第47卷第6期 华南理工大学学报(自然科学版) Vol.47 No.6
2019年6月 Journal of South China University of Technology (Natural Science Edition) June 2019

doi: 10.12141/j.issn.1000-565X.180360

基于Tiny-yolo的网络压缩与硬件加速方法

黄智勇 吴海华 虞智 仲元红

(重庆大学 微电子与通信工程学院,重庆 400044)

摘 要: 针对Tiny-yolo网络模型规模大、占内存多、计算量大、不易在嵌入式端实现的问题,提出了网络压缩、结合硬件加速的方法对其进行优化. 首先,分析网络连接关系,对网络贡献较小的连接进行裁剪实现网络压缩,裁剪后的权值矩阵采用稀疏化存储方式减少内存占用;其次,对权值进行量化,通过改变数据的位数,在保证精度误差范围内进一步减小内存占用量和计算复杂度;最后,根据Tiny-yolo网络结构特点提出了深度并行-流水的FPGA加速优化方案,最终实现了Tiny-yolo网络运算的硬件加速. 通过实验验证,网络裁剪结合量化可以实现36X左右的压缩比率,通过硬件加速优化,相比在最大频率为667 MHz的ARM Cortex-A9上运算实现了7X左右的运算加速.

关键词: 神经网络;Tiny-yolo;压缩;硬件加速;FPGA

中图分类号: TP399 **文章编号:** 1000-565X(2019)06-0051-06

Tiny-yolo[1]作为卷积神经网络(Convolutional Neural Network)的一种,与同样用于目标检测的网络模型R-CNN[2]、SSD[3]等相比,检测速度在GPU的支持下可以达到100+FPS. 将其移植到嵌入式端,可以使Tiny-yolo被应用于更多的场景.

卷积神经网络(CNN)的概念很早就被提出[4],其受到广泛关注是在2012年ImageNet竞赛上,Alex Krizhevsky[5]运用卷积神经网络将分类错误从26%降到了15%. 但网络的复杂性限制神经网络在嵌入式端实现,许多研究工作致力于提高网络检测精度的同时简化网络的复杂度.

Babak Hassibi和David G Stork等[6]提出基于分析损失函数的海森矩阵信息,删除影响较小的连接,达到压缩网络的目的. Yunchao Gong等[7]采用向量量化的方法压缩神经网络,达到减少神经网络模型占用存储空间的目的. Wenlin Chen等[8]提出散列网络的方法,利用桶式散列共享参数的方式简化网络. Song Han等[9]将网络裁剪、参数量化、霍夫曼编码方法结合使用,实现网络深度压缩.

硬件加速方面,利用FPGA(现场可编程门阵列)的并行特性对神经网络加速的研究也在持续,很多学者提出了各自的加速方案[10-13]. Clement Farabet等[14]使用FPGA对神经网络进行加速,最终可以对512×384大小的人脸识别图像达到10帧每秒的速度. Qiu等[15]针对卷积神经网络中数据所占位宽过大的问题,通过减小数据位宽,达到加速的目的. Gan Feng等[16]在使用FPGA并行加速的同时,更加关注功耗的问题.

Tiny-yolo的网络模型占用63.5MB的存储空间,而Yolo的模型更是达到了258MB,不利于在存储空间有限的嵌入式端存储. 同时,神经网络包含了大量的卷积运算,消耗时间多. 针对这两点问题,文中采用网络裁剪的方法对网络进行压缩,达到减小模型体积的目的;通过数据量化操作,对权重值进行量化,进一步减少模型所占内存,以及运行消耗的位宽;最后在嵌入式端,采用硬件加速,提高网络运算速度.

收稿日期: 2018-07-08

基金项目: 国家自然科学基金资助项目(61501069)

Foundation item: Supported by the National Natural Science Foundation of China(61501069)

作者简介: 黄智勇(1978-),男,博士,副教授,主要从事无线传感器网络建模和高效能嵌入式计算研究. E-mail: zyhuang@cqu.edu.cn

1 Tiny-yolo 移植方法

针对 Tiny-yolo 的特点,首先对网络进行压缩,满足嵌入式端移植要求,再利用硬件加速方法达到网络识别实时性要求. 文中从 3 个方面展开工作,如图 1 所示:

1) 裁剪网络权值,减小网络规模;

2) 量化数据类型,减小计算位宽消耗;

3) FPGA 深度并行 - 流水优化,提高网络运算速度.

