Hardware-friendly Deep Learning for Edge Computing

by

Yixing Li

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved December 2020 by the
Graduate Supervisory Committee:

Fengbo Ren, Chair
Sarma Vrudhula
Jae-sun Seo
Baoxin Li

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

The Internet-of-Things (IoT) boosts the vast amount of streaming data. However, even considering the growth of the cloud computing infrastructure, IoT devices will generate two orders of magnitude more than the capacity that centralized data center servers can process or store. This trend inevitability calls for the need for offloading IoT data processing to a decentralized edge computing infrastructure. On the other hand, deep-learning-based applications gain great progress by taking advantage of heavy centralized computing resources for training large models to fit increasingly complicated tasks. Even though large-scale deep learning models perform well in terms of accuracy, their high computational complexity makes it impossible to offload them onto edge devices for real-time inference and timely response.

To enable timely IoT services on edge devices, this dissertation addresses the challenge from two perspectives. On the hardware side, a new field-programmable gate array (FPGA)-based framework for binary neural network and an application-specific integrated circuit (ASIC) accelerator for natural scene text interpretation are proposed, with the awareness of the computing resources and power constraint on edge. On the algorithm side, this work presents both the methodology of building more compact models and finding better computation-accuracy trade-off for existing models.

ACKNOWLEDGMENTS

First and foremost, I am greatly indebted to my advisor, Dr. Fengbo Ren for his consistent guidance, encouragement, inspiration, and support. Dr. Ren motivated me to set a high standard from the beginning of my Ph.D. study, and he also tirelessly shaped me into both an independent researcher and a team player. I have benefited tremendously from his knowledge and experience, and his philosophies about work and life. I would also like to thank my committee members Dr. Sarma Vrudhula, Dr. Jae-sun Seo and Dr. Baoxin Li for their fruitful discussion and guidance over the years.

I would like to thank all PSCLab members: Kai Xu, Michael Riera, Zhikang Zhang, Saman Biookaghazadeh, Erfan B. Tavakoli, Masudul Quraishi and Yuhao Wang for brainstorming together, and all the fun we have had in our lab. Special thanks to my amazing co-authors – Zichuan Liu, Wenye Liu and Shuai Zhang for cross-country collaborations. I would like to thanks the researchers whoever took a time to give me some comments on my works. Those critical feedback and peer recognition have motivated me to become a better researcher in this Ironman Triathlon.

During the half a year I spent in the industry, I had wonderful intern experience with the guidance of my mentor Dr. Ming Kai Hsu at Cadence Design Systems, and Dr. Xin Lu and Ms. Laurie Byrum at Adobe. I was lucky to work on projects that were highly related to my research background, which allowed me to see the excitement of productizing the research ideas.

I would like to thank my friends for their support throughout the intensive job hunting season during the unforeseen circumstances in current pandemic, especially the accompany and emotional support of my boyfriend, Kun Zhang. Finally, I would like to thank my beloved parents for their understanding and support during the entire Ph.D. journey. This dissertation is dedicated to all of them.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

The Internet-of-Things (IoT) boosts the vast amount of streaming data. An estimation of 50 billion IoT devices will be deployed by 2020 (Computing (2016)). In the cloud-based computing scenario, all the data generated by these devices will be stored and processed by the centralized data center servers. However, even considering the growth of the cloud computing infrastructure, IoT devices will generate two orders of magnitude more than the capacity that centralized data center servers can store or process. In addition, due to the latency caused by long server-to-client distance or the network congestion, the recent centralized computing infrastructure is not suitable for any time-sensitive applications. This trend inevitability calls for the need for offloading the IoT data processing to a decentralized edge computing infrastructure (Biookaghazadeh *et al.* (2018)). On the other side, deep-learning-based applications gain great progress by taking advantage of heavy centralized computing resources for training large models to fit increasingly complicated tasks (LeCun *et al.* (2015)). Even though large-scale deep learning-based models perform well in terms of accuracy, their high computational complexity makes it impossible to offload them onto edge devices or fall for real-time inference and timely response.

To enable more IoT services on edge devices, this work contributes from two critical angles. On the edge device side, a new FPGA-based framework for binary neural network and an ASIC accelerator for natural scene text interpretation are proposed, with the awareness of the computing resources and power constraint on edge. On the algorithm side, this work presents both the methodology of building more compact models and finding better computation-accuracy trade-off points for

existing models. The majority of this dissertation has appeared in the following publications.

- Li, Yixing, Zichuan Liu, Kai Xu, Hao Yu, and Fengbo Ren. "A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 14, no. 2 (2018): 1-16.

- Li, Yixing, Zichuan Liu, Wenye Liu, Yu Jiang, Yongliang Wang, Wang Ling Goh, Hao Yu, and Fengbo Ren. "A 34-FPS 698-GOP/s/W binarized deep neural network-based natural scene text interpretation accelerator for mobile edge computing." IEEE Transactions on Industrial Electronics 66, no. 9 (2018): 7407-7416.

- Li, Yixing, Shuai Zhang, Xichuan Zhou, and Fengbo Ren. "Build a compact binary neural network through bit-level sensitivity and data pruning." Neurocomputing (2020).

- Li, Yixing, Akshay Dua, and Fengbo Ren. "Light-Weight RetinaNet for Object Detection on Edge Devices." In 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), pp. 1-6. IEEE, 2020.

- Li, Yixing, and Fengbo Ren. "BNN Pruning: Pruning Binary Neural Network Guided by Weight Flipping Frequency." In 2020 21st International Symposium on Quality Electronic Design (ISQED), pp. 306-311. IEEE, 2020.

This dissertation starts with systematic reviews of the related work of edge device designs and algorithm designs in Chapter 2. The works presented in Chapter 3 and 4 are related to edge computing hardware design. Chapter 3 presents an FPGA-based

inference framework, while Chapter 4 presents ASIC accelerators for binary convolution neural networks. Chapter 5-7 are focusing algorithm designs. Chapter 5 and Chapter 6 illustrate two different solutions for compressing binary neural networks. Chapter 7 demonstrates an application-specific solution of light-weight object detection networks for edge computing. Finally, Chapter 8 summarizes the dissertation, and the future works are discussed.

Chapter 2

RELATED WORK

In this chapter, related works for accelerating neural networks on edge are summarized into two categories: neural network compression algorithms and hardware acceleration on the edge server.

2.1 Neural Network Compression

When referring to hardware-friendly oriented designs, it is not fair to only emphasize compressing the network size. Other than that, the computational complexity is also essential. This chapter first discusses and evaluates the related work for network compression by emphasizing both factors. Then a simple benchmark study is presented to help the readers better understand the computational complexity in terms of hardware resource utilization of the existing work. It can reveal why the binary neural network (BNN) is a more superior solution to be deployed on the edge devices.

2.1.1 Reduce Precision

BinaryConnect (Courbariaux *et al.* (2015)) is a study in the early stage of exploring the binarized weight neural network. In the BinaryConnect network, the weights are binary values while the activations are still non-binary. Arbitrary value multiplies +1/-1 is equivalent to a conditional bitwise NOR operation. Hence, the convolution operations can be decomposed into conditional bitwise NOR operations and accumulation. It is a big step moving from full-precision multiplication to much simpler bitwise operations.

BinaryNet (Hubara *et al.* (2016)) is the first one that builds a network with both

binary weights and activations. The convolution operation has been further simplified as bitwise XNOR (Exclusive-NOR) and bit count operations. The hardware resource cost is minimized for GPU, FPGA and ASIC implementation. For GPU implementation, a 32-bit bitwise XNOR can be implemented in a single clock cycle with one CUDA core. For FPGA and ASIC implementation, there is no need to use DSP (Digital Signal Processor) resources anymore, which is relatively costly. Simple logic elements – LUTs (Look Up Tables) can be used to map bitwise XNOR and bit count operations, which makes it easy to map highly parallel computing engines to achieve high throughput and low latency.

XNOR-Net (Rastegari *et al.* (2016)) also builds the network based on binary weights and activations. However, it introduces a filter of full-precision scaling factors in each convolutional layer to ensure a better accuracy rate. Additional non-binary convolution operations are needed in each convolutional layer, which cost extra processing time and computing resources.

TernaryNet (Zhu *et al.* (2016)) holds ternary (-1, 0, +1) weights for its network. By increasing the precision level of the weights, it enhances the accuracy rate. Since ternary weights have to be encoded in 2 bits, the computational complexity will at least double, compared with BinaryNet.

LQ-Net (Zhang *et al.* (2018a)) studies the bit-width and accuracy tradeoff between different low-precision configurations. The lower bound of weight and activation precision are constrained to 1 bit and 2 bits, respectively. The bit-width of weight/activation in LQ-Net can be configured to 1/2, 2/2, 3/3, 2/32, 3/32 or 32/32. In this paper, LQ-Net only refers to its most compact version – 1-bit weight and 2-bit activation configuration. Its computational complexity will be the closest one to BinaryNet, while the accuracy is improved, especially for the large networks.

2.1.2 Reduce Connection

Network pruning (Han *et al.* (2015)) is revealed as the most popular technique for compressing pre-trained full-precision or reduced-precision CNNs (weights of the reduced-precision CNN are usually in the range of 8 bit - 16 bit (Suda *et al.* (2016)). It compresses the network by pruning out the useless weights, which gains speedup mainly by reducing the network size. Unlike all the other technique mentioned above, neither the weights nor activations of a pruned network are binary or ternary. Still, the computation complexity of the full-precision or reduced-precision multiply-add operation is much higher than that of the BNN. Overall, different kinds of pruning methods can be categorized as magnitude-based and optimization-based pruning.

For magnitude-based pruning (Han *et al.* (2015)), the key idea is to prune out the weights that have small numerical value, which contribute less to compute the output. In the case of BNN, the weights are constrained to +1/-1, so there is no relative small value weights which can't be applied with magnitude analysis. For optimization-based solution (Carreira-Perpinán and Idelbayev (2018)), the pruned network can be resulted in a non-structured or structured way. For non-structured ones, the prunable weights randomly distributes in the 4-D weight space. In a full-precision or reduced-precision CNN, the indexes of non-structured prunable weights can be stored in separated masking arrays. The inference speed can be benefited from skipping the computation of masked weights. However, in the case of BNN, since the weights are already in 1-bit data format, the masking array will introduce quite a lot overhead in memory footprint. Besides, additional logic for skipping the computation of masked weights would ruin the pattern of highly paralleled XNOR computations in BNN. The only possible way which can improve runtime performance is to apply optimization-based structured pruning (Zhang *et al.* (2018b); Luo *et al.*

(2017)) on BNN, but no existing work has done any related study. Usually pruning can be applied with 4-8 bit low-precision networks. However, how much redundancy the BNN still has is still unknown, and no existing solution has been proved to work effectively on such compact BNN.

2.1.3 Other Methods

Singular Value Decomposition (SVD) is one method that has been applied to BNN to compress its weight matrices (Lin *et al.* (2017a)). The basic idea is to decompose a matrix into lower rank matrices without losing much of the important data. SVD is able to provide high compression ratio for high rank matrices. However, for low-rank binary weight matrices of BNN, SVD can only bring 17% memory saving according to Lin *et al.* (2017a).

2.1.4 Comparison

A $W_{(10,10)} \times A_{(10,10)}$ matrix multiplication is implemented on a Xilinx Virtex-7 FPGA board for analyzing the computational complexity of the different architecture that mentioned above. The precision of elements in W and A are the same as the precision of weights and activations in each architecture. The matrix multiplication is fully mapped onto the FPGA. In other words, the proposed design doesn't reuse any hardware resource. So the resource utilization is a good reflection of computational complexity. Since 16 bits are enough to maintain the same accuracy rate as the full precision network (Suda *et al.* (2016)), the precision of any full precision weights or activations are set to be 16 bits. For the pruned network, 84% of the elements in W of the pruned network are set to zeros for a fair comparison. (Since pruned network can get up to 13x reduction (Han *et al.* (2015)) while BNN can get 32x, the size of the pruned network is 32/13=2.5x larger. With 16-bit weights, the total number

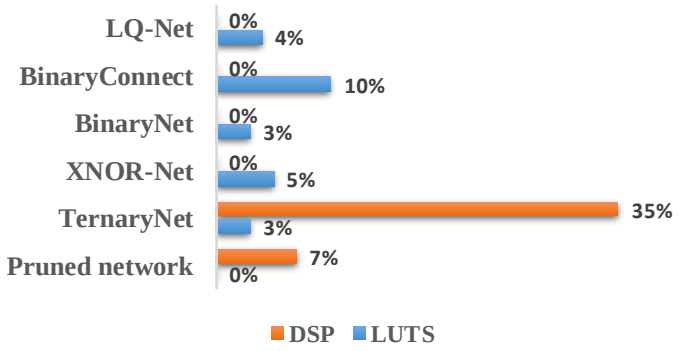


Figure 2.1: Resource Consumption of $W^{(10,10)} \times A^{(10,10)}$ Multiplication on a Xilinx Virtex-7 FPGA for Different Architecture

of non-zero weights of the pruned network is 2.5/16=16% of that of the binarized weight cases.) For LQ-Net (Zhang *et al.* (2018a)), this work only refer to its most compact configuration in this paper, which has 1-bit weights and 2-bit activations. As shown in Fig. 2.1, BinaryNet and LQ-Net apparently consumes the least amount of hardware resource among all these architecture.

In summary, for all the methods mentioned above, pruning can be categorized as connection reduction, while the rest can be categorized as precision reduction. However, both kinds of methods cannot be applied to the BNN. Regarding to the incompatibility of pruning, detailed explanation has been made in Chapter 2.2. For precision reduction, BNN has already reached the lower bound.

2.2 Hardware Acceleration on the Edge Server

The FPGAs, GPUs, ASIC and other special-purpose chips are designed to help resource-constrained, x86-based devices process large volumes of image or audio data through one layer after another of analytic criteria so the app can correctly calculate and weight the value of each.

For ASIC-based edge accelerators, Apple, Qualcomm and Huawei have all an-

nounced that on-device neural processing engine, aiming at partially moving their AI processing module on device. For GPU-based edge accelerators, Nvidia's Jetson series are designed for power-efficient edge computing with 7.5W power budget.

In this chapter, the discussion is only regarding the related work of FPGA- and ASIC-based accelerators, which generally are one order of magnitude more energy efficient then the GPU ones.

2.2.1 FPGA Inference Framework for Deep Learning

Thus far, GPU-based CNN accelerator is still dominant due to its improved throughput over CPUs. However, the high power consumption of GPUs has brought up cooling concerns in data center computing. On the other hand, FPGA-based CNN accelerator has been widely investigated due to its energy efficiency benefits. As the system throughput is proportional to the computing parallelism and operating frequency, the theoretical throughput of GPU-based and FPGA-based CNN accelerators can be estimated on the 1st order based on device specifications. A Titan X GPU has 3,072 CUDA cores, while a Virtex-7 FPGA has 3,600 DSP48 slices. For implementing a full-precision CNN, the computing parallelism of GPUs and FPGAs can be approximately the same. But, GPUs offer 5-10x higher frequency. As a result, FPGAs can hardly match up the throughput of GPUs for accelerating full-precision CNNs. Differently, for a BCNN, the operations in the convolution layers become bitwise XNORs and bit-count logic. A direct impact is that one can use LUTs instead of DSP48 slices to implement the bitwise operations on an FPGA. Hundreds of thousands of LUTs make it possible for a high-end FPGA to match up or surpass the throughput of a GPU, even considering the bitwise operation capability of CUDA cores. Moreover, FPGAs benefit from much higher energy efficiency, which makes it a superior solution for accelerating BCNN in a data center setting. Early research

effort (Hubara *et al.* (2016)) shows that GPU can get 7x speedup using a binary kernel for MNIST classification task on a binary multilayer perceptron (MLP). However, there have been very few studies on exploring FPGA-based accelerator architecture for binary neural networks.

2.2.2 ASIC Accelerator for Deep Learning

The architecture of ASIC accelerator for general NN acceleration is highly depends on the numerical precision. For CNNs with 8-32 bits, the dominant solution is the systolic array (Jouppi *et al.* (2017)). The layout pattern of processing elements (PEs) is a regular 2-D grid, which alleviates the routing issue and results in relatively high frequency range. For CNNs with lower numeral precision (1-3 bits), the most common solution is to design several general PEs for different kinds of layers. In the case of an ASIC accelerator for BNNs, it has three kinds of computing kernels in hardware: floating-point convolution, binary convolution and fully-connected kernels (Zhao *et al.* (2017)). Since it can only maps a single layer of the BNN at a time, only one kind of computing kernels is active at a time. Such a time multiplexing scheme limits the system throughput due to the low hardware utilization.

Beside the ASICs that designed for general CNN acceleration, there are also designs aiming at specific tasks, e.g. pedestrian detection and etc. In general, different tasks are associated with different NN architecture. A dedicated ASIC design can be tailored for a specific NN architecture and its dataflow. Another key point is, the exciting general NN ASIC accelerator may not support its architecture or dataflow control. A dedicated is needed in this case.

Chapter 3

FPGA INFERENCE FRAMEWORK FOR BINARY CONVOLUTION NEURAL NETWORKS

FPGA-based hardware accelerators for convolutional neural networks (CNNs) have obtained great attention due to their higher energy efficiency than GPUs. However, it is challenging for FPGA-based solutions to achieve a higher throughput than GPU counterparts. In this chapter, an optimized FPGA accelerator architecture tailored for bitwise convolution and normalization that features massive spatial parallelism with deep pipeline stages (Li *et al.* (2018)) are proposed. A key advantage of the FPGA accelerator is that its performance is insensitive to data batch size, while the performance of GPU acceleration varies largely depending on the batch size of the data.

3.1 Algorithm Reformulation for Efficient FPGA Mapping

3.1.1 Binary-encoded Convolution

When training the BCNN in (Hubara *et al.* (2016)), the weights and activations are constrained to either +1 or -1. For efficient FPGA mapping, +1/-1s is encoded as 1/0s in the proposed design. In this way, it only takes 1 bit to store a weight or an activation value. Moreover, the convolution operation in layer l is simplified into an XNOR dot product of the input feature map a_{l-1}^b the weight a_l^b, given as

$$y_l(x, y, z) = XnorDotProduct(a_{l-1}^b, a_l^b) \tag{3.1}$$

Equation (3.1) sums up 1s and 0s, which is different from the original BCNN that sums up -1s and +1s. The relation between the original output feature map y_{l0} and

the revised y_l in the proposed design can be expressed as

$$y_{l0} = 1 \times y_l + (-1) \times (cnum_l - y_l) = 2y_l - cnum_l, \tag{3.2}$$

where is the total number of bitwise XNOR operations needed for each y_{l0}. The difference between y_{l0} and y_l compensated in the normalization module in the proposed design.

