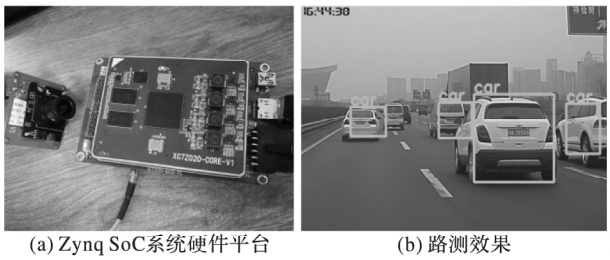
表 2 中,*k_step*、*n_step* 分别为 feature map 和权值矩阵分割后需分多次传入 PL 端的小矩阵数目。需要注意的是权值矩阵在传输当前层最后一个小矩阵时会多增加一行 bias 数据,而 feature map 矩阵在第 3 和第 9 次传入时,因为 *in_c* 无法被 16 整除,最后一次会传入规模比表格中数值更小的矩阵。以上两点未体现在表格中。

数据在 PL 端的存储方面,定义一个大小为 146×2 048 的 BRAM,并对第一个维度进行分割,分割为 146 个小 BRAM,其中 144 个 BRAM 存储两个计算所用矩阵,2 个 BRAM 用于最大池化操作。按照 1.3 节的存储方法,在每个 BRAM 的前 $w \times h$ 个单元中存入一个通道的 feature map 数据,从 $w \times h$ 开始之后的单元内存入 *out_c* 个单元的权值数据,在每个乘法计算周期内同时从 144 个 BRAM 中读出 144 对 feature map 和权值数据并分别送入 144 片 DSP48E 计算(综合后电路实际只使用了 110 片)。DSP48E 只有 144 片,按上述方式单个计算周期内只能实现 $144 \cdot (16 \cdot k\_size^2)$ 个乘加运算并行,乘加总量为 $in\_c \cdot out\_c \cdot k\_size^2 \cdot w \cdot h$,多个计算周期间通过 #pipeline 方式实现 $II = 1$ 的流水线(表 5)。由图 2 结合式(9)、(10)和(11)得到,各层卷积运算理论耗时,如表 2 的最后一列所示。

2.3 测试平台及其硬件资源

测试系统在自行开发的 PCB 上进行,开发环境为 Vivado2017 + SDK,系统开发硬件平台如图 7(a)。本文中所有 PL 端时钟频率均为 142 MHz(单周期长度 7 ns),但经过 Vivado 仿真后统计可看到最终实现仅 6.38 ns。实验时摄像

头传入图像尺寸为 640 ×480,经等比例缩放后实际传入检测网络的图片尺寸为 224 ×224 像素点图像,实验路测效果如图 7(b) 所示。



(a) Zynq SoC系统硬件平台　　(b) 路测效果

图 7　硬件平台及路测实验效果

Fig. 7　Hardware platform and experimental effect

板载有 Zynq-7020 核心以及 AR0132(sensor) + AP0101(外接 ISP) 并口摄像头。Zynq-7020 核心部分,PS 端 256 KB 片上存储; PL 端有 64 bit AXI_ACP 接口,8 通道 DMA,220 片 DSP48E(18 ×25 MACC) ,140 片(可拆分为 280 片) 共 4.9Mb BRAM 存储单元,85K 可编程逻辑单元以及 53 200 单元的 LUT。

在 SDK + Vivado2017 的开发环境下仿真综合后得到 PL 端资源消耗情况如表 3 所示,可以看出,系统只用到了 50% 的总 DSP 资源以 52% 的总 BRAM 资源。

表 3　PL 端资源消耗统计

Tab. 3　Statistics of resource consumption on PL end

硬件资源名	资源总数	实际消耗数	占比/%
BRAM_18K	280	148	52
DSP48E	220	110	50
FF	106 400	18 390	17
LUT	53 200	25 092	47

2.4　实验结果与耗时统计

本节统计并对比三种情形下的单幅图片处理耗时(检测模型均为 X-TINY YOLO) : 硬件资源无限制时的理想耗时、硬件资源有限时的预计耗时、最终实验测得的真实耗时。

理想条件下假设硬件资源充足,算法中将不存在数据重复传输问题且乘加运算全并行,此时耗时主要分为两部分: PS 之间 PL 数据传输(所有传输均可一次完成) 和乘加计算(全并行) ,可得到原模型理想条件下计算量及耗时统计如表 4。由表 4 知,硬件资源无限制时算法总耗时为 26.558 ms。

表 4　假设存储和计算资源充足条件下的模型传输量、计算量

Tab. 4　Algorithm throughput, computation and time-consuming statistics with sufficient hardware resources