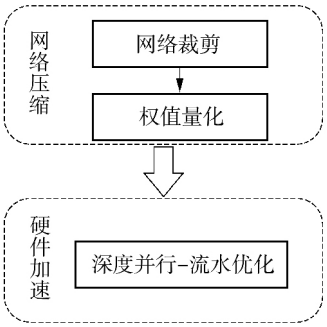


图 1 Tiny-yolo 网络的移植过程

Fig. 1 Migration process of the Tiny-yolo network

1.1 网络裁剪

Tiny-yolo 网络主要由卷积层和最大池化层组成,其权重主要分布在卷积层,文中主要针对卷积层进行裁剪,过程如图 2 所示.

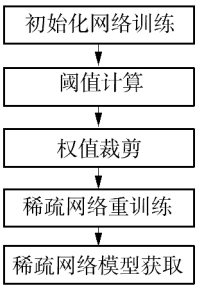


图 2 网络裁剪过程

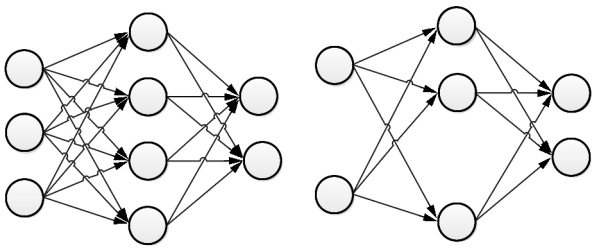
Fig. 2 Network tailoring process

首先训练初始化网络,达到两个目的:(1) 获得原有模型的识别精度,将其与裁剪后模型进行对比;(2) 根据模型的权重分布确定阈值 Ω,此阈值作为裁剪网络连接的标准,如下式:

$$\omega_{ij}^{'}=\begin{cases}\omega_{ij}, & \omega_{ij}\geqslant\Omega\\ 0, & \omega_{ij}<\Omega\end{cases} \tag{1}$$

文献[17]中证实了卷积神经网络训练,可以通过删除部分网络来避免过拟合问题;裁剪阈值数越高,网络压缩率就越高;随着裁剪阈值的进一步增加,识别精度又会逐渐下降;基于此分析,得到结论:裁剪阈值的计算直接由压缩率和识别精度共同决定. 由此确定阈值计算的原则:保持识别精度在误差范围内使得裁剪阈值数最大化.

裁剪后的网络变得稀疏化,采用原模型进行检测会产生较大误差,需要对稀疏网络进行重训练,获取新的网络模型,裁剪前后网络模型如图 3 所示.



(a)裁剪前网络连接 (b)裁剪后网络连接

图 3 裁剪前后网络连接

Fig. 3 Network connections before and after pruning

为保证网络的稀疏性,训练过程只对未被裁剪(权值非 0)的连接的权值进行更新,维持新模型识别精度在误差范围内,获得与稀疏网络相匹配的权重值,如图 4 所示.

$$\begin{bmatrix}\omega_{00} & \omega_{01} & \omega_{02}\\ \omega_{10} & \omega_{11} & \omega_{12}\\ \omega_{20} & \omega_{21} & \omega_{22}\\ \omega_{30} & \omega_{31} & \omega_{32}\end{bmatrix}$$

裁剪 ⇩ 重训练

$$\begin{bmatrix}\omega'_{00} & 0 & 0\\ 0 & 0 & \omega'_{12}\\ \omega'_{20} & 0 & \omega'_{22}\\ 0 & \omega'_{31} & 0\end{bmatrix}$$

图 4 模型权重裁剪前与裁剪重训练后

Fig. 4 Original weightsdata and weights after pruning & retrain

式(2)和式(3)体现了裁剪前后输入特征与输出之间的关系:

$$\begin{cases}Y_0=\omega_{00}\times X_0+\omega_{01}\times X_1+\omega_{02}\times X_2+b_0\\ Y_1=\omega_{10}\times X_0+\omega_{11}\times X_1+\omega_{12}\times X_2+b_1\\ Y_2=\omega_{20}\times X_0+\omega_{21}\times X_1+\omega_{22}\times X_2+b_2\\ Y_3=\omega_{30}\times X_0+\omega_{31}\times X_1+\omega_{00}\times X_2+b_3\end{cases} \tag{2}$$

$$\begin{cases}Y_0=\omega'_{00}\times X_0+0\times X_1+0\times X_2+b_0\\ Y_1=0\times X_0+0\times X_1+\omega'_{12}\times X_2+b_1\\ Y_2=\omega'_{20}\times X_0+0\times X_1+\omega'_{22}\times X_2+b_2\\ Y_3=0\times X_0+\omega'_{31}\times X_1+0\times X_2+b_3\end{cases} \tag{3}$$