Note that all the layers take the binary feature map of its previous layer as the input except for the first layer. In the proposed design, the input data is rescaled within the range of [-31,31] and use a 6-bit fixed-point data format, which helps to reduce the resource utilization of non-binary operations at the cost of a limited classification accuracy loss of <0.5%.

Since the input image size is 3×32×32, the computational complexity of the first layer is not a dominating factor. The fixed-point dot product of a 6-bit signed input a_0 and a 2-bit signed weight w_1 is denoted as

$$y_l = FpDotProduct(a_0, w_l) \tag{3.3}$$

3.1.2 Comparator-based Normalization

The parameters subject to training can be considered as constant values in the inference stage. Therefore, the binarization in (4), the normalization function in (2) and the value compensation in (6) are combined into a modified sign function defined as

$$NormBinarize(y_l, c_l) = \begin{cases} 1, y_l \geq c_l \\ 0, y_l < c_l \end{cases} \tag{3.4}$$

where c_l is a constant threshold derived by $c_l = (cnum_l + \mu - \beta\sqrt{\sigma^2 + \epsilon}/\gamma) \times 0.5$, and it is rounded to the nearest integer for hardware implementation. The impact of the

{1. The first layer}
$y_1 \leftarrow FpDotProduct(a_0, w_1)$
$a_1 \leftarrow NormBinarize(y_1, c_1)$
{2. Remaining hidden layers}
for l = 2 to 8 **do**
    $y_l \leftarrow XnorDotProduct(a_{l-1}^b, w_l^b)$
    If (l = 2,4,6) **then**
        $y_l \leftarrow MP(y_l)$
    end if
    $a_l \leftarrow NormBinarize(y_l, c_l)$
end for
{3. Output layers}
$y_l \leftarrow XnorDotProduct(a_{l-1}^b, w_l^b)$
$a_l \leftarrow Norm(y_l, c_l)$

Figure 3.1: Pseudo Code of the BCNN Algorithm.

proposed reformulation on hardware implementation is that both the reformulated normalization and binarization functions can be efficiently implemented as a single LUT-based comparator. In addition, one only needs to store one threshold value c_l for each output value rather than a set of training parameters μ, σ^2 β and γ.

3.1.3 BCNN Model Overview

The inference flow for the reformulated BCNN algorithm is summarized in Fig. 3.1. The convolution in the 1^{st} layer involves fixed-point dot product operations ($FpDot$

$Product$), Differently, bitwise XNOR dot product operations ($XnorDotProduct$) are used in all the other layers. Max-pooling (MP) applied in layers 2, 4 and 6. Normalization and binarization are combined as a single function ($NormBinarize$) which is applied in all layers except for the output layer. The output layer ends with the normalization function $Norm$ for classification.

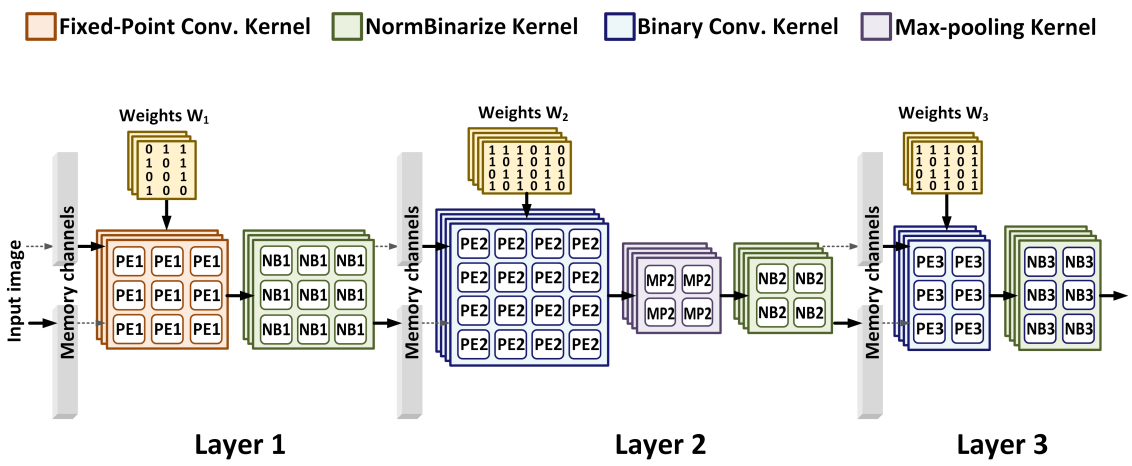


Figure 3.2: Overview of the Proposed Accelerator Architecture for BCNN.

3.2 Architecture Design and Optimization

3.2.1 Architecture Overview

The binary nature of the BCNN enables us to map all the weights, feature maps, and reference values (for normalization) onto the on-chip block RAMs (BRAMs) in a single FPGA. This eliminates any DRAM access latency and dramatically reduces the energy consumption of the system comparing to the existing work relying on off-chip storage (Zhang *et al.* (2015); Farabet *et al.* (2011); Qiu *et al.* (2016); Zhao *et al.* (2017)).

Fig. 3.2 shows the overall architecture of the proposed BCNN accelerator. The binary convolutional kernel in each layer is followed by a NormBinarize (NB) kernel with or without a Max-pooling (MP) kernel. All of the kernels are highly parallelized with an optimized number of processing elements (PEs) and operate in a single instruction multiple data (SIMD) fashion. A streaming architecture is enabled by using double-buffering-based memory channels to handle the data flow between adjacent layers. Each PE in the binary convolutional kernel handles an XNOR dot product operation, which is the core operation in both convolutional and fully-connected layers.

The PEs interface with the BRAMs in parallel to read the weights concurrently.

3.2.2 Architectural Parameters

(I) Loop Unrolling. Note that the three nested loops that accumulate the XNOR output values along the three dimensions of a convolutional filter has loop-carried data dependency. Unrolling data-dependent loops is the same as architectural unfolding, which will improve throughput by increasing the level of temporal parallelism. This trades off more hardware resource with improved computing parallelism. The unfolding factor is a critical architectural parameter in the proposed design, denoted as UF. UF has a maximum value of $WID \times HEI \times DEP$ in each layer.

Differently, the calculation of the pixel values along the three dimensions of an output feature map has no loop-carried data dependency. Unrolling independent loops is equivalent to creating spatial parallelism in the architecture to improve throughput. In the proposed design, the independent loops are fully unrolled to maximize the throughput. The unrolling factor of independent loops is denoted as P. Maximizing P generates a massively parallelized PE array by utilizing the abundant LUT resources on the FPGA.

(II) Pipelining. Loop pipelining is applied in the proposed architecture to further enhance the temporal parallelism and maximize the system throughput. Note that the queuing time to feed in the next data is the inversely proportional to throughput, which is referred to as initial interval I in this chapter. If there is a loop existing in the data path, the minimum initial interval will be limited by the loop latency of the recursive architecture. With loop pipelining, the next data can be feed whenever possible with the minimum initial interval. In the case of a fully pipelined implementation, new data comes in every clock cycle (l=1).

3.2.3 Throughput Modeling and Optimization

If only one XNOR operation and one accumulation are performed in each clock cycle, the total execution time $Cycle_{conv}$ clock cycles of a convolutional layer can be model as

$$Cycle_{conv} = WID \times HEI \times DEP \times FW \times FH \times FD \tag{3.5}$$

where WID, HEI, and DEP denotes the width, height, and depth of a convolutional filter, and FW, FH, and FD denotes the width, height and, depth of an output feature map, respectively. When architectural unfolding is applied in performing the XNOR dot product operation in each PE, will be divided by UF. Similarly, when spatial parallelism is applied to create PE arrays for processing P output pixels in parallel, $Cycle_{conv}$ will be further reduced by P times. The same PE array is reused to calculate the output feature maps with pipelining applied, which contributes to an I-cycle initial interval for the most inner loop. Thus, the throughput of the convolutional kernel with architectural optimization can be formulated as

$$throughput = max(C_1, C_2, C_3, ..., C_k). \tag{3.6}$$

where $freq$ is the system frequency. Note that $throughput_{conv}$ proportional to the estimated cycle count $Cycle_{est}$ convolutional layer, defined as

$$Cycle_{est} = \frac{Cycle_{conv}}{UF \times P} \times I. \tag{3.7}$$

In the proposed accelerator architecture, a double buffering scheme is used to further enhance the spatial parallelism of the system as shown in Fig. 4. The computation of each layer is triggered at the same time and alternates between two phases. Specifically, one channel of $fmap_{L-1}$ is used as the input of the L^{th} layer while the $L-1^{th}$ layer is writing new outputs into the other $fmap_{L-1}$ channel. When both

layers finish processing, the memory buffers swap, and the next processing phase is triggered. Therefore, the overall system-level throughput can be formulated as

$$throughput = \frac{max(C_1, C_2, C_3, ..., C_k)}{freq}, \tag{3.8}$$

where C_L is the execution time of the L^{th} architecture. C_L can be either $Cycle_{est}$ throughput modeling or $Cycle_r$ evaluating real execution throughput. One should note that the system throughput can be maximized with the optimal hardware utilization when all the layers have equal execution time ($C_1 = C_2 = C_3 = ... = C_k$). In the case that the L^{th} layer has longer execution time than other layers, one can always increase the parallelism of the L^{th} layer while decreasing that of other layers to gain throughput with minimum overhead in resource usage. Since the convolutional layers take up over 95% of the computation, only the optimization of convolutional layers is emphasized in this chapter. The fully-connected layer can be easily optimized to match up the system throughput using the same principle.

3.3 FPGA Implementation

In this chapter, it presents the strategy of mapping different computing units to maximize the FPGA resource utilization.

3.3.1 PE Unit

The block diagram of a PE unit is shown in Fig. 3.3. A PE unit handles the XNOR dot product operation of a weight vector and a feature map vector from the previous layer. The vectors are fed into an array of 2-input XNOR gates followed by a parallelized bit-count logic for accumulation. Since both the XNOR gates and the bit-count logic take binary values as input, the PEs can be efficiently implemented using the abundant LUT resources. This is the key to enabling massive computing

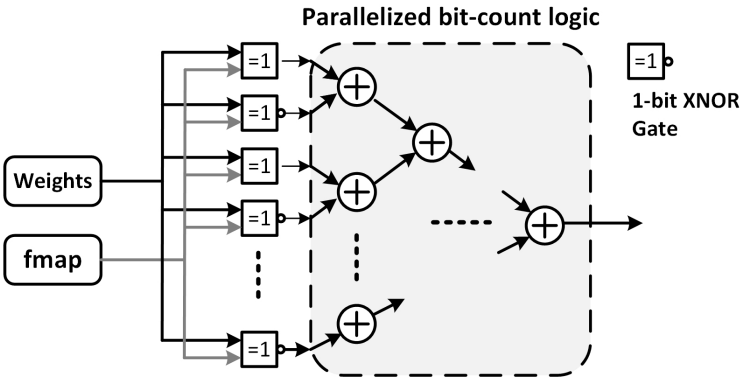


Figure 3.3: Processing Element (PE).

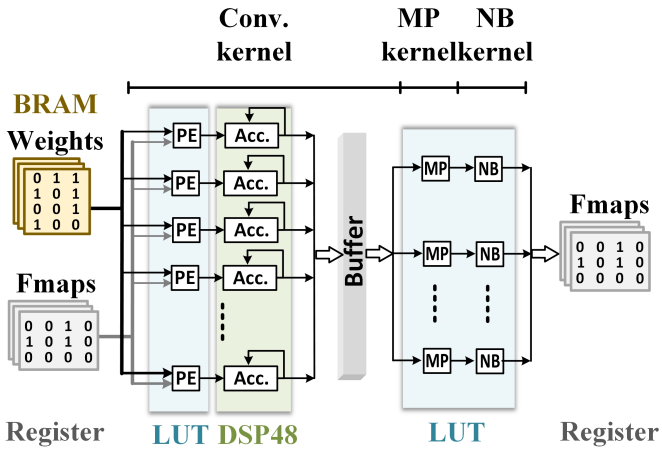


Figure 3.4: The Architecture of Computing Kernels and Their FPGA Mapping Schemes.

parallelism on an FPGA. Note that the number of XNOR gates in each PE is the same as the unfolding factor UF of the current layer. By accumulating the PE output, the pixel value of an output feature map can be computed by the bit-count logic.

3.3.2 Computing Kernels

Fig. 3.4 shows the architecture of the convolutional kernel followed by the Max-pooling and NormBinarize kernels. Each convolutional kernel has an array of PEs implemented using LUTs followed by an array of accumulators implemented using DSP48 slices. The number of PEs and DSP slices is equal to the spatial parallelism

factor P. Each convolutional kernel thereby computes P pixel values of the output feature map in parallel. Besides the weight arrays, only intermediate results of the accumulator outputs (bit-count results) within a single feature map are stored in BRAMs. Feature maps are mapped onto distributed RAMs.

For the convolutional layers 1, 3 and 5 without max-pooling, the outputs of accumulators are directly connected to the NB kernels. The hardware kernel of fully-connected layers is similar to Fig. 3.4. Note that the max-pooling is performed in pipeline with the computation of feature maps in the proposed implementation.

3.3.3 Memory

To read and write a large number of bits in the same clock cycle, one has to partition and reshape the memory arrays in the BCNN model. Partition essentially breaks down a large data array into smaller ones to fit in multiple BRAMs for parallel access. Reshaping basically redefines the depth and width of a single BRAM by grouping multiple words into a wider one. In the proposed design, the weight and $fmap$ arrays are mapped onto BRAMs and distributed RAMs (registers), respectively. Since the maximum word length of a BRAM in a Virtex-7 FPGA is limited to 32 bits, the weight array first was reshaped by 32 and then was partitioned into several BRAMs to guarantee enough memory bandwidth for the required system throughput.

3.4 Experimental Evaluation

The proposed accelerator architecture is implemented for the BCNN in (Hubara *et al.* (2016)) using the optimal architectural parameters shown in Table 3.1. UF and P are optimized for making $Cycle_{est}$ layer approximately the same based on the throughput model in Eq. 3.8. Each layer is also fully pipelined with an initial interval of $I = 1$. Note that the operations along the FW and the FD dimensions are fully

Table 3.1: Optimized Parameters for Each Layer

Layer	UF	P	$Cycle_{conv}$	$Cycle_{est}$	$Cycle_r$
Conv 1	27	32	3538944	4096	5233
Conv 2	384	32	150994944	12288	12386
Conv 3	384	16	75497472	12288	12296
Conv 4	768	16	150994944	12288	13329
Conv 5	768	8	75497472	12288	12386
Conv 6	1536	8	150994944	12288	14473

Table 3.2: FPGA Resource Utilization Summary

Resource	LUTs	BRAMs	Registers	DSP
Used	342126	1007	70769	1096
Available	433200	2060	607200	2800
Utilization/%	78.98	48.88	14.30	39.14

unfolded for maximizing the throughput.

3.4.1 Design Environment

C language is used to describe the accelerator architecture. Vivado HLS is used to produce the RTL codes. The Vivado Design Suite is used to map the design onto a Xilinx Virtex-7 XC7VX690 FPGA. The execution time in terms of clock cycles is reported by Vivado HLS and the system frequency is reported by Vivado Design Suite after the implementation stage.

3.4.2 FPGA Implementation Results

As shown in Table 3.1, the real execution time $Cycle_r$ the synthesis report for each layer is well aligned with $Cycle_{est}$ by the proposed model in Eq. 3.7. The throughput bottleneck is layer 6 in this case. Running at a system frequency of 90 MHz, the FPGA-accelerated BCNN achieves an image processing throughput of 6,218 frames per second (FPS), which is the highest throughput for the same dataset reported by far. The top-1 accuracy rate is 87.8%, which is only 0.3% lower compared to the software model in Theano.

Table 3.3: Experiment Results and Comparison

	Device	Clock (MHz)	Bit-width	GOPS	Power (W)	Energy Efficiency (GOPS/W)	Performance Density (GOPS/kLUT)	Latency (ms)
Farabet	Virtex 6	200	16	147	10	14.7	0.98	-
Zhang	Virtex 7	100	32 float	62	18.7	3.3	0.14	-
Qiu	Zynq-7000	150	16	137	9.6	14.3	0.75	224.6
Suda	Stratix-V	120	8 ~ 16	117.8	25.8	4.56	0.45	262.9
Ma	Arria-10	150	8 ~ 16	645.25	21.2	30	4.01	47.94
Zhang	Intel Xeon	200	32 float	123.48	13.18	9.37	0.62	263.27
Zhang	Arria-10	385	fixed	1790	37.46	47.78	4.19	35.5
Zhao	Zynq-7000	143	1 ~ 2	207.8	4.7	44	4.43	5.94
Andri	YodaNN*	-	1	525	0.06	8600	-	-
Jouppi	Google TPU*	700	8	92000	40	2300	-	-
This work	Virtex 7	90	1	7663	8.2	935	22.4	0.99

To reduce runtime, a bottom-up design strategy was adopted by synthesizing the proposed design layer by layer in Vivado HLS and implementing the entire system in Vivado Design Suite. The overhead introduced by initialization is negligible. Table 4 shows the resource utilization summary for the entire BCNN implementation. LUTs are used for mapping all the computing kernels, including binary convolution, MP and NB kernels. Feature maps of convolutional layers are mapped onto distributed RAMs result in additional LUT consumption. The BRAM usage is mostly consumed by all the weight matrices. Flip-flops are used for storing feature maps and constructing a deep pipeline. Around 30% of the DSP slices are used by the 1^{st} layer to perform fixed-point multiplication. For the rest of convolutional layers, DSP slices are used for accumulating PE outputs as shown in Fig. 3.4.

Existing FPGA-based CNN implementations are compared in Table 3.3. To minimize the impact of different FPGA models on throughput, energy efficiency and performance density defined as throughput normalized to resource utilization are used as the performance metrics for comparison. Compared with the FPGA implementations of floating-point or reduced-precision CNNs, BCNN implementation of this work

achieves 4-124× higher GOPS, 20-283× better energy efficiency and 5-160× better performance density. Even compared with the BCNN implementation in (Zhao *et al.* (2017)), this work achieves 5× better performance density in terms of GOPS/kLUT. The work in (Zhao *et al.* (2017)) implements three kinds of computing kernels in hardware: floating-point convolution, binary convolution and fully-connected kernels. Since this reference work maps a single layer of the BCNN at a time, only one kind of computing kernels is active at a time. Such a time multiplexing scheme limits the system throughput due to the low hardware utilization. In the proposed design, all the layers of the BCNN are mapped into a streaming architecture with optimized architectural parameters, and the data is flowing throughout the entire architecture in a deep pipeline. Therefore, the kernels are constantly active, and the utilization rate of the hardware resources is high. In addition, (Zhao *et al.* (2017)) consumes extra power for loading the weights from off-chip memory layer by layer in addition to the FPGA power reported. On the contrary, there is no such overhead in the proposed architecture since the network is fully mapped and trained parameters are stored on chip.