操作名称	总运算次数/10^6	运算时间/ms
权值传输	3.954	7.219
feature map 传输	0.533	3.896
乘加运算	304.217	15.443
总计	—	26.558

实际算法实现时由于硬件资源限制,存在矩阵分割引起的重复传输、乘加运算无法全并行的问题,按照第 2 章的解决方案进行数据的分割传输和流水线(流水线内全并行) 计算,在 SDK 开发环境下完成硬件加速函数的编译和综合后形成的 Performance Estimates 表格主要部分如表 5 所示,其展示了流水线性能,可用于预估实际算法耗时。其中,每一个 Loop 消耗的时钟数计算公式如式(12) :

$$clock\_num = II \cdot Trip\_Count + Iteration\_Latency - 1 \tag{12}$$

由表 5 可看出算法主要耗时集中在 1、4 和 6 三个 Loop 中。其中: Loop1 为权值数据从 DDR3 到 BRAM 的传输; Loop4 为 feature map 从 DDR3 到 BRAM 的传输; Loop6 为计算和结果传回,由于计算和传出在同一个 Loop,故传出数据不耗时,故 Loop6 主要是 DSP48E 的乘加运算。Achieved *II* 栏表示实际实现的流水线启动间隔,*Iteration_Latency* 表示单级流水线延时,而 *Trip_Count* 表示每个 Loop 的流水线级数。由表 5 知,Achieved II 均实现最小值 1,而 *Iteration_Latency* 相比 *Trip_Count* 小很多,可以忽略,因此消耗的时钟周期约等于 *Trip_Count* 值,其大小随卷积层数变化而改变,故未能在表格中详细列出。表 6 为预计耗时按功能细分的时间统计。最终得出在有硬件资源限制的小型 Zynq SoC 平台(XC7Z020 核心芯片) 中预计可实现耗时为 33.68 ms。

表 5　PL 端硬件加速函数性能评估

Tab. 5　Performance evaluation of hardware acceleration function on PL side

Loop Name	*Iteration_Latency*	Achieved *II*	*Trip_Count*	总评估时间/ms
Loop1	3	1	changing	33.68
Loop2	3	1	changing	
Loop3	1	1	changing	
Loop4	25	1	changing	
Loop5	3	1	changing	
Loop6	94	1	changing	

表 6　硬件资源有限时算法预计耗时

Tab. 6　Time-consuming of algorithm with limited hardware resources

模块名称	耗时/ms
权值矩阵传输	8.27
feature map	4.86
卷积计算	20.55
总计	33.68

实验时,在 PS 端使用 gettimeofday( ) 函数获得微秒级精度的时间统计。方式是在调用硬件加速函数之前和之后分别计时,两次时间差为单次函数耗时,而单幅图片传入后完成的多次函数耗时总和为网络实际耗时。实验时对 50 张车辆前方路况照片进行循环检测,多次检测后取耗时平均值作为实验结果。实验表明单幅图片网络检测的平均耗时为 40.8 ms,与理论耗时相差 7.12 ms,可能是由于 PL 端时钟不稳定造成。

综上所述,X-TINY YOLO 车辆检测算法在硬件资源无限制的理想 Zynq SoC 平台架构下最短可达到 26.558 ms 的单帧检测时间; 在硬件资源有限制的 XC7Z020 核心最短预计可达 33.68 ms的单帧检测时间; 最终实验测得单帧图片处理的真实耗时为 40.8 ms,即 24.5 帧/s 的图片处理效率,能够满足实时检测需求,但算法理论耗时与实验真实测得的耗时仍存在 7.12 ms的差距。

3 结语

本文首先对 TINY YOLO 目标检测网络的各卷积层进行裁剪和重训练得出适合在小型 Zynq SoC 系统上运行的 X-TINY YOLO 网络，然后通过硬核并行和流水线调度实现算法加速，最终在以 Zynq-7020 为核心的系统板上实现了 24.5 帧/s的车辆检测算法。优化后网络相比原网络的 mAP 指标下降了 5 个百分点，但修改后网络理论计算量仅为原网络理论计算量的 1/2。文中基于 Zynq-7020 的硬件资源限制，提出了对数据进行分割传入的方式，用以解决小型 Zynq SoC 系统对检测网络进行硬件加速时存在的 PS-PL 数据传输瓶颈问题；然后以提高并行效率为目标，提出在 PL 端乘数与被乘数成对存入同一 BRAM 单元的存储办法，解决了 BRAM 数量和端口稀缺带来流水线结构和数据冒险问题，从而提高了流水线调度效率（$II = 1$）。最后使用 DSP48E 硬核在单级流水线中实现了计算的全并行，但实验发现单帧理论耗时与实际耗时存在着 7.12 ms 偏差，后续研究可通过 SDK 提供的 trace 分析方法对硬件加速函数的运行时序进行解析，从而进一步提升系统性能。