式中,X 为输入特征,Y 为输出特征,ω 为裁剪前权

重矩阵,ω'为裁剪重训练后权重矩阵.

权重文件存储采用稀疏化存储方式: 裁剪后权重值为 0 的数目远大于非 0 数目,采用存储权重非 0 值以及这些非 0 值在权重矩阵中位置的方式能有效减少权重文件所占内存空间.

1.2　权值量化

量化的方法被广泛应用于卷积神经网络,其中比较常见的是二值化网络[18],把权值量化为 -1 和 1,从而达到减小位宽的目的,当然也有其它类型的量化,如文献[19]. 文中在保证网络识别精度的前提下,将 Tiny-yolo 原有的 32bits 的 Float 类型量化为 8bits(-128 至 127),可以减少 4 倍左右的位宽消耗. 量化表达式为:

$$Y = \begin{cases} x \times \dfrac{127}{x_{\max}}, & x \geqslant 0 \\ x \times \dfrac{-128}{x_{\min}}, & x < 0 \end{cases} \tag{4}$$

式中,x 为权重值,$x_{\max}$、$x_{\min}$ 分别为最大权重值和最小权重值.

预测输出端,将网络输出值反量化,公式如下:

$$Y' = \begin{cases} x' \times \dfrac{x_{\max}}{127}, & x \geqslant 0 \\ x' \times \dfrac{x_{\min}}{-128}, & x < 0 \end{cases} \tag{5}$$

式中,x'为网络输出,Y'为反量化后的网络输出.

数据的量化与反量化过程如图 5 所示,分别将输入特征与卷积矩阵值进行量化,卷积运算后得到输出值,再进行反量化得到最终结果.

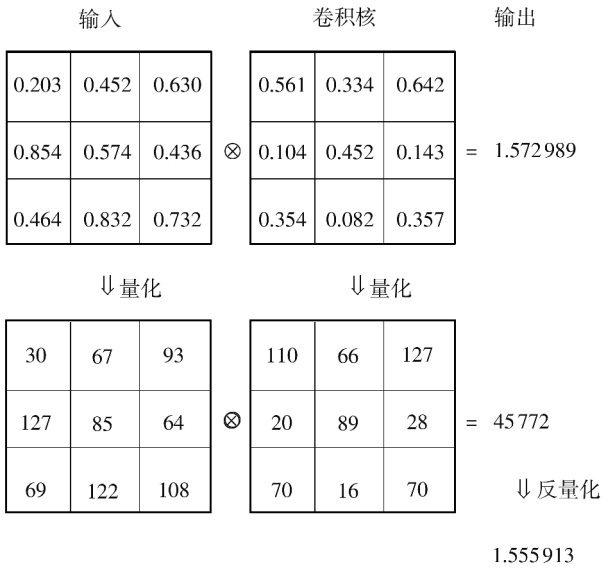


图 5　权重量化与反量化过程

Fig. 5　Weight quantization and inverse quantization process

1.3　硬件加速

Tiny-yolo 的前向识别过程是使用训练好的模型与新的输入图像做卷积运算与池化运算,利用网络的最终输出,对图片中的物体进行识别. CPU 采用串行的方式进行处理,处理速度决定于 CPU 的运算速度,文中利用 FPGA 的并行计算优势,设计一种深度并行 - 流水的 FPGA 加速优化方案.

考虑 Tiny-yolo 的网络结构,其前 12 层网络均为卷积层 + 池化层结构. 卷积、池化操作是输入特征与卷积核进行卷积运算,经过激活函数输出相应特征.

$$x_j = g\left(\sum_{i=1}^{n} x_i \otimes \omega_i + b_j\right) \tag{6}$$

式中,x_j 表示输出特征,x_i 为上一层的特征值,ω_i 为对应权值,b_j 为偏置系数,g 为非线性的激活函数:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

卷积层完成所有输出特征计算后,将数据传递给池化层进行池化操作.