3.4.3 FPGA-based Versus GPU-based BCNN

Fig. 3.5 compares the performance of the BCNN accelerated by a Titan X GPU and the proposed FPGA-based design. For GPU acceleration, the baseline kernel is designed for floating-point computation, and the XNOR kernel is optimized for bitwise operations (Hubara *et al.* (2016)). In the XNOR kernel, it concatenates 32 1-bit values into a 32-bit value. At the peak performance, each CUDA core can execute 32 bitwise operations per clock cycle. That is the reason why BCNN can also gain remarkable speedup on a GPU when using the XNOR kernel for compilation.

GPU acceleration is apparently sensitive to the size of workload (batch size here).

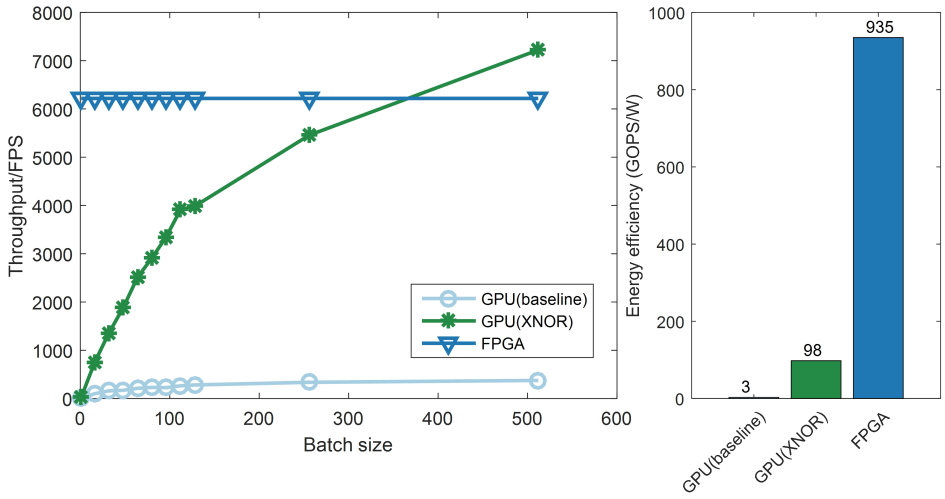


Figure 3.5: Throughput and Energy Efficiency Comparison with GPU Implementations.

One of the keys to achieving high performance in GPU computing is to hide the long latency of functional units by data-level interleaving especially when there are loop-carried data dependency existed in the algorithm. Only when the workload is large enough, a GPU is able to maintain high thread-level parallelism to achieve a high throughput. Differently, the FPGA-based solution is invariant to the batch size of data. Experiment results show that the proposed design significantly outperforms the GPU acceleration using the baseline kernel in terms of both throughput and energy efficiency. Even compared with the GPU acceleration using the XNOR kernel, which is reported as the best GPU-based CNN performance by far, the proposed design achieves a 75× better energy efficiency and an 8.3× better throughput for processing data in a small batch size of 16. For processing data in a large batch size of 512 (the maximum size that fit into the GPU memory), the proposed design can match the throughput of the GPU acceleration with a 9.5× better energy efficiency.

Therefore, the FPGA-based BCNN solution is a clearly better choice for accelerating the data center applications that process online individual requests in small batch sizes. In a recent study conducted by Baidu, a dominant Internet company in

China with 600 million active users, it is reported that the typical on-line prediction workload in terms of batch size is around 8 to 16 (Ouyang *et al.* (2014)). Such small workload is not enough for GPU to achieve its peak throughput performance. Thus, the FPGA-based solution is more superior in handling this kind of requests from individual users.

For processing static data in large batch sizes, the proposed solution is on a par with a Titan X GPU in terms of throughput while delivering much higher energy efficiency. This renders the FPGA-based solution a better choice for energy constrained applications, such as mobile-based advanced driver assistance systems (ADAS). In the ADAS application, a large batch of data needs to be processed for monitoring real-time road condition. In this case, both throughput and energy efficiency are essential and the FPGA-based solution can be deployed.

3.5 Summary

In this chapter, an optimized accelerator architecture tailored for BCNNs is proposed. It is demonstrated for the 1st time that the FPGA-based BCNN solution can greatly outperform a Titan X GPU in terms of both throughput and energy efficiency for processing accurate image classification tasks. The proposed BCNN accelerator running on a Virtex-7 FPGA is 8.3x faster and 75x more energy-efficient than a Titan X GPU for processing individual online requests in small batch sizes. For processing static data in large batch sizes, the proposed solution is on a par with a Titan X GPU in terms of throughput while delivering 9.5x higher energy efficiency. Thus, BCNNs are ideal for efficient hardware implementations on FPGAs regardless of the size of workload. The bitwise operations in BCNNs allow for the efficient hardware mapping of convolution kernels using LUTs, which is the key to enable massive computing parallelism on an FPGA. Applying the optimal levels of architectural un-

folding, parallelism, and pipelining based on the proposed throughput model is the key to maximizing the system throughput. Building memory channels across layers with data-flow control is the key to constructing a streaming architecture to further improve the throughput.

Chapter 4

ASIC ACCELERATORS FOR BINARY CONVOLUTION NEURAL NETWORKS

The scene text interpretation is a critical part of the natural scene interpretation. Currently, most of the existing work is based on high-end graphics processing units (GPUs) implementation, which is commonly used on the server side. However, in Internet of Things (IoT) application scenarios, the communication overhead from the edge device to the server is quite large, which sometimes even dominates the total processing time. Hence, the edge computing oriented design is needed to solve this problem. In this chapter, an architectural design and implementation of a natural scene text interpretation (NSTI) accelerator, which can classify and localize the text region on pixel-level efficiently in real-time on edge devices are proposed.

To target the real-time and low-latency processing, the binary convolutional encoder–decoder network is adopted as the core architecture to enable massive parallelism due to its binary feature. Massively parallelized computations and a highly pipelined data flow control enhance its latency and throughput performance. In addition, all the binarized intermediate results and parameters are stored on chip to eliminate the power consumption and latency overhead of the off-chip communication.

4.1 Preliminary

Conventionally, text recognition has been vastly investigated for document images (LeCun *et al.* (1998)). However, in the natural scene, the background is much more complicated than that of the document images, which makes the scene text recognition become a more challenging task. With the recent development in neural networks

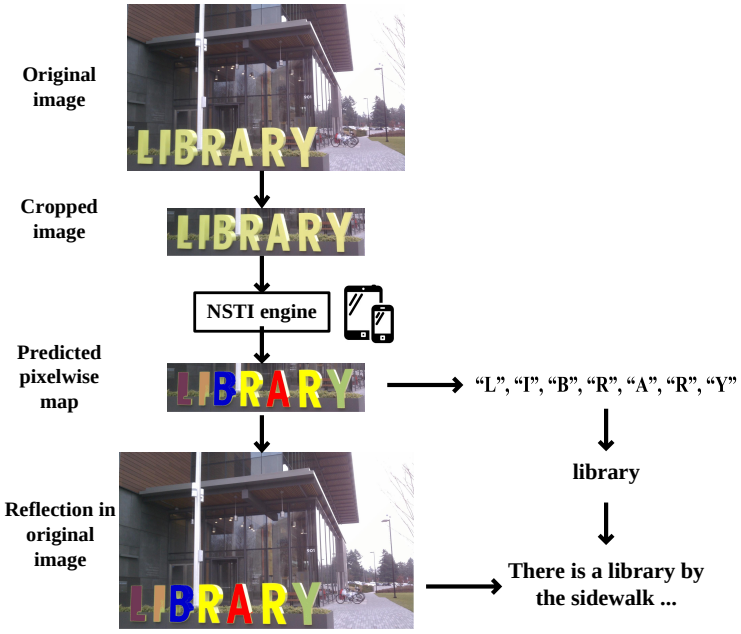


Figure 4.1: Natural Scene Text Interpretation System.

and deep learning (Rahmani *et al.* (2016, 2018)), the accuracy of natural scene text recognition has outperformed the traditional feature selection methods by using features selected automatically The related work can be categorized as character-level based and word-level based solutions. The character-level based solutions (Bissacco *et al.* (2013); Wang *et al.* (2012)) detect and recognize character one at a time. Its front-end is a sliding window approach for character proposals, which makes it suffer from the processing time. The word-level based solution (Jaderberg *et al.* (2014)) requests large fully-connected layer to generate the probability for thousands of word classes, which place a heavy burden on memory access. The shared limitation of either character-level (Bissacco *et al.* (2013); Wang *et al.* (2012)) or word-level based (Jaderberg *et al.* (2014)) solutions is that their architecture is not capable of achieving a low-latency performance. In (Liu *et al.* (2018a)), it performs one-shot text interpretation with a binary convolutional encoder-decoder network (B-CEDNet). Since most of the computation in B-CEDNet are bitwise operations, it opens a new oppor-

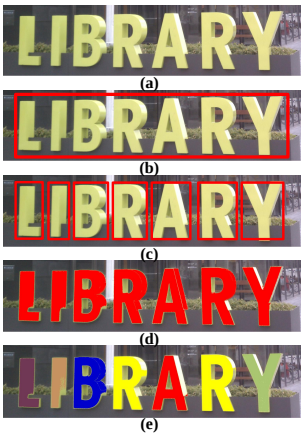


Figure 4.2: Comparison of Different Levels of Natural Scene Text Processing.

tunity for hardware acceleration. However, all the previous work mentioned above is implemented by high-end GPUs (such as Nvidia Titan X). The power-hungry high-end GPUs are not able to be deployed on energy-constrained mobile devices. If GPUs are deployed on the server side, the communication overhead from a client to a cloud server is quite large, which sometimes even dominates the total processing time. However, long latency is not tolerant in augmented reality (AR) applications. If one chooses to use low-power oriented GPUs, such as Nvidia Tegra X1, on the power constrained edge devices, it will get 20× performance (in terms of Flops) degradation compared with the Nvidia Titan X GPU (Inference (2015)). Considering the performance degradation factor, the frame rate in (Liu *et al.* (2018a)) will drop from 200 fps to 20 fps when it is mapped onto a Tegra X1. As such, it cannot maintain a real-time throughput on a lower-power GPU. In addition, the power consumption of a Tegra X1 is 6W (Inference (2015)), which is still too power hungry for a smartphone. Hence, an edge-computing oriented design is needed to solve this problem.

To target a low-latency and real-time processor for energy-efficient natural scene text processing on edge devices, this work propose an ASIC B-CEDNet-based natural scene text interpretation (NSTI) accelerator. As shown in Fig. 4.1, the processor takes the cropped natural scene image as the input and outputs a map of pixelwise

classification results with the same size as input. In comparison with generating a bounding box for each character or the entire word (as shown in Fig. 4.2(b) and (c)), the pixelwise classification output (in Fig. 4.2(a)) shows morphological boundary, which is much more user-friendly in AR applications. Compared with binary classification results for the text and non-text regions in Fig. 4.2(d), the proposed processor can identify different characters in a one-shot prediction. In addition, with the localization, morphological and categorized information, it largely alleviates the workload for the back-end word-level prediction and even scene description as shown in Fig. 4.1. The bitwise operation dominated computation in B-CEDNet enables massive parallelism of multiply-add operations (MACs) in the proposed processor. The binarized parameters and intermediate results are fully mapped on chip to eliminate the communication cost (regarding power consumption) instead of loading them from off-chip memory.

In this chapter, the first session discusses the CEDNet architecture for pixelwise interpretation from the algorithm perspective. Then, the second session introduces its binary counterpart, the B-CEDNet architecture. It emphasizes the differences between two architectures and explains how its binary feature brings new opportunity for the hardware acceleration.

4.1.1 Convolutional Encoder-decoder Network (CEDNet)

Conventionally convolutional neural networks (CNNs) are used for image classification tasks (LeCun *et al.* (2015); Krizhevsky *et al.* (2012); Goodfellow *et al.* (2016)). Generally, they are composed of convolutional layers, pooling layers, and fully-connected layers [16]. To perform image classification, the network only generates one prediction for the entire image. Therefore, CNNs cannot be directly deployed for the pixelwise interpretation of images. In Fig. 4.3, the convolutional

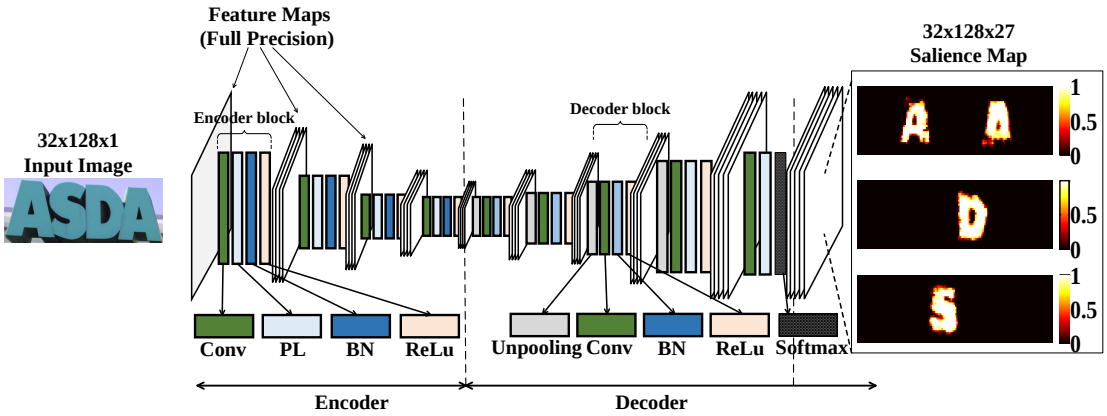


Figure 4.3: Architecture of the Convolutional Encoder-decoder Network (CEDNet).

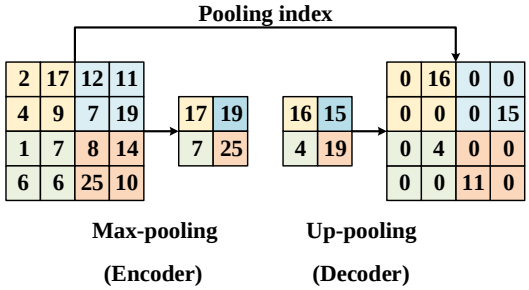


Figure 4.4: Pooling and Up-pooling Layers.

encoder-decoder network (CEDNet) is proposed in (Badrinarayanan *et al.* (2017)) for the multi-class pixelwise classification. A CEDNet takes the scene text images as input. The body of the network can be divided into the encoder part and decoder part. The output of the CEDNet is a salience map $S \in \mathbb{R}^{W_I \times H_I \times C}$, which contains the probability information of each pixel over C categories (including one background class), where C is 27 (characters are case insensitive). The encoder part is a stack of encoder blocks, while the decoder is a stack of decoder blocks. Each encoder block contains a convolutional (Conv) layer, a pooling layer (PL), a batch normalization (BN) layer and a rectified linear unit (ReLU) layer. The convolutional layer applies convolutional operations on input feature map $a_{k-1} \in \mathbb{R}^{W_{k-1} \times H_{k-1} \times D_{k-1}}$ with trainable weight matrix $w_k \in \mathbb{R}^{w_k \times h_k \times D_k \times D_k}$, where the subscript k indicates the k^{th} block.

The convolutional operations can be formulated as

$$s_k(x,y,z) = \sum_{i=1}^{w_k}\sum_{j=1}^{h_k}\sum_{l=1}^{D_{k-1}} w_k(i,j,l,z) \\ *a_{k-1}(i+x-1, j+y-1, l), \tag{4.1}$$

where $s_k \in \mathbb{R}^{W_k \times H_k \times D_k}$ is the output of k^{th} Conv layer. Equation (4.1) shows that the computation of s_k along three dimensions has no data dependence, which can be highly paralleled in an ASIC implementation. The Conv layer is intended to extract high-level features, which are critical for the pixelwise classification. In the PL layer, it pools out the critical information and eliminates the non-critical one. The PL layer can perform either max pooling or average pooling (LeCun *et al.* (2015)). A max-pooling layer is shown in Fig. 4, it pools out the maximum value in each 2×2 window. By introducing the pooling layer, the size of the feature map is shrinking as the network goes deeper. The BN layer is mainly used for accelerating training process (Ioffe and Szegedy (2015)). So in the inference stage, the BN layer is also applied to match the training process forming a stable distribution of the activations (a_k). The output of k^{th} BN layer is represented as follows:

$$a_k(x,y,z) = \frac{s_k(x,y,z) - \mu(x,y,z)}{\sqrt{\sigma^2(x,y,z) + \epsilon}}\gamma(x,y,z) \\ +\beta(x,y,z), \tag{4.2}$$

where μ and σ^2 is the mean and variance over the mini-batch training data, while γ and β are trainable scaling factors. The activation function is a nonlinear transformation. The most commonly used activation function (LeCun *et al.* (2015)), ReLU function is represented as

$$a_k(x,y,z) = \begin{cases} 0, a_k(x,y,z) \leq 0 \\ a_k(x,y,z), a_k(x,y,z) \geq 0. \end{cases} \tag{4.3}$$

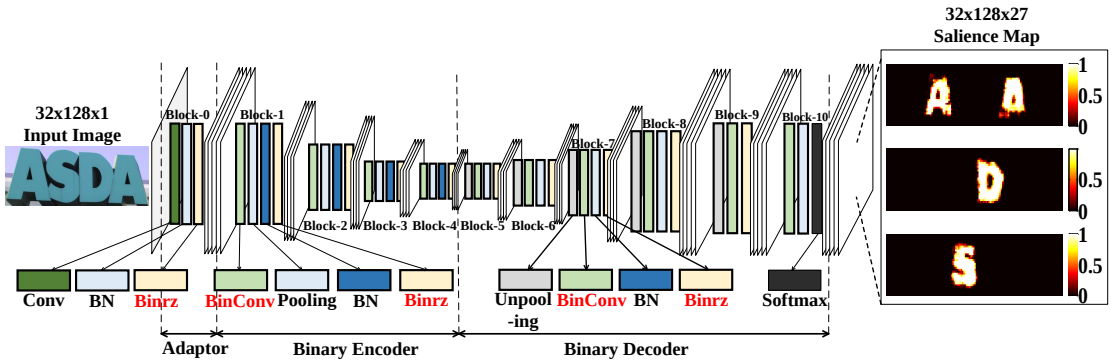


Figure 4.5: Architecture of the Binary Convolutional Encoder-decoder Network (B-CEDNet).