参考文献（References）

[1] 傅沈文. 复杂环境下基于图和条件随机域的运动车辆检测[J]. 计算机应用，2012，32(6)：1581－1584.（FU S W. Moving vehicle detection in complex environments based on graph and conditional random field [J]. Journal of Computer Applications, 2012, 32(6): 1581－1584.）

[2] 陈艳，严腾，宋俊芳，等. 基于高斯混合模型和 AdaBoost 的夜间车辆检测[J]. 计算机应用，2018，38(1)：260－263.（CHEN Y, YAN T, SONG J F, et al. Night-time vehicle detection based on Gaussian mixture and AdaBoost [J]. Journal of Computer Applications, 2018, 38(1): 260－263.）

[3] 王德宇，徐友春，李永乐，等. 基于深度学习的车辆检测方法[J]. 计算机与现代化，2017(8)：56－60.（WANG D Y, XU Y C, LI Y L, et al. Vehicle detection based on deep learning [J]. Computer and Modernization, 2017(8): 56－60.）

[4] SRINIVASA N. Vision-based vehicle detection and tracking method for forward collision warning in automobiles [C]// IV 2002: Proceedings of the 2002 IEEE Intelligent Vehicle Symposium. Piscataway, NJ: IEEE, 2002: 626－631.

[5] SCHAMM T, VON CARLOWITZ C, ZOLLNER J M. On-road vehicle detection during dusk and at night [C]// IV 2010: Proceedings of the 2010 IEEE Intelligent Vehicles Symposium. Piscataway, NJ: IEEE, 2010: 418－423.

[6] CHANG W C, CHO C W. Online boosting for vehicle detection [J]. IEEE Transactions on Systems, Man & Cybernetics Part B, 2010, 40(3): 892－902.

[7] REZAEI M, TERAUCHI M. Vehicle detection based on multi-feature clues and Dempster-Shafer fusion theory [C]// PSIVT 2013: Proceedings of the 2013 Pacific-Rim Symposium on Image and Video Technology. Berlin: Springer, 2013: 60－72.

[8] REN S, HE K, GIRSHICK R, et al. Faster R-CNN: towards real-time object detection with region proposal networks [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, 39(6): 1137－1149.

[9] REDMON J, FARHADI A. YOLO9000: better, faster, stronger [C]// CVPR 2017: Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition. Washington, DC: IEEE Computer Society, 2017: 6517－6525.

[10] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition [J]. ArXiv Preprint, 2015, 2015: 1512.03385.

[11] MARTENS J. Deep learning via Hessian-free optimization [C]// ICML 2010: Proceedings of the 2010 International Conference on International Conference on Machine Learning. Madison, WI: Omnipress, 2010: 735－742.

[12] AN D C, MEIER U, MASCI J, et al. Flexible, high performance convolutional neural networks for image classification [C]// IJCAI 2011: Proceedings of the 2011 International Joint Conference on Artificial Intelligence. Menlo Park, CA: AAAI Press, 2011: 1237－1242.

[13] CHEN Y, SUN N, TEMAM O, et al. DaDianNao: a machine-learning supercomputer [C]// Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway, NJ: IEEE, 2015: 609－622.

[14] YUN S B, KIM Y J, DONG S S, et al. Hardware implementation of neural network with expansible and reconfigurable architecture [C]// ICONIP 2002: Proceedings of the 9th International Conference on Neural Information Processing. Piscataway, NJ: IEEE, 2002: 970－975.

[15] FARABET C, MARTINI B, CORDA B, et al. NeuFlow: a runtime-reconfigurable dataflow processor for vision [C]// CVPR 2011 Workshops: Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition. Washington, DC: IEEE Computer Society, 2011: 109－116.

[16] SANKARADAS M, JAKKULA V, CADAMBI S, et al. A massively parallel coprocessor for convolutional neural networks [C]// ASAP 2009: Proceedings of the 2009/20th IEEE International Conference on Application-Specific Systems, Architectures and Processors. Piscataway, NJ: IEEE, 2009: 53－60.

ZHANG Yunke, born in 1993, M. S. candidate. His research interests include artificial intelligence, high performance computing, deep neural network, pattern recognition.

LIU Dan, born in 1969, Ph. D., associate professor. His research interests include network security, Web data mining, image processing.