卷积层和池化层之间传递数据,会消耗大量硬件资源,同时增加整个网络的运算时间. 文中将卷积层和池化层合并,一方面可以减少卷积层和池化层之间的数据传输,减少资源消耗; 另一方面池化操作可以和卷积运算同时进行,提升运算速度. 如图 6 所示.

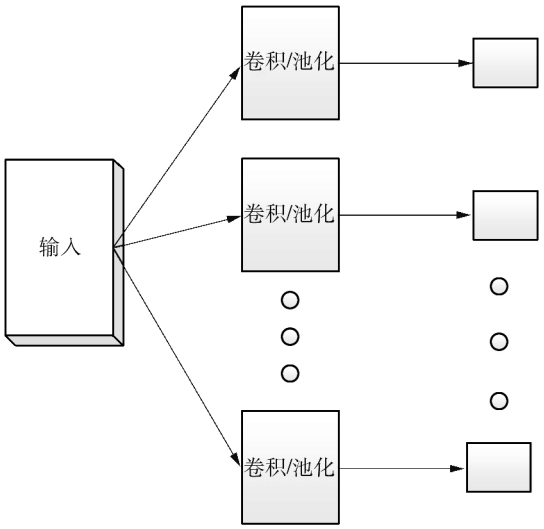


图 6　卷积池化操作

Fig. 6　Convolution pooling operation

Tiny-yolo 的后面 3 个卷积层独立存在,对其设计独立卷积 IP 核进行并行加速,层间数据传输采用流水线结构,提升运行速度.

整个计算过程中,卷积计算是耗时最多的操作. Tiny-yolo 的卷积核的尺寸为 3 × 3(除最后一层),根据卷积核大小进行展开,对卷积操作进行加速. 在单

个周期内就可以完成一个 3×3 大小的卷积操作,理论上可以提升 9 倍的速度,如图 7 所示.

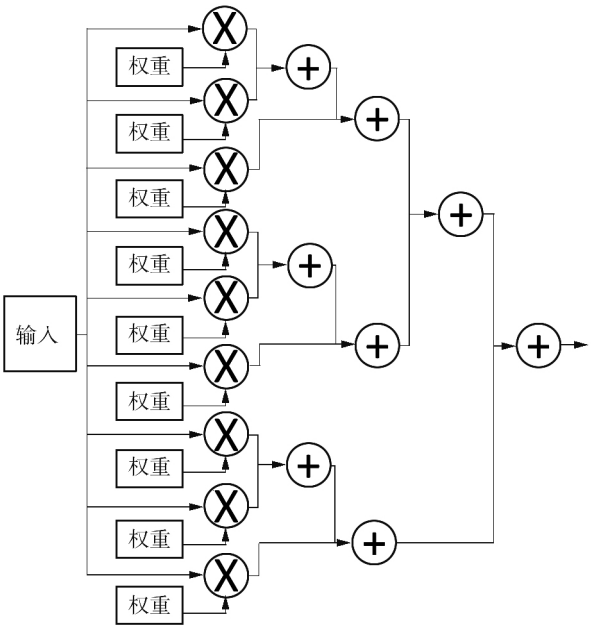


图 7 根据卷积核大小进行展开的卷积操作

Fig. 7 Convolution operations expanded according to the size of the convolution kernel

2 实验结果

2.1 Tiny-yolo 网络裁剪与量化

选择 VOC2007 和 VOC2012 数据集训练 Tiny-yolo 网络模型,记录模型的识别精度. 提取此模型的权重值,分析其特征值分布如图 8 所示.