The entire encoder part is similar to a CNN without fully-connected layers.

Since the output salience map is desired to be the same size as the input, in the decoder part, each decoder block substitutes the pooling layer with the up-pooling layer. As shown in Fig. 4.4, the up-pooling (UPL) layer pools back the maximum value to the same index in corresponding max-pooling layer. As such, the output salience map can represent the same localized information as the input. In order to predict the pixelwise character appearance probability, the output block replaces the ReLU function with softmax function. As shown in the rightmost part of Fig. 4.3, it only plots the salient map slices for character "A", "D" and "S". The lighter color code means higher confidence level and vice versa. The CEDNet architecture can enable highly parallelized MAC computing inside every encoder or decoder block. It eliminates both the run-time bottle stage in sliding window-based proposal and the computation-intensive fully-connected layer.

4.1.2 Binary Convolutional Encoder-decoder Network (B-CEDNet)

Even though the mobile devices are getting more and more computing power, it is still hard to deploy full-precision CNNs for efficient computing on mobile edge devices. Since the CNN architecture is proved to have huge redundancy (Cheng *et al.*

(2015)), different methods (Courbariaux *et al.* (2015); Hubara *et al.* (2016); Rastegari *et al.* (2016); Han *et al.* (2015); Jacob *et al.* (2018)) have been proposed to reduce the computation complexity and/or alleviate the memory access issues. Some approaches (Han *et al.* (2015)) focus on minimizing total number of parameters, which mainly alleviate the memory access issues. While other approaches (Courbariaux *et al.* (2015); Hubara *et al.* (2016); Rastegari *et al.* (2016); Jacob *et al.* (2018)) reduce the precision of weights and activations, which can both reduce the computation complexity and alleviate the memory access issues. Among these approaches, binarization (Hubara *et al.* (2016); Rastegari *et al.* (2016); Lin *et al.* (2017d)) can push the weights and activations to be represented in binary format $w_k^b \in \{0,1\}^{W_k \times H_k \times D_k \times D_k}$, and $a_{k-1}^b \in \{0,1\}^{W_{k-1} \times H_{k-1} \times D_{k-1}}$. It can achieve up to 32× memory saving and converting the convolution operations to bitwise XNOR operations for much more efficient computing. It has been proved in (Liu *et al.* (2018a)), binarization approach can be adopted in CEDNet to build a binary convolutional encoder-decoder network (B-CEDNet as shown in Fig. 4.5 for pixelwise text classification with merely no accuracy drop.

In the B-CEDNet, it replaces the Conv layer and ReLU layer with the binary convolutional layer (BinConv layer) and Binarization layer (Binrz layer), respectively. The equation for the BinConv layer and Binrz layer is shown in (4.4) and (4.5), respectively.

$$s_k(x,y,z) = \sum_{i=1}^{w_k} \sum_{j=1}^{h_k} \sum_{l=1}^{D_{k-1}} \sim (w_k^b(i,j,l,z) \oplus a_{k-1}^b(i+x-1, j+y-1, l)) \tag{4.4}$$

$$a_k^b(x,y,z) = \begin{cases} -1, a_k(x,y,z) \leq 0 \\ +1, a_k(x,y,z) \geq 0 \end{cases} \tag{4.5}$$

The most costly computation, full-precision multiplication, is now converted into the hardware-friendly bitwise XNOR operation. For GPU implementation, one MAC module can process 32-bit bitwise XNOR instead of one 32-bit multiply-add operation. For FPGA implementation, the BinConv layer is no longer needed to be implemented in DSP slices. Massive LUTs can be used for efficiently implementing bitwise operations. For ASIC implementation, it is flexible enough to build tailored computing units for a BinConv layer with tree-like bitwise XNOR and bit-count logics. With simplified basic computing units, it is able to map massive computing units to target a high system throughput.

B-CEDNet has brought new opportunity in energy-efficient edge-computing applications. Compared with power-hungry GPU-based solutions and overhead of routing in FPGA-based solutions, a tailored ASIC solution for B-CEDNet can be the most energy efficient solution with high throughput performance. It is able to satisfy the need for real-time and low-latency processing in power-constrained edge-computing device for scene text interpretation.

4.2 Architecture Design

Most existing ASIC/FPGA-based CNN accelerators are only compatible with encoder blocks (down-sampling trend) for image classification, recognition and detection tasks (Moons and Verhelst (2016); Tu *et al.* (2017)). While some optimized designs for decoder blocks (up-sampling trend) for super resolution applications (Zhang *et al.* (2017)). The proposed architecture is customized for the convolutional encoder-decoder network. The Fig. 4.6 shows the ASIC architecture of the proposed Natural

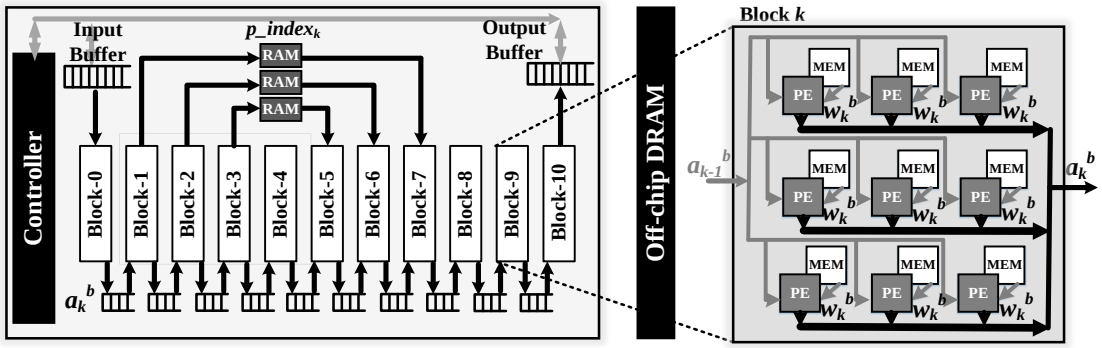
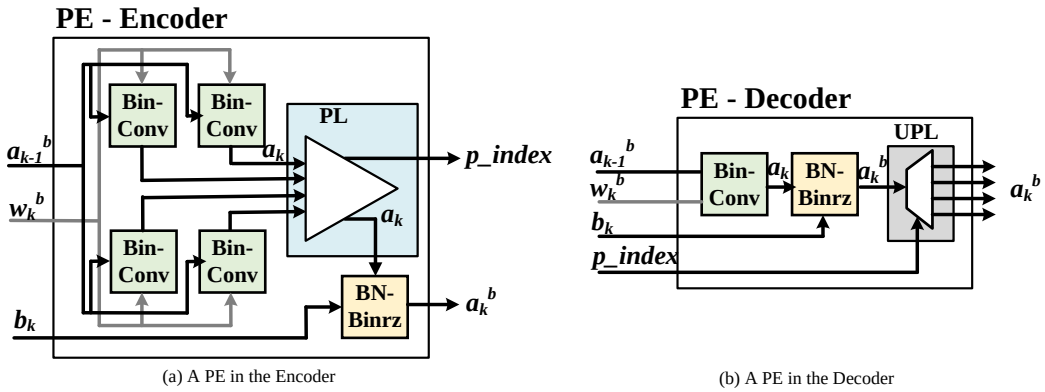


Figure 4.6: Architecture of the Binary Convolutional Encoder-decoder Network (B-CEDNet).

Scene Text Interpretation (NSTI) accelerator. The NSTI accelerator takes the scene text image from the off-chip DRAM as the input. Then it is processed through computing blocks in a streaming manner. The computing blocks, Block-0 to Block-10, are corresponding to 11 blocks in Fig. 4.5. Each computing block is built upon a processing element (PE) array, as shown in the right half of the Fig. 4.6. Each PE performs the operations of convolution, max-pooling/un-pooling, activation function and batch normalization. The spatial parallelism of the NSTI accelerator is reflected on the block level, PE level and sub-PE level. The temporal parallelism is reflected in highly pipelined steaming data flow. Both massive spatial parallelism and temporal parallelism enable high throughput performance of the proposed NSTI accelerator. Reduction in computation complexity to bit-level operations benefits in power saving. Storing all the weights (w^b) and intermediate results (a^b) on chip to minimize off-chip communication gives extra credits to energy saving.

4.2.1 Processing Elements

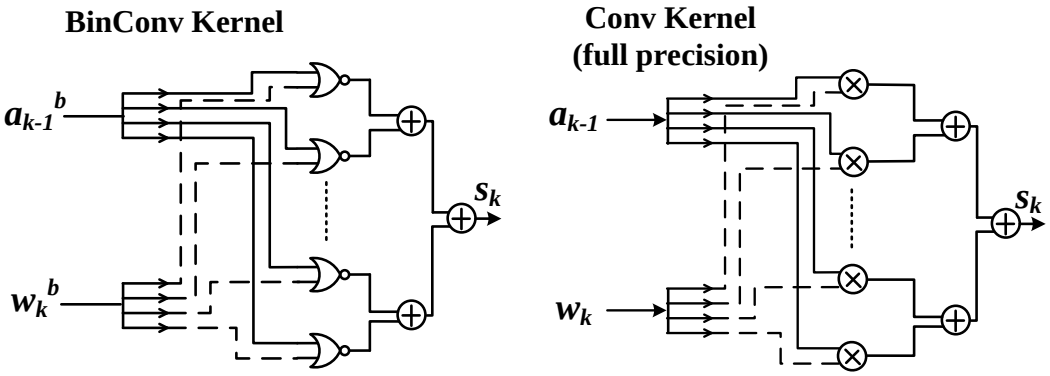
Each computing block in Fig. 4.6 performs the computation corresponding to Fig. 4.5. Therefore, Block-1 to Block-4 and Block-5 to Block-8 are identical, respectively. Although the functions vary among these blocks, the structure inside each block is the



(a) A PE in the Encoder

(b) A PE in the Decoder

Figure 4.7: Processing Elements (PEs).



(a) BinConv kernel in the encoder and the decoder

(b) Conv kernel in the adaptor

Figure 4.8: BinConv Kernel and Conv Kernel.

same as shown in Fig. 4.6. In each block, PE arrays take the feature map a_{k-1}^b from previous layers and weight w_k^b values from its local memory (ROM) as the inputs, and output the feature map a_k^b of the current layer. All the PEs in the same block work simultaneously. The differences among these blocks exist in their processing elements (PEs).

The PEs of encoder and decoder are shown in Fig. 4.7. The PE of the encoder in Fig. 4.7(a) has 4 BinConv kernels, a PL kernel and a BN-Binrz kernel, while the PE of the decoder in Fig. 4.7(b) has a BinConv kernel, an UPL (un-pooling) kernel and a BN-Binrz kernel. For the convenience of the ASIC implementation, the UPL layer in block $k+1$ is grouped with block k for building the computing block. Therefore,

in each decoder PE, it starts with a BinConv kernel and ends with an UPL kernel. If BinConv kernels in an encoder PE are substituted with Conv kernels, it becomes a PE for the adapter. BinConv kernels in Fig. 4.7(a) and Conv kernels of the adapter are both implemented in a tree-like structure as shown in Fig. 4.8. A Conv kernel has a floating-point operation on each node, while a BinConv kernel performs bit-level XNOR and bit-count. They both are implemented by pure combinational logics. In each Conv/BinConv kernel, it computes one $s_k(x, y, z)$ at a time, that is to say, the parallelism factor in terms of number of operations is $w_k \times h_k \times D_k$. The computation of the BN and Binrz layer can be simplified as a threshold function (Li *et al.* (2017)), which can be implemented by a single 2-input comparator, denoted as BN-Binrz kernel in Fig. 4.7. The PL kernel is implemented with a 4-input comparator, which also encodes the index of the maximum value in pooling region. The pooled out value and its index are stored in buffer. Then feed them into the UPL kernel in its symmetric decoder block as shown in Fig. 4.6. The DEMUX in the UPL kernel of Fig. 4.7(b) writes back the (pooled maximum) value with the index information to the right location in the RAM. The up-pooled data in the buffer serves as the input of next decoder block.

4.2.2 Memory Design

In DL-based ASIC designs (Chen *et al.* (2017); Desoli *et al.* (2017); Bong *et al.* (2017)), the communication to the off-chip DRAM is very power-intensive. The binary feature of B-CEDNet enable us to store all the weights (w^b) and intermediate results (a^b) on chip to minimize off-chip communication for energy saving. As shown in 4.1, the first and second column indicates the memory size of weight values in the non-binary case (CEDNet) and binary case (B-CEDNet), respectively. The total memory size of weights in the B-CEDNet has 30× saving, comparing with the non-binary one.

The ideal memory saving results from converting full-precision network (32 bits) to binarized-weight network should be 32×. Since the first layer still has non-binary weights, the real compression ratio is a little bit less than the ideal case. 2,144 KB distributed ROMs are built to store all weight values, as shown in Fig. 4.6. In a PE array, each PE has its local ROM attached. This can alleviate the routing issue in the bottom-up design flow. There are total 423 KB binarized intermediate results a^b between blocks. Since the size of a^b is relatively small, synthesized shift registers are used to buffer a^b. This can enable global voltage scaling with the core computing part to get a more energy efficient point in chip testing stage. 172 KB Block SRAMs (hard macros) are used between the encoder block (Block-1, -2 and -3) and its symmetric decoder block (Block-5, -6 and -7) to buffer the pooling index. For the innermost block, Block-4, the outputs of its max-pooling layer is directly up-pooled (up-sampled). As such, there is no need to store the pooling index of block 4.

4.3 Dataflow Control

In Fig. 4.9, it shows the data flow control across different blocks. Since all the layers share the same depth, B-CEDNet is simplified to a 2-D version in the following discussion. In a BinConv layer, the filter size of the weight matrix is 3×3 and the stride is equal to 1. While in a PL layer, the filter size is 2×2 and the stride is equal to 2. In an encoder block, since a BinConv kernel and its corresponding PL kernel are grouped into one PE (as shown in Fig. 4.7(a)), the size of sliding window should be 4×4. In the decoder case, the size of the sliding window is 3×3 with only one BinConv PE.

The color code is in Fig. 4.9 indicates the location of the sliding window regarding to the clock cycle. Each sliding window is located by the pixel of its upper-left corner.

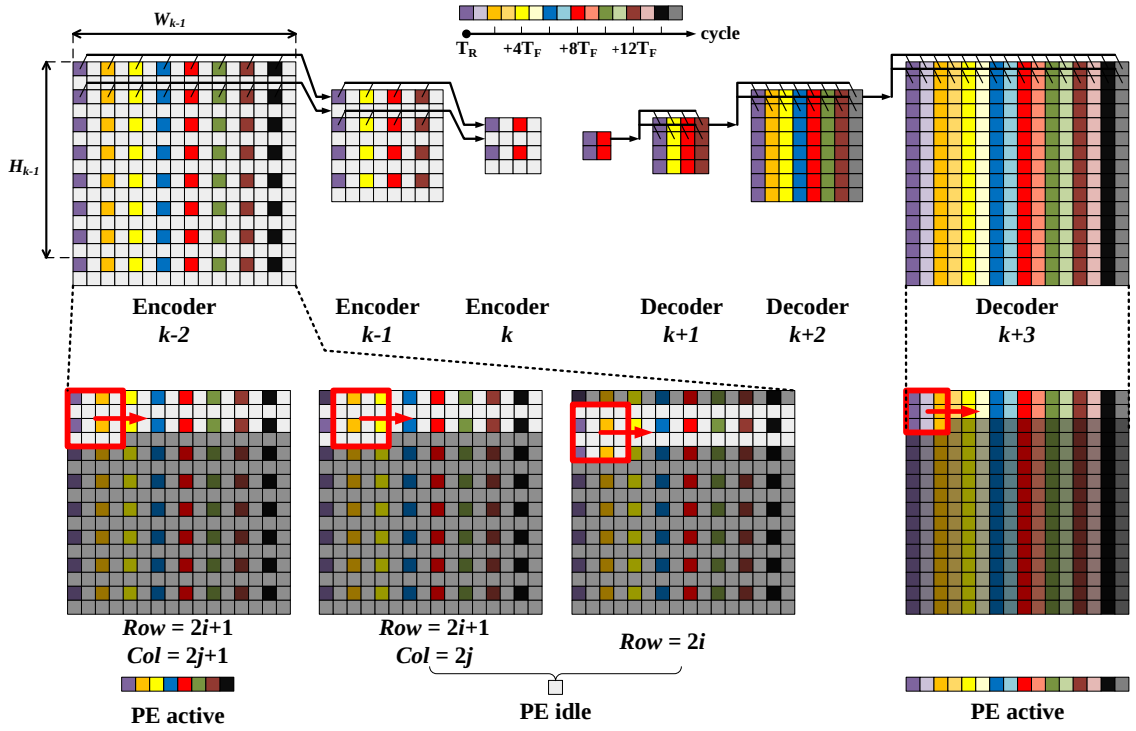


Figure 4.9: Data Flow Control Between Blocks.

In the first pixel of each row, the reference time is defined as T_R, where R is the row index. The same feature map region (where the red sliding window is) is reused at time T_F and process it with different weight values. In order to maximize the data reuse, $(F-1) \times W_{k-1} + F$ pixels are buffered at a time, where the size of the sliding window is $F \times F$. The buffered data in Fig. 4.9 are in relatively high brightness. Feature map reuse helps to reduce the frequency of fetching new (feature map) data, which will result in energy saving. After T_F cycles, the window slides to the right with the stride equal to 1. All the pixels with non-white color codes indicate the PEs are active. In the active mode, the PEs read in new data from the previous block, execute the computation and write the processed data to the next block. Since PEs are implemented by combinational logics, once the buffered data is ready, the current block produces valid results simultaneously. All the other pixels in the white color code indicate the PEs are idle, where the PEs only reads new data into the buffer.

Table 4.1: Memory Summary (Unit:KB)

Block index	0	1	2	3	4	5	6	7	8	9	10	Total
w	22	2008	9008	9008	9008	9008	9008	9008	9008	9008	54	66126
w^b	22	71	289	289	289	289	289	289	289	289	9	2144
a^b	<<1	50	16	8	4	1	8	16	16	16	N/A	423
p_index	N/A	131	33	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	172

Table 4.2: Chip Summary

Symbol	Quantity
Technology	40nm 1p10m CMOS
Transistor flavor	HVT 92.8%, SVT 7.2%
Gate count	2811 kGates
I/Os	Digital: 13/27, Power: 33
Core VDD	0.9 V
I/O VDD	1.8 V
Core size	12.7 mm^2

As shown in Fig. 4.9, the PEs are active in 1/2, 1/4 and 1/8 of total time in the encoder block $k-2$, $k-1$ and k, respectively. In order to maximize the utilization (active time ratio) of PEs, it is designed to have $4\times$, $2\times$ and $1\times$ number of PEs, accordingly. Similarly, in the decoder block $k+1$, $k+2$ and $k+3$, number of PEs increases as $1\times$, $2\times$ and $4\times$. Therefore, the proposed data flow control makes all the computing blocks work in a highly pipelined fashion, which enhances the throughput performance of the NSTI accelerator.

4.4 Experimental Evaluation

The configuration of the B-CEDNet is the same as (Liu *et al.* (2018a)). The chip summary is shown in Table 4.2. The NSTI accelerator is implemented in a 40nm 1p10m process using a standard-cell-based design flow. The RTL code is synthesized in Synopsys Design Compiler (DC). To achieve the target throughput, a clock period of 33.33 ns (30 MHz) evaluated at the worst-case process, voltage, and temperature (PVT) corner is targeted throughout the chip implementation. Taking into account

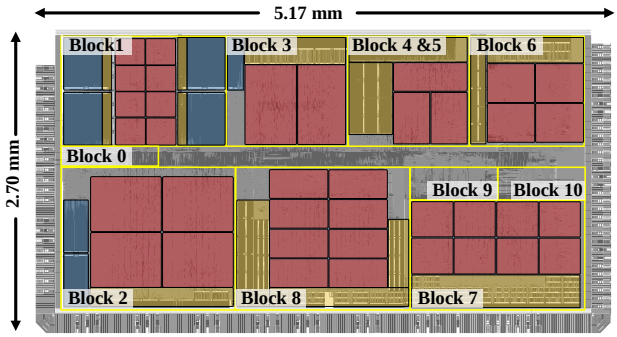


Figure 4.10: The Layout of NSTI Accelerator.

the overhead to be introduced by the subsequent physical design, a 40% timing slack is used during the synthesis. Specifically, the NSTI accelerator is synthesized with a target clock frequency of 30/(1-40%) = 50 MHz. To reduce leakage power, the NSTI accelerator is first synthesized using high-threshold (HVT) standard cells only. Then, standard-threshold (SVT) standard cells are selectively inserted into the critical paths for timing improvement. This is carried out by switching on the leakage optimization tool in DC. Overall, the chip occupies a core area of 12.7 mm^2 with an aspect ratio of 0.52 and integrates 2811 kGates. The layout of the accelerator is shown in Fig. 4.10. The computing blocks and the buffers for intermediate results are colored in red. The pooling indexes (shown as RAM in Fig. 4.6) have consumed a memory size of 172 KB. To reduce area cost, RAMs are realized by dual-port SRAM hard macros, which are in blue. The 2,144 KB local ROMs for weights are distributed in each block, colored in yellow. For the leakage reduction purpose, HVT devices are used in 92.8% of the logic cells. The chip has 13 digital inputs, 27 digital outputs, and 33 power pads supply core and I/O power domain. The I/O domain has a constant supply voltage of 1.8 V, and the logic and memory domain both have a normal supply voltage of 0.9 V. The post-layout simulation is performed to verify the functionality.

As shown in Fig. 4.11, the first row is the input images of the NSTI accelerator. The accelerator takes the cropped text region and outputs the prediction of each

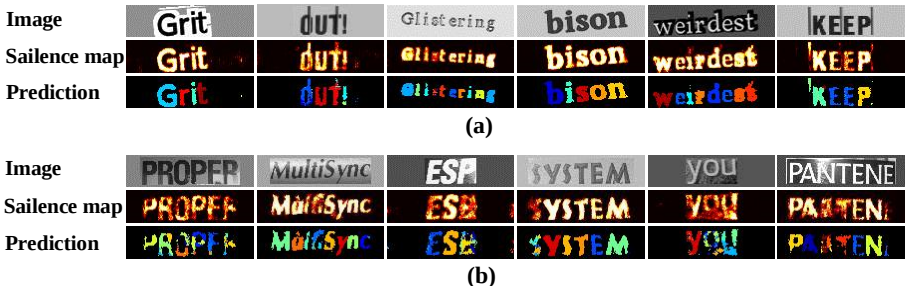


Figure 4.11: Visualization of NSTI Accelerator Output

pixel as shown the third row. In the second row, it merges the 3-D salience map into a 2-D salience map, showing the confidence level of each pixel to the ground true class. In Fig. 4.11(a), it shows some good prediction examples with high confidence level and clean prediction boundary. In Fig. 4.11(b), some bad predictions with low confidence level are shown, which have uneven illumination or low contrast input images. By evaluating the pixelwise classification accuracy, the NSTI accelerator achieves an accuracy of 90% and 91% testing on two public datasets, ICDAR-03 and ICDAR-13, respectively.

The implementation results are summarized in Table 4.3. In this highly paralleled architecture, it is able to map 46 PEs, which contains 193 MMACs (Mega multiply-add operations) in total onto this chip. The total number of operations (MAC operation is counted as 2 operations) in B-CEDNet is 39 G. The NSTI accelerator can work at a frame rate of 34 fps (1,326 GOP/s) with the peak energy efficiency of 7825 GOP/s/W and the real energy efficiency of 698 GOP/s/W. The total power of the NSTI accelerator is 1.9 W with the core consuming 0.8 W. The dynamic power is estimated based upon simulation waveform of the test cases.

The first two columns in Table 4.3 compares exactly the same architecture (B-CEDNet) by GPU (Liu *et al.* (2018a)) and the proposed ASIC implementation. It should be noted that the GPU-based implementation for B-CEDNet (binary) already delivered 8× better throughput than that of the CEDNet (non-binary) (Liu *et al.*

Table 4.3: Experiment Results and Comparison

		This work	Liu	Andri[1]*	Andri[2]*	Pham*	Chen*	Desoli*
ASIC/GPU		ASIC	GPU	ASIC	ASIC	ASIC	ASIC	ASIC
Binary		Yes	Yes	Yes	Yes	No	No	No
Process (nm)		40	28	65	65	45	65	28
Core VDD (V)		0.9	N/A	0.6	1.2	1	0.82-1.17	0.58-1.10
Power (W)		1.9	80	N/A	N/A	0.49	0.16-0.33	0.02-0.09
Freq (MHz)		30	1000	400	N/A	400	100-250	200-1175
Frame rate (fps)		34	200	N/A	707	N/A	57	41
Latency (ms)		40	5	39	N/A	N/A	70	24
Num. of OP		39 G	39 G	1.2 G	1.2 G	N/A	2.7 G	1.3 G
Throughput (GOP/s)	**Peak**	14,868	N/A	N/A	N/A	360	68	473
	Real	1,326	7,800	31	853	331	37	55
Area (mm^2)		12.7	601	2.16 MGE	2.16 MGE	14.06	7.53	50.4
Area efficiency (GOP/s/mm2)		893	N/A	37 GOP/s/MGE	1043 GOP/s/MGE	32	9	294
Energy efficiency (GOP/s/W)	**Peak**	7825	N/A	N/A	N/A	741	414	3064
	Real	698	97	40950	24844	681	112-228	N/A

* Technology scaling to 40 nm (for all the reference with process and core VDD specified):
Delay~1/S, Area~1/S , Power~$1/U^2$, where S=L/40nm, U=V_{DD}/0.9V

(2018a)). Compared with its optimal GPU-based implementation counterpart (Liu *et al.* (2018a)), this work provides 7× better energy efficiency while still maintaining a real-time frame rate with less than 2 W power consumption. Therefore, the proposed accelerator can enable real-time scene text interpretation on the power-constrained mobile devices.

This work is also compared with other ASIC designs for convolutional neural network acceleration (CNN). For a fair comparison, we use the technology scaling rules proposed in (Stillmaker and Baas (2017)) to scale all those reference with process and core VDD specified (marked with * in Table 4.3). All of (Chen *et al.* (2017); Andri *et al.* (2018); Pham *et al.* (2012); Desoli *et al.* (2017)) are general CNN accelerators rather than task-specific ones. These general CNN accelerators are not compatible with the encoder-decoder architecture, which cannot be used to accelerate B-CEDNet.

Reference (Andri *et al.* (2018)) is built upon the binary weight CNN, while (Chen *et al.* (2017); Pham *et al.* (2012); Desoli *et al.* (2017)) are built upon fixed-point CNNs. Compared with the throughput-optimal test set in (Andri *et al.* (2018)), the proposed accelerator achieve 1.5× better throughput in terms of GOP/s. (Since the total number of operations in different network vary a lot, GOP/s is a better reflection of throughput rather than the frame rate.) Compared with fixed-point CNN ASIC designs (Chen *et al.* (2017); Pham *et al.* (2012); Desoli *et al.* (2017)), the proposed accelerator can delivery 4×-36× better throughput. In terms of the latency, even if the number of operations of the proposed network is 14×-32× larger than (Andri *et al.* (2018)[1]) and (Rahmani *et al.* (2018)), the proposed accelerator achieves the best latency among them. Only when compared with (Desoli *et al.* (2017)), its latency is 1.7× better than this work, due to 30× less number of operations in its network. Among all these ASIC designs, the proposed accelerator is the only one that can process the B-CEDNet with 39 Giga operations in a real-time manner and low latency of 40 ms. Binary feature of B-CEDNet enable us to map 46 PEs containing 193 MMACs for massive spatial parallelism. Highly pipelined data flow control enable more temporal parallelism. Both spatial and temporal parallelism contribute to optimize the throughput and latency in the proposed design. The energy efficiency gap between this work and (Andri *et al.* (2018)[2]) can be explained by following points. First, a trade-off is made for a better throughput, since the primary task is to guarantee a real-time throughput. Second, (Andri *et al.* (2018)) has designed customized on-chip memory for a low-power design. Additionally, (Andri *et al.* (2018)) stores the intermediate results (between blocks/layers) and parameters in off-chip DRAM, which are excluded in power consumption reports. The power consumption is under consideration for the entire application rather than just for the computation core. Reduction in computation complexity to bit-level operations

benefits in power saving. Store all the weights and intermediate results on chip eliminating off-chip communication for the sake of extending battery life.

4.5 Summary

This chapter presents an ASIC accelerator for real-time and low-latency natural scene text interpretation on power-constrained mobile devices. The NSTI accelerator takes the cropped scene text image as input and output a salience map for pixelwise classification result. To target a real-time throughput and low latency, a B-CEDNet is adopted as the core architecture to enable massive spatial parallelism. A highly pipelined data flow control is applied to enable temporal parallelism. Moreover, all the binarized intermediate results and parameters are stored on chip to eliminate the power consumption and latency overhead of off-chip commutation. This NSTI accelerator is implemented in a 40nm CMOS technology, which can process 128×32 scene text images at 34 fps with an latency of 40 ms for pixelwise interpretation with accuracy no less than 90%. Its real energy-efficiency is 698 GOP/s/W and its peak energy-efficiency can get up to 7825 GOP/s/W. In the IoT applications, the proposed accelerator can be used in power-constrained edge devices to enable real-time augment reality applications for natural scene understanding.

Chapter 5

COMPRESS BINARY NEURAL NETWORKS VIA SENSITIVITY ANALYSIS

Since CNNs are believed to have huge redundancy, a hypothesize was made that the BNN also has redundancy and it is able to get a more compact BNN. To best knowledge, there is only one related work pruned the first layer of a BNN with the observation of barely any accuracy drop (Guo *et al.* (2018)). Since they only compress the first layer, the impact on the entire network is fairly limited. On the contrary, this chapter proposes the methodology of exploring the BNN redundancy across the entire network.

5.1 BNN Reconstruction

The key concept is to explore the BNN redundancy across the entire network by the bit-level analysis of the input data. The first thing needs to be done is to reformat the input and modifying the first layer for the BNN reconstruction. Then, it shows the redundancy exists in BNN through statistical analysis of the non-binary first layer. At last, the training method is presented.

5.1.1 Bit-sliced Binarized Input

A single image in the dataset can be represented as $D_{(W,H,C)}$, where W is the width, H is the height, and C is the number of channels, as shown in Fig. 5.1. The raw data is usually stored in the format of a non-negative integer with the maximum value of A. Then a lossless conversion from integer (fixed-point) to N-bit binary

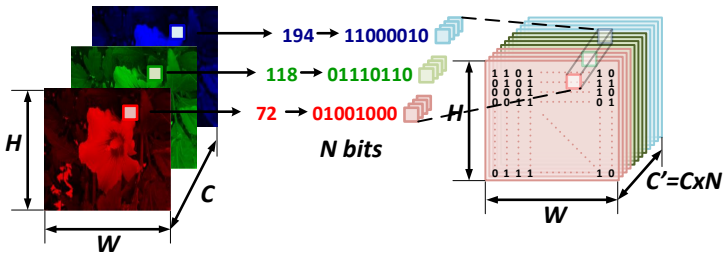


Figure 5.1: Conversion from Fixed-point Input to Bit-sliced Binary Input

format is defined as the $int2b$ function.

$$D^b_{(W,H,C')} = int2b(D_{(W,H,C)}, N), \tag{5.1}$$

where $C' = C \times N$ and $N = ceil(log_2(A+1))$. After $int2b$ conversion, each channel of an image is expanded to N channels in binary format.

5.1.2 Non-binary First Layer

Experimental observation shows that the bit-sliced input has a negative impact on the accuracy rate. There are two main reasons. Since the input data is in the bit-sliced format, the data-preprocessing methods, e.g., mean removal, normalization, ZCA whitening, cannot be applied here, which results in an accuracy drop. In addition, compared with a standard first layer in BNN, the computational complexity drops, which may hurt the accuracy rate. Therefore, the first layer is designed to have full-precision float-point weights to keep the computational complexity of the first layer the same as a standard first layer in BNN.

More importantly, non-binary first layer can help to analyze the importance level of different input bit slices. More specifically, if the weights in first layer are closed to zero, the corresponding input data points will not contribute to the computation in the rest of layers. It can also be interpreted as these input features are filtered out. As shown in Fig. 5.2, input bit slices and first layer weight slices have one-to-

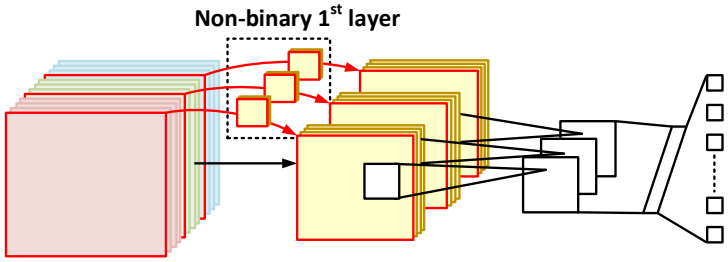


Figure 5.2: Corresponding Relationship Between Input Bit Slices and Non-binary First Layer

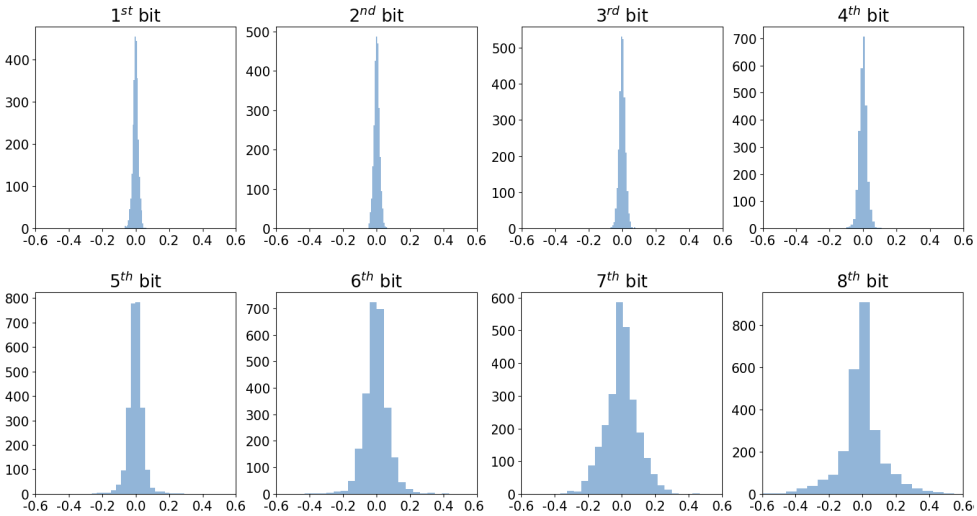


Figure 5.3: Histogram of Distributions of Weight Magnitude Associated with Different Input Bits

one correlations. Thus, the weights associated with the N^{th} input bits are grouped separately for statistical analysis. In Fig. 5.3, it shows the histograms of first-layer weight magnitude distributions associated with different input bit slices. For the weights associated 1^{st}-3^{rd} bit slices, the weight magnitude is very closed to zero. From the weights associated with 4^{th} input bit slice, the weight magnitude spreads out in a wider range. Therefore, it is hypothesized that the lower bits of input slices can be redundant for the classification task.

Although switch the first layer to non-binary makes the network size increased, the growth is somewhat limited. For example, in a 9-layer BinaryNet (Hubara *et al.*

(2016)), the size of the first layer is only 0.02% of the entire network. It has been proved that, with 16-bit quantization of the weights, the NNs are still able to preserve the accuracy (Suda *et al.* (2016)). With the bit-slice input, the network size will slightly increase by 3%, which can be negligible.

With the bit-sliced input and non-binary first layer, the BNN model is reconstructed and refer it as the reconstructed BNN. Although the computational complexity is the same, the new structure helps to reduce the redundancy in BNN, which will be elaborated in the following chapters.