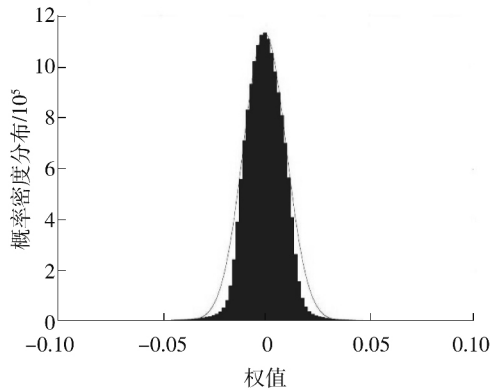


图 8 未裁剪的 Tiny-yolo 网络模型权重分布

Fig. 8 Weight distribution of unclipped Tiny-yolo network model

神经网络权重值大小表示输入对输出特征值的重要程度,裁剪掉权重值较小,即对网络输出影响较小的连接,达到稀疏化网络的目的. 根据 Tiny-yolo 权重值分布特征图,基于识别精度在误差范围内满足裁剪阈值最大化的原则,经过多组实验,选择裁剪阈值为 0.015. 表 1 为实验过程所设阈值、训练集、测试集、模型 mAP(平均精确度).

表 1 阈值确认

Table 1 Threshold validation

阈值	训练集	测试集	平均精确度
0.000	Voc2012 + Voc2007	Voc2007	53.16
0.005	Voc2012 + Voc2007	Voc2007	57.42
0.010	Voc2012 + Voc2007	Voc2007	55.99
0.015	Voc2012 + Voc2007	Voc2007	54.52
0.020	Voc2012 + Voc2007	Voc2007	38.96

所设阈值与识别精度的关系如表 1 所示:实验过程采用步进值 0.005 调整裁剪阈值,裁剪前 mAP 值为 53.16,当裁剪阈值达到 0.02 的时候,mAP 值下降到 38.96,阈值为 0.015 时,mAP 值与裁剪前最接近.

对稀疏网络重训练,获得新的网络模型. 分析其权重分布,如图 9 所示,Tiny-yolo 的权重分布由原来的正态分布变为了双峰式分布,0 值附近的权重值远少于裁剪前的模型,权重值较大的连接明显加强.

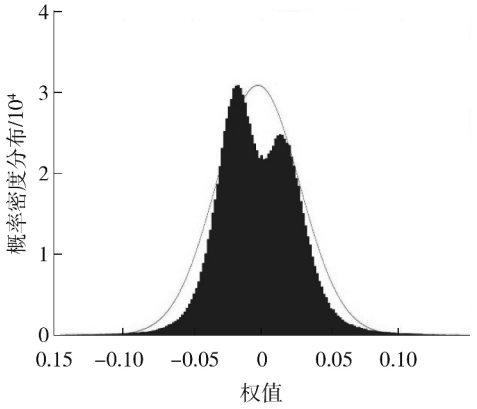


图 9 裁剪之后 Tiny-yolo 的权值分布图

Fig. 9 Weight distribution diagram of Tiny-yolo after pruning

表 2 展示了 Tiny-yolo 网络权值裁剪效果,分析可知,在保持识别精度的前提下,整个网络连接裁剪率达到了 92.54%,其权重值个数(非 0 值)由近 1 600万减少到不足 200 万;权重文件采用稀疏化存储方式,所占内存由原来的 63.5 MB 变为 4.55 MB.

Tiny-yolo 网络输入及权值采用 32 bits 的 Float 类型. 为了进一步减少权重文件所占内存空间,将 32 bits 的 Float 类型量化为 8 bits 的 Char 类型. 每层权值数据映射到 -128 至 127 进行量化,节省了4 倍左右的存储空间,同时也减少了网络运算中数据的传输量. 量化和反量化过程会带来一定误差,最终测试 Map 较之原数据类型约有 2 点下降,但对最终识别结果影响不大.

表 2　Tiny-yolo 每层卷积层权重值裁剪前后个数及裁剪率

Table 2　Number and clipping rate of weight value of each convolution layer before and after pruning

卷积层	权重个数		裁剪率/%
	裁剪前	裁剪后	
1	432	396	8.33
2	4608	3482	24.44
3	18432	12278	33.39
4	73728	41177	44.15
5	294912	123929	57.98
6	1179648	319806	72.89
7	4718592	212027	95.51
8	9437184	386854	95.91
9	128000	82448	35.59
总计	15855536	1907051	92.54

2.2　硬件加速实现

最终的目的是在嵌入式端使用 Tiny-yolo 对输入图像进行处理，为了提高运算速度，根据 Tiny-yolo 的结构特点，在 FPGA 平台上进行硬件加速.

Tiny-yolo 网络整体结构如表 3 所示，输入为 416 ×416 ×3，即 3 个特征值，采用 3 个通道并行方式读取输入特征数据.