5.1.3 Binary Constrained Training

For BinaryNet-based experiment, the training method is adopted from (Hubara *et al.* (2016)). The objective function is shown in Eq. 5.2, where W_1 represents the weights in the non-binary first layer and W_l represents the weights in all the other binary layers. The loss function L here is a hinge loss. In the training stage, the full-precision reference weights W_l are used for the backward propagation, and. the binarized weights $W_l^b = clip(W_l)$ (Hubara *et al.* (2016)) are used in the forward propagation. As the authors propose in (Tang *et al.* (2017)), the reference weights in the binary layers $W_l (l \geq 2)$ should be punished if they are not close to +1/-1. Also, a L_2 regularization term is applied for the non-binary first layer. In the LQ-Net-based experiment, the training method is exactly the same as proposed in the original paper (Zhang *et al.* (2018a)).

$$J(W_l, W_1, b) = L(W_l^b, W_1, b) + avg(||W_1||_2^2) + \lambda(avg \sum_{l=2}^{L}(1 - ||W_1||_2^2) \tag{5.2}$$

5.2 Sensitivity Analysis

With training method in Chapter 3.3.1, a reconstructed BNN model is trained with the bit-sliced input and non-binary first layer. In the post-training stage, the

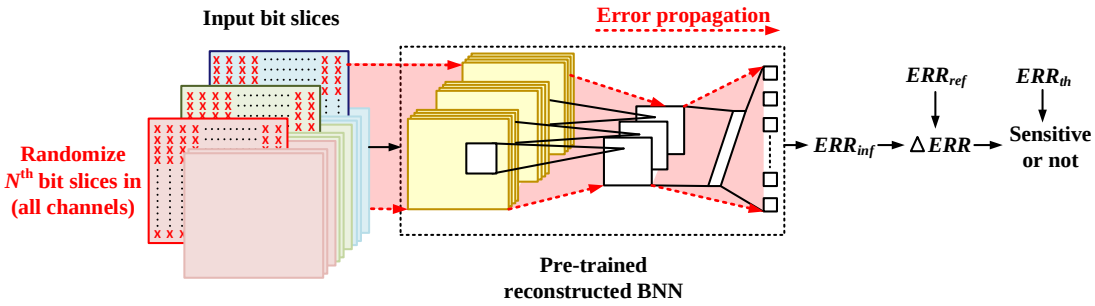


Figure 5.4: Sensitivity Analysis of the Reconstructed BNN with Distorted Input.

redundancy throughout the entire network can be measured, and also the sensitivity of the bit-sliced input to the accuracy performance is evaluated.

As shown in Fig. 5.4, the reconstructed BNN is pre-trained as initial. Then, the N^{th} bit (N^{th} least significant bit) slices in RGB channels are substituted with binary random bit slices. The reason of why to choose binary random bit slices over pruning is that, pruning reduces the size of the network. The experiment needs to eliminate any other factors that can influence the accuracy performance. If the difference between the actual inference error ERR_{inf} and the reference point ERR_{ref} ($\Delta ERR = ERR_{inf} - ERR_{ref}$) is less than an error-tolerant threshold ERR_{th}, the N^{th} bit slices are classified as prunable.

Without retraining the network, the error brought by random bit slices will propagate throughout the entire network as shown in Fig. 5.4. With this tight constraint, if there can be merely no accuracy drop in the inference stage. it can be inferred that these bit slices with less sensitivity to the accuracy performance are useless in the training stage and there are redundant connections throughout the entire network. It also indicates that the existing redundancy in BNN allows us to further shrink the network size. After evaluating the sensitivity of each bit slice, one can also analyze the sensitivity of a stack of slices by using the same method. Then, a collection of insensitive bit slices are found, which are prunable in the training stage. If P out of

N slices are categorized as accuracy insensitive, the number of channels C' can be reduced by N/P times. That is to say, the size of the input array is reduced by N/P times.

5.3 Rebuild a Compact BNN

In the most popular CNN architectures, such as AlexNet (Krizhevsky *et al.* (2012)), VGG (Simonyan and Zisserman (2014)) and ResNet (He *et al.* (2016)), the depth incremental ratio of feature map from one layer to the next layer is either doubled or remaining the same. Intuitively speaking, it is useful to keep the same depth incremental ratio across the entire network. Thus, a good starting point of rebuilding a compact BNN (CBNN) is shrinking the depth of all the layers by N/P times. Since there is a quadratic relation between depth and the net-work size, the reduction of the network size of the CBNN is expected to be $(N/P)^2$ times.

Although this work has not explored how to build an accurate model to optimize the network compression ratio, it emphasizes the entire flow that proves and reduces the redundancy of the entire BNN, and enables speedup in the inference stage with the CBNN.

5.4 Experimental Evaluation

This chapter will first walk through the flow presented in Chapter 5.1 - 5.3 with experimental results on the CIFAR-10 classification task in Chapter 5.4.1. Chapter 5.4.2 will present additional results on SNVH, Chars74K, GTSRB and ImageNet datasets.

For the experiment setup, relative smaller models (AlexNet-scale) are built based upon Hubara et al.'s BinaryNet in Theano and test it with CIFAR-10, SNVH, Chars74K and GTSRB dataset. Due to the severe accuracy drop of fully binarized NNs (such

as BinaryNet) in large models, relative larger models (ResNet-scale) are tested based upon a more relax BNN – LQ-Net with 1-bit weights and 2-bit activations. LQ-Net experiment is conducted in Tensorflow and it is tested with ImageNet dataset. The description of each dataset is listed as follow.

CIFAR-10 (Krizhevsky *et al.* (2009)). This is a dataset for a 10-category classification task with 32×32 RGB images. The training dataset contains 50,000 images and the testing dataset contains 20,000.

SVHN (The Street View House Numbers) (Netzer *et al.* (2011)). This dataset is a real-world house number dataset from Google Street View images. It has 73,257 digits for training and 26,032 digits for testing, with the image size of 32×32.

Char74K (De Campos *et al.* (2009)). This dataset contain 62 characters (0-9, A-Z and a-z) from both natural images and synthesized images. 80% of the Char74K images serve as the training set and the rest 20% serve as the testing set, with the image size of 56×56.

GTSRB (The German Traffic Sign Benchmark) (Stallkamp *et al.* (2011)). This dataset includes 43-class traffic signs. The traffic sign images are resized to 32×32. It has 39,209 training data and 12,630 testing data.

ImageNet (Deng *et al.* (2009)). ImageNet is a large scale dataset which has more than 14 million hand-annotated images. Here, a subset of ImageNet – ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012), which is commonly used for large scale classification task. The ILSVRC2012 dataset covers 1000 categories. The training and testing data contains 1.2 million and 50,000 images, respectively. Average image resolution is around 450x450 pixels.

5.4.1 Experiment on CIFAR-10

Subsections of Chapter 5.4.1 show the experimental results corresponding to the methodology in Chapter 5.1-5.3, respectively.

BNN Reconstruction

Following the input data conversion method in Chapter 5.1.1, the raw data of CIFAR-10 dataset can be denoted as $\text{CIFAR}_{(32,32,3)}$. Each pixel value is represented by a non-negative integer with magnitude $A = 255$. Thus, $N = ceil(log_2(255 + 1)) = 8$ bits are enough for lossless binary representation. Then, the bit-sliced input can be denoted as $\text{CIFAR}^b_{(32,32,24)}$.

An image in CIFAR-10 dataset with different bit-level distortion is shown in Fig. 5.5. This image belongs to the "horse" category. In Fig. 5.5, the left most ones are the same original image without any distortion. The N^{th} bit indicates the N^{th} least significant bit (LSB). The distortion here is injected by replacing the entire bit slice with a randomly generated binary bit map. In Fig. 5.5(a), only one single bit slice get distorted at a time. Since only up to 1/8 elements of CIFAR^b get distorted, all the distorted image can show a clear boundary of the horse with limited noise, except the rightmost one with 7^{th} bit slice gets distorted. If further distortion is applied to CIFAR^b in multiple bit slices from the 1^{st} to N^{th} bit slices, the corresponding images are shown in Fig. 5.5(b) and (c). The images in Fig. 5.5(c) are different from Fig. 5.5(b) that they don't maintain 8-bit precision. Instead, the proposed method directly prunes the 1^{st} to N^{th} bit slices of the images in Fig. 5.5(c). From visualization, in both Fig. 5.5(b) and (c), the turning points is at 5^{th} bit.

Distorted images in Fig. 5.5(a) and (b) are used for sensitivity analysis in reconstructed BNN (Recon. BNN). Pruned images in Fig. 5.5(c) are used for training

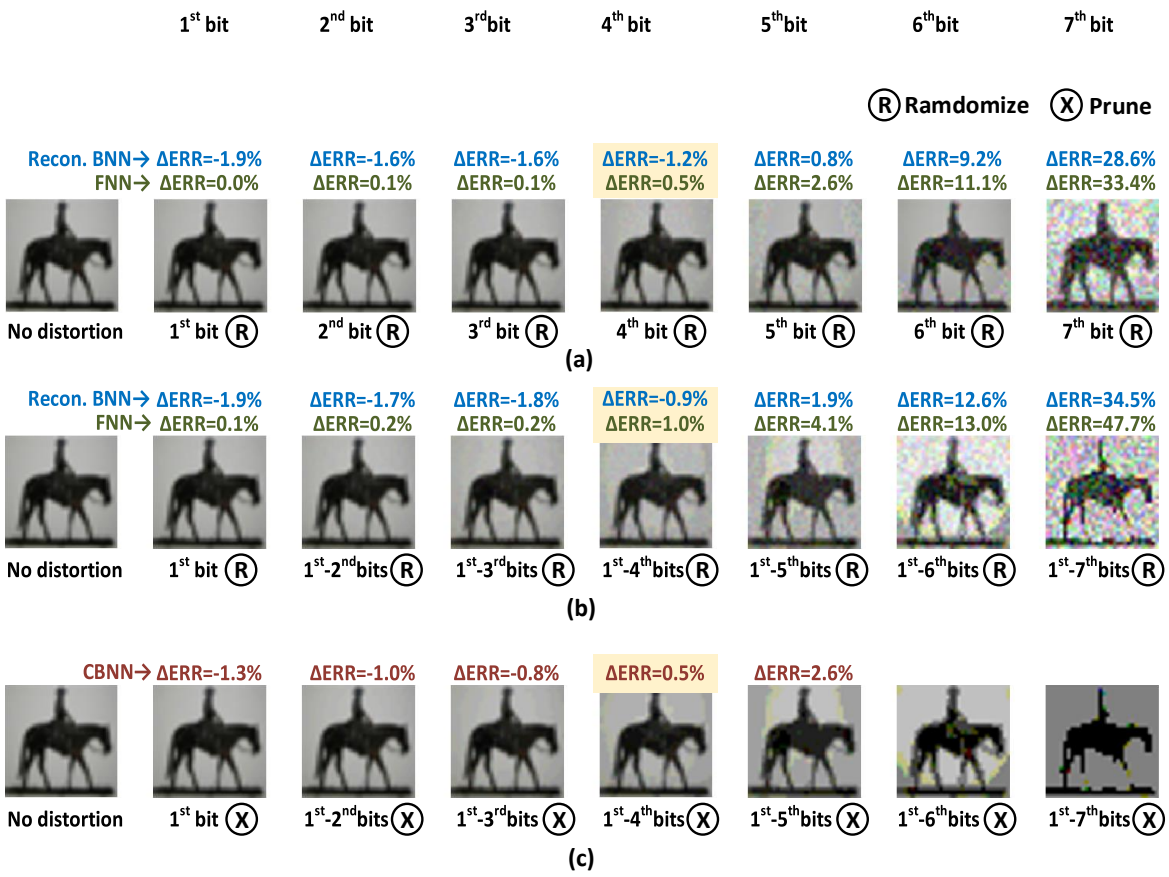


Figure 5.5: Visualization of a Horse Image in CIFAR-10 with Different Bit-level Distortion in Spatial Domain and Frequency Domain.

Table 5.1: Performance Comparison with Different Input Format and 1^{st} Layer Configuration

Arch.	Input	First layer	Network size	Error rate
BNN	full precision	binary	1x	11.6%
FBNN	bit slices	binary	1.01x	14.0%
Reconstructed BNN	bit slices	non-binary	1.1x	10.1%

CBNN.

As illustrated in Chapter 5.1, the proposed structure of the reconstructed BNN is different from the original BNN in both input format and the first layer. Table 5.1 compares the performance results of three network structures with different numerical precision in their input and 1^{st} layer. The baseline BNN design is the one in (Hubara

Table 5.2: Sensitivity Analysis of Single Bit Slice in Each Channel with Random Noise Injected

Arch.	N^{th} bit	ERR/%	ΔERR/%	Arch.	N^{th} bit	ERR/%	ΔERR/%
BNN	0	11.6	0.0	**FNN**	0	10.4	0.0
Recon. BNN	0	10.1	-1.5	**FNN**	0	10.4	0.0
	1	9.8	-1.9		1	10.4	0.0
	2	10.0	-1.6		2	10.4	0.1
	3	10.1	-1.6		3	10.4	0.1
	4	**10.5**	**-1.2**		**4**	**10.9**	**0.5**
	5	12.5	0.8		5	13.0	2.6
	6	20.9	9.2		6	21.4	11.1
	7	40.3	28.6		7	43.8	33.4

et al. (2016)), with full precision input and a binarized 1^{st} layer. Here a CNN with bit-sliced input, binarized weights and activations is defined as a FBNN. A FBNN has bit slices input but BNN does not. By training with the method in Chapter 5.1, FBNN shows 2.4% in the accuracy drop, compared with BNN. The accuracy here is affected by computational complexity degradation in the 1^{st} layer and unnormalized input data. It also gives us some insights that the FBNN is hard to get a good accuracy rate, which is in accord with Tang et al.'s opinion in (Tang *et al.* (2017)). By introducing bit slices input and non-binary 1^{st} layer to reconstruct the BNN, the accuracy drop can be compensated as shown in Table 5.1. It's able to even get a better error rate than the baseline BNN with a slightly increased network size. It also gives more margin in compressing the network.

Sensitivity Analysis of the Reconstructed BNN

With a pre-trained reconstructed BNN presented in the last chapter, now it's able to do bit-level sensitivity analysis as stated in Chapter 5.2.

First, analyze the sensitivity of a single bit slice. The results are shown in Table 5.2. The data shows in Table 5.2 is the average over 10 trials. In addition to the reconstructed BNN, evaluation is also performed on the bit-level sensitivity of the

Table 5.3: Sensitivity Analysis of 1-N^{th} Multiple Bit Slices in Each Channel with Random Noise Injected

Arch.	1-N^{th} bits	ERR/%	ΔERR/%	Arch.	1-N^{th} bits	ERR/%	ΔERR/%
BNN	0	11.6	0.0	**FNN**	0	10.4	0.0
Recon. BNN	0	10.1	-1.5	**FNN**	0	10.4	0.0
	1	9.8	-1.9		1	10.4	0.1
	1-2	9.9	-1.7		1-2	10.5	0.2
	1-3	9.9	-1.8		1-3	10.5	0.2
	1-4	**10.7**	**-0.9**		**1-4**	**11.3**	**1.0**
	1-5	13.6	1.9		1-5	14.4	4.1
	1-6	24.3	12.6		1-6	23.3	13.0
	1-7	46.1	34.5		1-7	54.1	43.7

input with its full-precision counterpart, which is denoted as FNN. With FNN, it intends to show that the data itself has redundancy, which can be reflected in both binary domain or fixed-point domain with the same pattern. The architecture in the first row is taken as the reference design. The 1^{st} row of ERR column is the ERR_{ref} and the others are ERR_{inf}. $\Delta ERR = ERR_{inf} - ERR_{ref}$. BNN is the reference design for the reconstructed BNN. FNN with non-distorted input is the reference design for the full-precision ones. It is interesting that the 1^{st}, 2^{nd} and 3^{rd} bit slices are at the same sensitivity level, concluded from the almost unchanged ΔERR. In sensitivity analysis, the point where ΔERR flips the sign or increases abruptly is defined as the turning point. The turning point here is the 5^{th} bit.

Second, analyze the sensitivity of bit slices stacks. Each stack contains 1^{st} to N^{th} bit slices in each color channel. The results are shown in Table 5.3. For the 1^{st}, 2^{nd} and 3^{rd} bit slices, it makes no difference if distortion is injected in one of them or all of them. The 4^{th} makes a slight difference of around 0.5% accuracy drop and the 5^{th} bit is also the turning point with around 3% accuracy drop.

Even when randomize 50% of the entire input values (1^{st} to 4^{th} bit slices) and the variation propagates through the entire network, the accuracy doesn't change much.

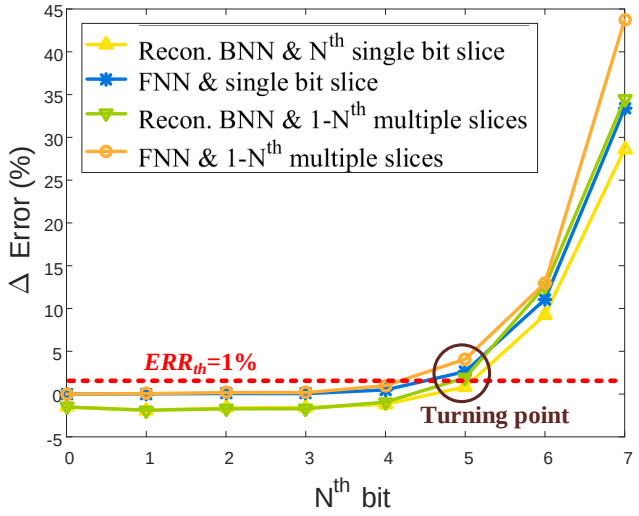


Figure 5.6: Error Rate of Randomizing One or Multiple Bit Slices in Sensitivity Analysis.

Therefore, these bits are useless in the training stage. This validates the hypothesis that the BNN still has redundancy. In Fig. 5.6, the error rate turning point is circled at the 5^{th} bit slice. The trend of error rate in the binary domain and full-precision domain (shown in Fig. 5.6) align well. Simply setting an error-tolerant threshold ERR_{th} to determine how many bits are prunable can help the entire process to be automatic. Here, ERR_{th} is set to 1%. It can be concluded that 1^{st}-4^{th} bit slices here are redundant and prunable through bit-level sensitivity analysis. Accordingly, the reconstructed BNN can be shrunk to reduce the redundancy and get a more compact architecture.

Rebuild a Compact BNN (CBNN)

Since 4 out of 8 bit slices are prunable, one can rebuild a compact BNN with the depth of each layer shrunk by half. The performance of CBNN is shown in Table 5.4. CP. Ratio represents compression ratio and GOPs stands for Giga operations (one operation is either an addition or a multiplication). Regarding the network size,

Table 5.4: Performance of CBNNs on CIFAR-10

Arch.	1-Nth bits	*ERR* %	Δ*ERR* %	Network size		GOPs	
				MB	CP. ratio	#	CP. ratio
BNN	0	11.6	0.0	1.75	1x	1.23	1x
CBNN	1	10.3	-1.3	1.38	1.3x	0.98	1.3x
	2	10.6	-1.0	1.02	1.7x	0.72	1.7x
	3	10.8	-0.8	0.71	2.5x	0.50	2.5x
	4	**11.8**	**0.2**	**0.45**	**3.9x**	**0.32**	**3.8x**
	5	14.2	2.6	0.25	7.0x	0.18	6.8x

16 bits are used for measuring non-binary weights in the 1^{st} layer, since it has been proved that 16-bit precision is enough to maintain the same accuracy (Suda *et al.* (2016)). The alternatives of pruning 1-N^{th} ($N = 1, 2, ..., 5$) bit slices and shrink the layerwise depth by 1/8 to 5/8 are also provided. The results align with the sensitivity analysis that 1-3^{rd} bit slices have little impact on the classification performance. The choice of pruning 1-4^{th} bit slices is the optimal one to maximize the compression ratio with <1% accuracy drop. Since the size of the 1^{st} layer is larger than that of BNN, ideal network size compress ratio (4x) cannot be achieved regarding the entire network. The actual compression ratio of the network size is 3.9x and the compression ratio of number of GOPs is 3.8x.

5.4.2 Experiment on Other Datasets

This chapter will skip the sensitivity analysis and just show the result comparison between the baseline and the final CBNNs processed in the same procedure.

For SVHN and Char74K datasets, the baseline architecture has half of the depth in each layer as the one for CIFAR-10. For GTSRB, the baseline architecture has the same filter configuration as the one for CIFAR-10. Since the input size of GTSRB is larger than CIFAR-10, so the network for GTSRB has the same depth but larger width and height in each layer. For ImageNet dataset, the same ResNet-18 architecture as

Table 5.5: Performance Results of CBNNs on SVHN, Chars47k, GTSRB and ImageNet Datasets

Arch.	Dataset	1-Nth bits	*ERR* %	Δ*ERR* %	Network size		GOPs	
					MB	CP. ratio	#	CP. ratio
Binary-Net	SVHN	0	4.8	0.0	0.44	1x	0.31	1x
		1	4.9	0.1	0.36	1.2x	0.26	1.2x
		2	5.1	0.3	0.26	1.7x	0.19	1.6x
		3	**5.0**	**0.2**	**0.18**	**2.4x**	**0.13**	**2.4x**
		4	6.6	1.8	0.12	3.7x	0.08	3.7x
	Chars47k	0	15.4	0.0	0.44	1x	0.31	1x
		1	15.3	-0.1	0.36	1.2x	0.26	1.2x
		2	15.3	-0.1	0.26	1.7x	0.19	1.6x
		3	15.2	-0.2	0.18	2.4x	0.13	2.4x
		4	**16.3**	**1.0**	**0.12**	**3.7x**	**0.08**	**3.7x**
	GTSRB	0	1.0	0.0	1.81	1x	3.89	1x
		1	1.0	0.0	1.39	1.3x	2.98	1.3x
		2	1.2	0.2	1.02	1.8x	2.19	1.8x
		3	1.6	0.6	0.71	2.5x	1.52	2.6x
		4	**2.0**	**1.0**	**0.46**	**3.9x**	**0.97**	**4.0x**
LQ-Net	ImageNet	0	37.1	0.0	150	1x	3.0G	1x
		1	37.1	0.0	115	1.3x	2.6G	1.3x
		2	37.2	0.1	113	1.8x	2.3G	1.8x
		3	**38.5**	**1.5**	94	**2.5x**	**1.9G**	**2.6x**

Zhang et al.'s work is used.

In Table 5.5, it shows the performance results of CBNNs evaluating on different datasets and network setting. The baseline for each dataset is shown in the first row of each dataset region. For Chars47k and GTSRB, the CBNNs are able to maintain no more than 1% accuracy drop, achieving 3.7x and 3.9x network size reduction, respectively. For SVHN dataset, the accuracy drop between pruning 1-3rd bits and pruning 1-4th bits is large. In order to preserve no more than 1% accuracy drop, the network size reduction is yield to 2.4x. For ImageNet dataset, the accuracy drop 1.5%, while gaining 2.5x network size reduction.

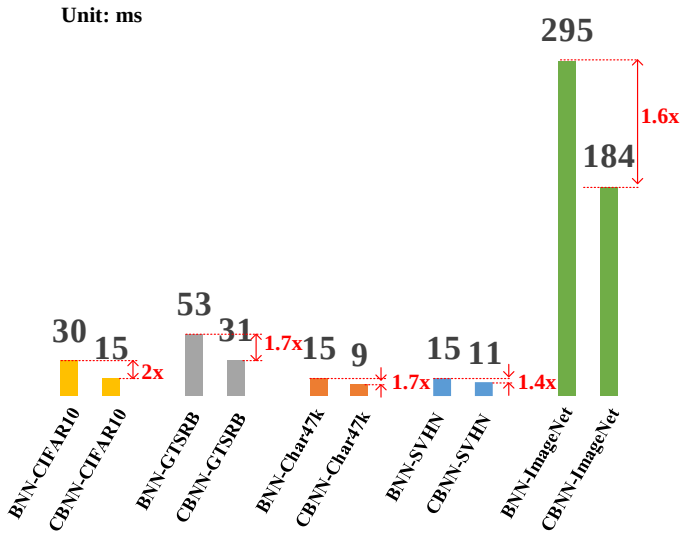


Figure 5.7: Runtime Comparison of Different Network Compression Technique.

5.4.3 Runtime Evaluation

The actual runtime performance of CBNNs is evaluated on Nvidia GPU Titan X. The batch size is fixed as 128 in all the experiments. XNOR-based GPU kernel (Hubara *et al.* (2016)) is used for CBNN implementation. The computational time is calculated by averaging over 10 runs.

Fig. 5.7 illustrates the actual runtime and runtime speedup of 4 CBNNs compared with their baseline BNNs. The configurations are the same as the highlight ones in Table 5.4 and Table 5.5. For the CBNNs processing CIFAR-10, GTSRB, Char47k and ImageNet datasets, their network size and total GOPs shrink 3.7-4.0x, resulting in the speedup of 1.6-2.0x. For the CBNN processing the SVHN dataset, its network size and total GOPs shrinks 2.4x, resulting in a speedup of 1.4x. As it is proved in (Han *et al.* (2016)), combining pruning, quantization and Huffman coding technique, an FNN can achieve up to 4x speedup. (Hubara *et al.* (2016)) demonstrate that a multilayer perceptron BNN can get 5x speedup compared with its full-precision counterpart. On

top of the BNN, the proposed CBNN can give extra 1.4-2.0x speedup. Therefore, the CBNN can achieve 7.0-9.9x speedup compared with FNN.

5.5 Summary

In this chapter, a novel flow is proposed to explore the redundancy of BNN and remove the redundancy by bit-level sensitivity analysis and data pruning. In order to build a compact BNN, one should follow these three steps. Specifically, first reconstruct a BNN with bit-sliced input and non-binary 1^{st} layer. Then, inject randomly binarized bit slices to analyze the sensitivity level of each bit slice to the classification error rate. After that, prune P accuracy insensitive bit slices out of total N slices and rebuild a CBNN with depth shrunk by (N/P) times in each layer. The experiment results show that the error variation trend in sensitivity analysis of the reconstructed BNN is well aligned with that of CBNN. In addition, the CBNN is able to get 2.4-3.9x network compression ratio and 2.4-4.0x computational complexity reduction (in terms of GOPs) with no more than 1% accuracy loss compared with BNN. The actual runtime can be reduced by 1.4-2x and 7.0-9.9x compared with the baseline BNN and its full-precision counterpart, respectively.

Chapter 6

PRUNING BINARY NEURAL NETWORK VIA WEIGHT FLIPPING FREQUENCY

As both 0s and 1s are non-trivial in BNNs, it is not proper to adopt any existing pruning method of full-precision networks that interprets 0s as trivial. In this chapter, a pruning method tailored to BNNs is presented and, it also illustrates that BNNs can be further pruned by using weight flipping frequency as an indicator of sensitivity to accuracy.

6.1 Preliminary

In this chapter, the pruning methods that involve multiple rounds of pruning and retraining in a iterative fashion is referred as iterative pruning. The other pruning methods that prune a network based on optimization techniques without the need of retraining is referred as optimization-based pruning.

6.1.1 Iterative Pruning

full-precision neural networks (Han *et al.* (2016); Luo *et al.* (2017); Molchanov *et al.* (2019)) generally follow a three-step training pipeline, as shown in Fig. 6.1. If not starting from a pre-trained floating-point network, training one is the first step. The second step is to prune out redundant connections determined by certain threshold values. Intuitively, if the absolute values of weights are sufficiently small, their influence on the output values is fairly limited. The third step is to fine-tune the network to re-train the inherited weights. The second and third steps are repeated to further prune and fine-tune the network in an iterative fashion until satisfactory

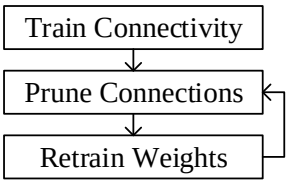


Figure 6.1: The Overview of the General Pruning Flow.

performance is achieved. Unfortunately, it is impossible to directly apply the iterative pruning method developed for full-precision networks to BNNs. Since 0s and 1s are both non-trivial in BNNs, the absolute values of the binarized weights are no longer a valid indicator of their sensitivity to accuracy thus cannot be used to guide the pruning of BNNs. Therefore, it is improper to adopt any iterative pruning methods for full-precision networks that interpret 0s as trivial in BNN pruning.

6.1.2 Optimization-based Pruning

For the same reason, the optimization-based pruning methods (Molchanov *et al.* (2019); Yu *et al.* (2018)) that adopt a pruning objective function based upon the exact values of network weights cannot be applied to pruning BNNs (Molchanov *et al.* (2019)) either. In case that the pruning objective function is based upon the separated masking arrays of weights (Yu *et al.* (2018)), the pruned network models will have unstructured weights. As a result, their hardware implementations will require extra memory to store the flags of prunable weights as well as extra logic for manipulating the memory pointers to skip the corresponding computation, which creates difficulty in achieving actual runtime speedup on CPUs and GPUs.

6.2 Methodology

To tackle the above-mentioned problems, a novel BNN-pruning method is proposed, which adopts weight flip frequency (f) as an indicator of the sensitivity of binary weights to accuracy to guide the pruning of BNNs. Experiments are conducted to empirically validate that a high and low f indicates a low and high sensitivity of binary weights to accuracy, respectively. Therefore, f is an effective indicator for identifying the binary weights that have a high criticality to pruning and can be pruned with a well-bounded accuracy drop.

6.2.1 Weight Flip Frequency

In this chapter, weight flip frequency (f) is defined as the flipping count of a weight during a specified last stage of training (from $epoch_{start}$ to $epoch_{end}$). In BNNs, weight flipping means that a weight value switches from 0/-1 to 1 or 1 to 0/-1. During the specified training interval, f is increased by 1 whenever the weight value flips. A hypothesis is made that, toward the end of the training, the last few percent of accuracy gain stems from the update of the weights with a high weight flipping frequency, and these weights have little influence on the accuracy that has already been established. Therefore, f can be used as the indicator of the sensitivity of binary weights to accuracy as well as their criticality to pruning . An f being equal to 0 or 1 means that the weights belong to this f group remain stable or become stable

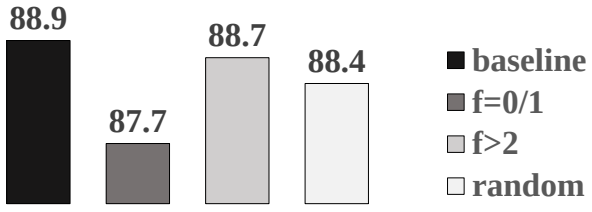


Figure 6.2: Accuracy Comparison for Randomizing Different Groups of Weights.

toward the end of the training, which implies that, if one flips their values, it is most likely to hurt the accuracy. On the contrary, an f being high or hitting the upper bound (the weight flips in every iteration) means that the values of this group of "noisy weights" most likely has little influence on the accuracy gain. Such behavior can be interpreted as that the training is trying fine-tune these "noisy weights" to further shape the decision boundary of classification but failed as they turned out to have little impact on the accuracy gain.

The experiments are conducted to validate our hypothesis. The experiments are conducted on the CIFAR-10 (Krizhevsky *et al.* (2009)) dataset with a 9-layer binarized NIN (Lin *et al.* (2013)). The network architecture is the same as the one described in Lin *et al.* (2013). The training interval for f analysis is set as the epoch range that contribute to the last 1% of accuracy gain toward the end of training. The weights are partitioned into three groups. The 1^{st} and 2^{nd} group of weights satisfies f=0/1 and f=2, respectively. The 3^{rd} group of weights contain the same amount of weights as the 1^{st} group, but are randomly selected from the entire weight set. Fig. 6.1 shows the accuracy comparison when randomizing different groups of weights during inference. Note that only weight randomization is performed in the experiments, and no fine-tuning of the weights is performed after the randomization. The results are averaged over ten trials. The experiment results show a 1.1% gap between the baseline model and the one with the 1^{st} group of weights randomized. Also, randomizing the 1^{st} group of weights achieves even lower accuracy than randomizing the same amount of randomly selected weights (the 3^{rd} group).

These results indicate that the 1^{st} group of weights, where f is equal to 0 or 1, is the most sensitive to the final accuracy rate, while the 2^{nd} group of weights with a high f has little impact. This validates our hypothesis and suggests that $f \geq 2$ is a viable threshold for identifying the prunable binary weights.

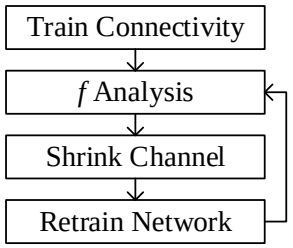


Figure 6.3: The Overview of the BNN Pruning Flow.

6.2.2 BNN Compression Flow

The overall flow of BNN pruning is shown in Fig. 6.3, and Fig. 6.4 illustrates the training interval during which f analysis shall be performed. Δacc is the accuracy drop tolerance defined by the user. $epoch_{end}$ is the final epoch of the training phase. $epoch_{start}$ is the starting point of f analysis, where the accuracy at $epoch_{start}$ is Δacc lower than the final accuracy at $epoch_{end}$.

The first step in the BNN-pruning method is to train a BNN from scratch. In the case of a pre-trained BNN model is available, the last stage of training (from $epoch_{start}$ to $epoch_{end}$) needs to be performed again for the purpose of f analysis.

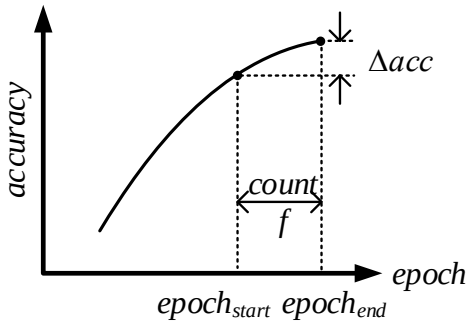


Figure 6.4: The Illustration of Training Interval for f Analysis.

The second step to analyze the weight flip frequency – f. One shall log the statistics of f for each weight during the training interval from $epoch_{start}$ to $epoch_{end}$. Subsequently, one shall calculate the portion of insensitive weights ($p_L\%$) in the L^{th} layer that satisfy $f \geq 2$ for each layer.

The third step is to reduce the size of the BNN by shrinking the number of channels in the L^{th} layer by $p_L\%$, and the forth step is to retrain the network at the reduced size. One shall repeat the second, third, fourth steps in a iterative fashion until the maximum percentage of insensitive weights ($p_L\%$) is close to 0, which indicates that there is no room for further compression.

Preserving the pruned architecture and fine-tune the inherited weights in BNN pruning is not recommended for two main reasons. First, the insensitive weights are found to be sparsely distributed and unstructured in our experiments. Unstructured pruning can hardly result in any saving for BNNs as one has to introduce memory overhead to label the prunable weights that only have 1 bit as well as extra logic for manipulating the memory points to skip the corresponding computation. Second, prior work shows that retraining inherited weights in conventional pruning methods typically does not produce better performance than simply training a smaller model from scratch (Liu *et al.* (2018b)). Therefore, it is recommended reducing the size of the BNN network at each layer guided by the f analysis and $p_L\%$ and retraining the smaller BNN model in BNN pruning. Comparing to the intuitive approach of exploring the appropriate network size through binary search, the proposed BNN-pruning method can quantitatively analyze the layer-wise redundancy to effectively reduce the BNN size, which greatly reduces the search space and time.

Table 6.1: The Layer-wise BNN-pruning Results of the Binarized NIN at Each Iteration.

layer	filter	N^{th} iteration								
		baseline	1st		2nd		3rd		4th	
			p_L/%	channels	p_L/%	channels	p_L/%	channels	p_L/%	channels
conv2d	5x5	192	N/A	192	N/A	192	N/A	192	N/A	192
binconv2d	1x1	160	5.0	152	5.9	143	0.7	142	3.5	137
binconv2d	1x1	96	5.2	91	5.5	86	1.2	85	4.7	81
pooling										
binconv2d	5x5	192	1.5	189	1.1	187	0.5	187	1.0	185
binconv2d	1x1	192	9.9	173	5.8	163	3.1	158	6.3	148
binconv2d	1x1	192	3.1	186	2.7	181	0.6	180	2.2	176
pooling										
binconv2d	3x3	192	1.6	189	1.1	187	0.5	187	1.1	185
binconv2d	1x1	192	4.2	182	4.9	173	1.2	171	3.5	165
conv2d	1x1	10	N/A	10	N/A	10	N/A	10	N/A	10
pooling										
GOPs (M)	220		206		193		190		180	

6.3 Experimental Evaluation

The proposed BNN-pruning method is tested with the CIFAR-10 (Krizhevsky *et al.* (2009)) dataset on two BNNs: the binary versions of a 9-layer binarized NIN (Lin *et al.* (2013)) and the AlexNet (Krizhevsky *et al.* (2012)). The binarization method used in this chapter is introduced by the XNOR-net (Rastegari *et al.* (2016)). The source code we build is based on (Yu (2019)).

The first set of experiments is conducted on the binarized NIN. The network architecture used in our experiment is shown in Table 6.1. Table 6.1 also shows the layer-wise pruning results in four iterations of BNN pruning. Note that the 1^{st} and the last layer are floating-point layers, and we do not apply any pruning on them. In each iteration, our target Δacc is set to 0.5%. After four iterations of BNN pruning, the final network model has a 20% reduction in GOPs (Giga operations). The accuracy of the baseline model is 86.5%, and the accuracy of the final network model is subject to the target accuracy drop of 0.5%. Compared with directly performing a binary search

Table 6.2: The Layer-wise BNN-pruning Results of the Binarized AlexNet at Each Iteration.

layer	filter	N^{th} iteration								
		baseline	1st		2nd		3rd		4th	
			p_L/%	channels	p_L/%	channels	p_L/%	channels	p_L/%	channels
con2d	3x3	128	N/A	128	N/A	128	N/A	128	N/A	128
binconv2d	3x3	128	5.5	121	3.3	117	5.1	111	1.8	109
pooling										
binconv2d	3x3	256	6.3	240	4.6	229	2.6	223	2.7	217
binconv2d	3x3	256	8.2	235	7.2	218	6	205	4.4	196
pooling										
binconv2d	3x3	512	4.5	489	2.2	478	3.6	461	1.3	455
binconv2d	3x3	512	1.8	503	2.8	489	0.2	488	1.4	481
pooling										
GOPs (M)	640		579		535		502		481	

for the appropriate network size, the proposed BNN-pruning method can provide a layer-wise, quantitative guideline to effectively shrink the BNN size. For the runtime evaluation, we tested the layer-wise runtime of the binarized NIN using an optimized binary kernel on an NVIDIA TitanX GPU. The final network model after BNN pruning leads to a 15% runtime speedup as comparing to the baseline model.