表 3　Tiny-yolo 网络结构

Table 3　Tiny-yolo network structure

层名称	输出特征数	卷积核尺寸	输入	输出
卷积层	16	3 ×3	416 ×416 ×3	416 ×416 ×16
池化层		2 ×2	416 ×416 ×16	208 ×208 ×16
卷积层	32	3 ×3	208 ×208 ×16	208 ×208 ×32
池化层		2 ×2	208 ×208 ×32	104 ×104 ×32
卷积层	64	3 ×3	104 ×104 ×32	104 ×104 ×64
池化层		2 ×2	104 ×104 ×64	52 ×52 ×64
卷积层	128	3 ×3	52 ×52 ×64	52 ×52 ×128
池化层		2 ×2	52 ×52 ×128	26 ×26 ×128
卷积层	256	3 ×3	26 ×26 ×128	26 ×26 ×256
池化层		2 ×2	26 ×26 ×256	13 ×13 ×256
卷积层	512	3 ×3	13 ×13 ×256	13 ×13 ×512
池化层		2 ×2	13 ×13 ×512	13 ×13 ×512
卷积层	1024	3 ×3	13 ×13 ×512	13 ×13 ×1024
卷积层	1024	3 ×3	13 ×13 ×1024	13 ×13 ×1024
卷积层	125	1 ×1	13 ×13 ×1024	13 ×13 ×125

前 12 层网络，将卷积层与池化层进行合并运算，运用 FPGA 的并行性特点，卷积运算与池化操作同时进行，减少数据传输，提升运算速度.

从表 3 中可以看到，卷积核大小为 3 ×3，采用 9 倍的展开(UNROLL) ，对其卷积运算进行并行加速，实现一个时钟周期内输出一个特征点，代码描述如下:

```
for( int c = 0; c < channel; c + + ) {
for( int i = 0; i < row; i + + ) {
  for( j = 0; j < col; j + + ) {
  #pragma HLS PIPLELINE II = 1
      for( int k = 0; k < 9; k + + ) {
        #pragma HLS UNROLL
        //进行卷积计算
      }
      //池化计算
      }
    }
    }
```

在 ZedBoard 的 Cortex-A9 上运行 100 次的平均运行时间约为 7.9s. 深度并行 - 流水的 FPGA 加速优化方案，对整个计算过程实现了 7X 的加速，其运行 100 次的平均运行时间约为 1.1s.

3　结语

Tiny-yolo 网络移植存在内存消耗大、计算量大、计算过程时间长的问题. 文中首先对神经网络进行裁剪优化，在保证识别精度在误差范围内，裁剪率达到了 92.54%. 将裁剪后网络进行模型重训练，使用稀疏化存储方式，其权重文件所占空间内存由 63.5 MB 下降为 4.55 MB，并获得了 8X 左右的压缩比率. 通过对权值数据类型进行量化处理，使其模型所占内存进一步压缩 4X. 最后根据 Tiny-yolo 网络特点，基于 FPGA 平台，设计了深度并行 - 流水加速方案，相对于 Cortex-A9 实现了 7X 加速.

参考文献:

[1] REDMON J, DIVVALA S, GIRSHICKR, et al. You only look once: unified, real-time object detection [C] //2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR) . Las Vegas, NV, United States: IEEE, 2016: 779-788.

[2] GIRSHICK R, DONAHUE J, DARRELL T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation [C] //IEEE Conference on Computer Vision and Pattern Recognition(CVPR) . [S. l.]: IEEE Computer Society, 2014.

[3] LIU Wei, ANGUELOV D, ERHAN D, et al. SSD: single shot multi Boxdetector [C] // European Conference on Computer Vision. Amsterdam, The Netherlands: Springer International Publishing, 2016: 21-37.

[4] FUKUSHIMA K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position [J] . Biological Cybernetics, 1980, 36(4) : 193-202.

[5] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Ima-

geNet classification with deep convolutional neural networks [C] // International Conference on Neural Information Processing Systems. [S. l.]: Curran Associates Inc, 2012: 1097-1105.

[6] HASSIBI B, STORK D G. Second order derivatives for network pruning: optimal brain surgeon [J]. Advances in Neural Information Processing Systems, 1992, 5: 164-171.

[7] GONG Yun-chao, LIU Liu, YANG Ming, et al. Compressing deep convolutional networks using vector quantization [J]. Computer Science, 2014, 35(8): 55-58.

[8] CHEN Wen-lin, WILSON J T, TYREE S, et al. Compressing neural networks with the hashing trick [C] // International Conference on International Conference on Machine Learning. [S. l.]: JMLR. org, 2015: 2285-2294.

[9] HAN Song, MAO Hui-zi, DALLY W J. Deep compression: compressing deep neural networks with pruning, trained quantization and huffmancoding [J]. Fiber, 2015, 56(4): 3-7.

[10] CADAMBI S, MAJUMDAR A, BECCHI M, et al. A programmable parallel accelerator for learning and classification [C] // International Conference on Parallel Architectures and Compilation Techniques. [S. l.]: IEEE, 2010: 273-284.

[11] SANKARADAS M, JAKKULA V, CADAMBI S, et al. A massively parallel coprocessor for convolutional neural networks [C] // IEEE International Conference on Application-Specific Systems, Architectures and Processors. [S. l.]: IEEE Computer Society, 2009: 53-60.

[12] FARABET C, POULET C, LECUN Y. An FPGA-based stream processor for embedded real-time vision with convolutional networks [C] // IEEE, International Conference on Computer Vision Workshops. [S. l.]: MIT Press, 2009: 878-885.

[13] PERKO M, FAJFAR I, TUMA T, et al. Low-cost, high-performance CNN simulator implemented in FPGA [C] // IEEE International Workshop on Cellular Neural Networks and Their Applications. [S. l.]: IEEE, 2000: 277-282.

[14] FARABET C, POULET C, HAN J Y, et al. CNP: An FPGA-based processor for convolutional networks [C] // International Conference on Field Programmable Logic and Applications. Prague: IEEE, 2009: 32-37.

[15] QIU Jian-tao, WANG Jie, YAO Song, et al. Going deeper with embedded FPGA platform for convolutional neural network [C] // Acm/sigda International Symposium on Field-Programmable Gate Arrays. Philadelphia: ACM, 2016: 26-35.

[16] FENG Gan, HU Zu-yi, CHEN Song, et al. Energy-efficient and high-throughput FPGA-based accelerator for Convolutional Neural Networks [C] // IEEE International Conference on Solid-State and Integrated Circuit Technology. Hangzhou, China: IEEE, 2017: 624-626.

[17] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: a simple way to prevent neural networks from overfitting [J]. Journal of Machine Learning Research, 2014, 15(1): 1929-1958.

[18] COURBARIAUX M, BENGIO Y, DAVID J P. Binary connect: training deep neural networks with binary weights during propagations [C] // International Conference on Neural Information Processing Systems. [S. l.]: MIT Press, 2015: 3123-3131.

[19] WU J, CONG L, WANG Y, et al. Quantized convolutional neural networks for nobilenevices [C] // IEEE Conference on Computer Vision and Pattern Recognition. LAS VEGAS: IEEE Computer Society, 2016: 4820-4828.

Method of Network Compression and Hardware Acceleration Based on Tiny-yolo

HUANG Zhiyong WU Haihua YU Zhi ZHONG Yuanhong

(School of Microelectronics and Communication Engineering, Chongqing University, Chongqing 400044, China)

Abstract: Existing works based on Tiny-yolo often need large-scale network model, occupy more memories, rely on massive calculation and are not easy to deploy on embedded devices. To solve these problems, an efficient optimization method on network compression and hardware acceleration was proposed. Firstly, connections which have less contribution to the network was pruned after analyzing the network connection relationship and sparse storage was adopted for the clipped weight matrix to reduce the memory consumption. Secondly, memory footprint and computational complexity within the guaranteed accuracy error was further reduced through quantifying the weight data and changing the number of digits. Finally, according to the characteristics of the Tiny-yolo network structure, a deep parallel-stream FPGA acceleration optimization scheme was proposed and the hardware acceleration of the Tiny-yolo network computation was achieved. Experiments demonstrate that the purposed method based on network pruning and quantization can achieve about 36X compression for network model and approximately 7X speedup compared with CPU by hardware acceleration.

Key words: neural network; Tiny-yolo; compression; hardware acceleration; FPGA