In addition, we tested the proposed BNN-pruning method on the binarized AlexNet with the CIFAR-10 dataset (Krizhevsky *et al.* (2009)), as shown in Table 6.2. In this set of experiments, we set the target Δacc to 1.0%. After four iterations of BNN pruning, the final network model results in a 40% GOPs reduction and a 25% runtime speedup on an NVIDIA TitanX GPU, subject to the 1.0% target accuracy drop. The BNN pruning results and their impact on runtime speedup are summarized in Table 6.3. Overall, the experiment results show that using the weight flipping frequency as

Table 6.3: Experiment Results of BNN Pruning.

Arch.	GOPs reduction	Speedup	Δacc
NIN	20%	15%	0.5%
AlexNet	40%	25%	1.0%

the indicator of weight sensitivity to guide BNN pruning can lead to 20-40% GOPs reduction and 15-25% runtime speedup at the limited cost of 0.5-1.0% accuracy drop.

6.4 Summary

This chapter presents a study that explores the weight redundancy in BNNs and propose a generic solution for BNN pruning. The weight flipping frequency f is an effective indicator of the sensitivity of binary weights to accuracy and their criticality to pruning. Reducing the BNN size by shrinking the number of channels in the L^{th} layer by a factor of $p_L\%$ is the key to effectively removing redundancy in BNN pruning.

Chapter 7

LIGHT-WEIGHT OBJECT DETECTION NETWORKS

Object detection is the key module in face detection, tracking objects, video surveillance, pedestrian detection, etc. (Redmon *et al.* (2016); Redmon and Farhadi (2017)). With the recent development of deep learning, it boosts the performance of object detection tasks. However, regarding the computational complexity (in terms of FLOPs), a detection network can possibly consume three orders of magnitude more FLOPs than a classification network, which makes it much more challenging to be deployed on an edge device.

There are two common methods to reduce the FLOPs in a detection network. One way is to switch to another backbone, while the other is to reduce the input image size. The first method results in noticeable accuracy drop if substitutes a ResNet backbone (He *et al.* (2016)) with a more shallow one. Typically it is not considered as a good accuracy-FLOP trade-off scheme with a small variation. With regard to reducing the input image size, it is an intuitive way to reduce the FLOPs. However, the accuracy-FLOP trade-off curve shows degradation in a polynomial trend (Huang *et al.* (2017)). There is an opportunity to find a more linear degradation tendency curve for a better accuracy-FLOP trade-off. This work proposes only to replace certain branches/layers of the detection network with light-weight architecture and keep the rest of the network unchanged. For the RetinaNet, the heaviest branch is the succeeding layers of the finest FPN (P3 in Fig. 7.1), which takes up to 48% of the total FLOPs. This work proposes different light-weight architecture variants. Moreover, the proposed method can also be applied to other blockwise-FLOPs-imbalance detection networks.

7.1 Preliminary

In this chapter, the mAP is used as the indicator to categorize the existing object detection solutions. The mAP-20-tier solutions are the most aggressive ones that target highly energy- and resource-constrained devices, such as battery-powered mobile devices. The existing solutions, such as YOLOv1, YOLOv2, SSD, MobileNetv2-SSDLite (Redmon *et al.* (2016); Redmon and Farhadi (2017); Liu *et al.* (2016)) have pushed hard to reduce the memory consumption by trading off their accuracy performance. Their detection accuracy on the large dataset (COCO test-dev 2017 (Lin *et al.* (2014))) yields to mAP of 22-25%. The mAP-40-iter solutions, such as MaskRCNN and its variations, on the contrary, are targeting the best mAP performance with less concern about computation resources. The mAP-30-iter solutions do not scarify the accuracy performance too much but are more aware of the computational efficiency. In the mAP-30-tier, popular solutions include Faster R-CNN, RetinaNet, YOLOv3 (Ren *et al.* (2015); Lin *et al.* (2017c); Redmon and Farhadi (2018)), and their variants. These solutions can be potentially deployed on edge GPUs (e.g., Nvidia T4 GPU) or FPGAs (e.g., Intel Arria 10 FPGA based acceleration card), since the on-board memory resources are generally enough for preloading the weights of the mAP-30-tier networks. (Huang *et al.* (2017)) verifies the linear relation between FLOP count and inference runtime for the same kind of network.

RetinaNet is taken as the baseline since it has the best FLOP-mAP trade-off in the mAP-30-tier. The FLOP count is used as a key indicator for comparison. By applying Faster R-CNN (Ren *et al.* (2015)), RetinaNet (Lin *et al.* (2017c)) and YOLOv3 (Redmon and Farhadi (2018)) on the same task for COCO detection dataset, which takes an input image around 600×600-800×800, the mAP will hit in the range of 33%-36%. However, the FLOPs of Faster R-CNN (Ren *et al.* (2015)) is around 850

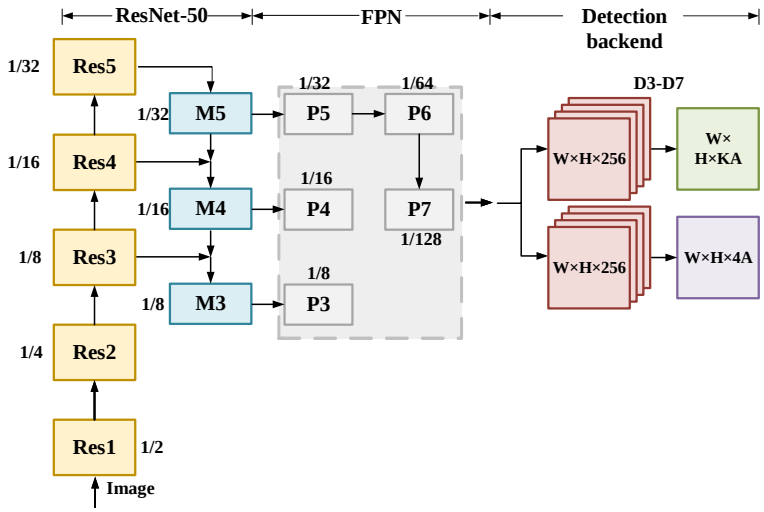


Figure 7.1: RetinaNet (ResNet50-FPN-800x800) Network Architecture.

GFLOPs (gigaFLOPs), which is at least 5x more than that of RetinaNet and YOLOv3 (Redmon and Farhadi (2018)). Apparently, Faster R-CNN is not competitive in terms of computational efficiency. From YOLOv2 (Redmon and Farhadi (2017)) to YOLOv3 (Redmon and Farhadi (2018)), it is interesting that the authors have aggressively increased the number of FLOPs from 30 to 140 GFLOPs to gain mAP improvement from 21% to 33%. Even with that, the mAP of YOLOv3 is 2.5% lower than RetinaNet with 150 GFLOPs. Also, a low-end version of MaskRCNN (He *et al.* (2017)) with mAP of 37.8% cannot beat RetinaNet in terms of runtime. These observations inspire us to take the RetinaNet as the baseline to explore the feasibility of creating a light-weight version of it.

In the following paragraphs, the RetinaNet network is analyzed with a focus on the distribution of the number of floating-point operations (FLOPs) across different layers. Then, the approach of building a light-weight RetinaNet is discussed.

The RetinaNet architecture is composed of three parts – a backbone, a feature pyramid network (FPN) (Lin *et al.* (2017b)), and a detection backend, as shown in Fig. 7.1. The image is first processed by the backbone, which usually is the ResNet

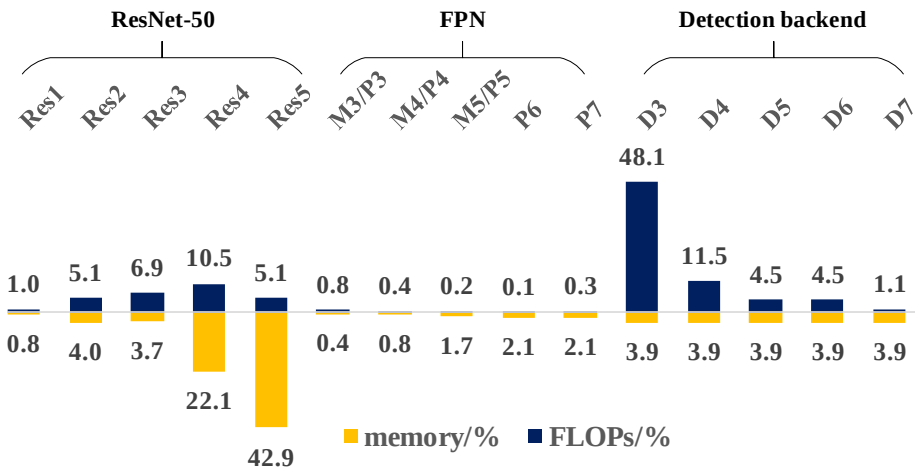


Figure 7.2: The FLOPs and Memory (Parameter) Distribution of RetinaNet Across Different Blocks.

Architecture. Here, it is worthy of noting that although MobileNet's performance (Howard *et al.* (2017)) is on a par with ResNet in classification tasks, MobileNet may not be a good alternative to ResNet for detection tasks. From both (Huang *et al.* (2017)) and the author's observation, using MobileNet (Howard *et al.* (2017)) as the backbone for detection tasks will suffer from much more accuracy drop than it does for classification tasks. The main reason is that the confidence scores of a MobileNet-based backbone are the trade-off for lower computation costs. Therefore, a MobileNet-based backbone is hardly a desirable choice for high precision object detection networks. The backbone, together with the subsequent FPN forms an encoder-decoder-like network. The benefit of the FPN is that it merges the features of consecutive layers from the coarsest to the finest level. After that, the multi-scale pyramid features (P3-P7) feed into the backend where two detection branches are used for bounding box regression and object classification. Note that the detection branch and bounding box branch do not share weights. While the weights of each branch are shared across the pyramid features (P3-P7).

The FLOP distribution of RetinaNet across different blocks is shown in Fig. 7.2., where each block is corresponding to the same block in Fig. 7.1. The detection back-

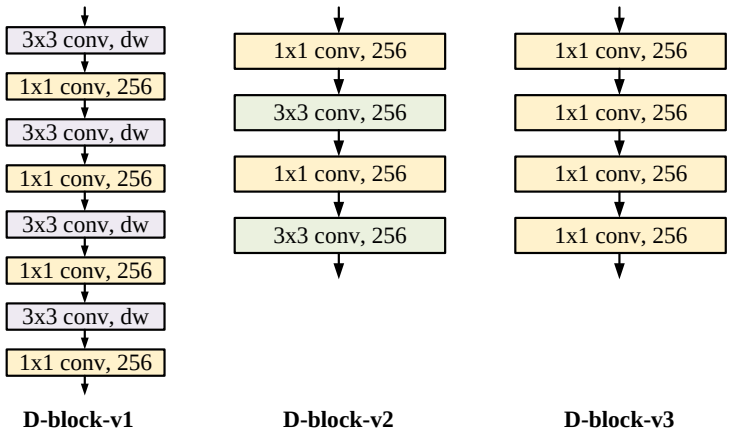


Figure 7.3: Light-weight Blocks for Detection Backend.

end D3-D7 is the succeeding layer of P3-P7, respectively. As in the original design, D3-D7 share the same weight parameters, the average memory cost of D3-D7 is shown in Fig. 7.2. The FLOP count of the D3 block dominates the total FLOP count at 48.1%. This unbalanced FLOP distribution is quite different from that of the ResNet architecture, which has a small FLOP count variance across different blocks. The unbalanced FLOP distribution presents an opportunity to get a meaningful overall FLOP reduction at little cost of accuracy drop by only reducing computational complexity of the heaviest layer. Specifically, the FLOPs of D3 can be reduced by half, the total FLOPs can be reduced by 24%.

7.2 Architecture Design

7.2.1 Light-weight Block

Intuitively, it is able to get FLOP reduction by reducing the filter size. As shown in Fig. 7.3, different block designs are proposed for the detection branches of ResNet. The D-block-v1 applies the MobileNet (Howard *et al.* (2017)) building block. A 3×3 depth-wise (dw) convolution is followed by a 1×1 convolutional block to substitute

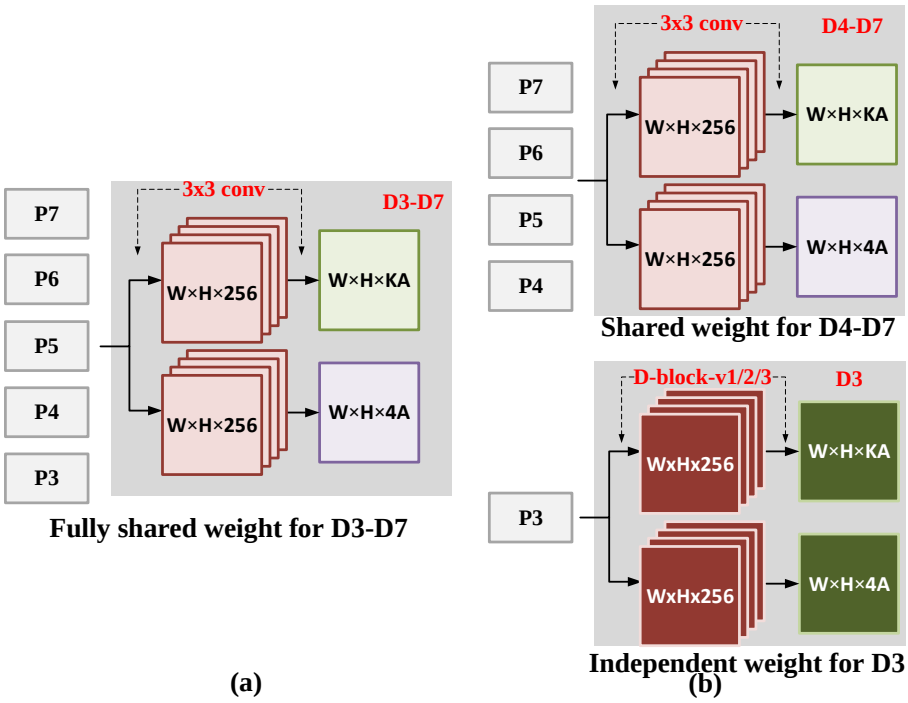


Figure 7.4: Fully and Partially Shared Weights for Detection Backend.

an original layer. The D-block-v2 alternately uses the 1×1 and 3×3 kernel. It is inspired by YOLOv1 (Redmon *et al.* (2016)), which has replaced the 3×3 kernels without introducing residual blocks. The reduction of D-block-v3 is more aggressive, which replaces all the 3×3 convolutions with 1×1 convolutions. Apparently, the light-weight block causes a certain accuracy drop as a trade-off for less computation cost. Therefore, this work proposes to add limited overheads to compensate for the accuracy drop here with a partially shared weights scheme.

7.2.2 Partially Shared Weights

As illustrated in the last paragraph, the light-weight detection blocks is to trade off lower computational complexity with limited accuracy drop. To compensate for the accuracy drop, this work proposes to replace the fully shared weight scheme in the original RetinaNet with a partial shard weight scheme. As shown in Fig. 7.2, P3-P7

are the multi-scale feature map outputs of FPN, which are then fed into detection backend D3-D7, respectively. Although D3-D7 share the weight parameters, D3-D7 have different input sizes (P3-P7), respectively, and D3-D7 are processed in serial. Fig. 7.4(a) is the original detection backend that D3-D7 fully share the weights. In Fig. 7.4(b), only D4-D7 share the weights with the original configuration, while D3 is processed by the light-weight D-block-v1/v2/v3 proposed in Fig. 7.3.

The proposed partially shared weights scheme mainly has two advantages. First, as D3 has its independent weight parameters, it can learn more tailored features for its branch, which can compensate for the accuracy drop brought by lower computational complexity. For another, this enables us not to touch the rest of the network but only solving the heaviest bottleneck block. Also, since the backbone (ResNet-50) dominates the memory consumption (as shown in Fig. 7.2), the overhead of memory consumption here (less than 1%) can be negligible.

7.3 Experimental Evaluation

7.3.1 Experimental Setup

To measure the accuracy performance, experiments are performed in Caffe2 with 4 Titan X GPUs. The experiments build upon the open-source code of RetinaNet in (Girshick *et al.* (2018)). As the original work is trained with 8 GPUs, this work scales down the base learning rate by 2x and extend the training epochs by 2x, as suggested in (Goyal *et al.* (2017)). Besides, since (Liu *et al.* (2018b)) proves the deep neural network is less easy to overfit when its computational complexity is reduced by network compression, training epochs are further extended (by the same ratio of FLOP reduction) for getting a better accuracy rate. In all the experiments, the network configuration is the same as RetinaNet-ResNet50-FPN.

To evaluate the runtime performance on FPGA-based edge devices, RetinaNet and light-weight RetinaNet are both mapped on an Intel Arria 10 GX 1150 FPGA acceleration card hosted by an Linux edge server. Intel FPGA SDK for OpenCL version 18.0 is used to compile the device code. The host code is written in C/C++ and the device code in OpenCL C language. SystolicArrayCNN – an open-source optimized OpenCL kernel is used for CNN acceleration (Dua (2019)). Each layer is run with an optimized OpenCL-based FPGA kernel for the runtime and power evaluation.

7.3.2 Performance on COCO Dataset

The COCO dataset (Lin *et al.* (2014)) is considered as the most challenging dataset for object detection. The training and testing of light-weight/original RetinaNet is conducted with 2017 COCO training dataset and COCO test-dev, respectively.

Table 7.1 shows the comparison among different light-weight blocks that proposed in Chapter 7.2.1. In this set of experiments, the light-weight block is only used in the regression branch (for the bounding box) of detection backend, which is the upper branch shown in Fig. 7.1 detection backend. The results of Table 7.1 show that the D-block-v1 – the one with the MobileNet building block has 0.8% lower mAP compared with the D-block-v3, which has the same FLOP reduction percentile. It also aligns with our analysis in Chapter 7.2 that although MobileNet is proven to a powerful light-weight classification network architecture, MobileNet building block is not guaranteed to be the best building block substitution for other computer vision tasks. Therefore, with the same scale of FLOP reduction, D-block-v3 is chosen instead of D-block-v1 in the following experiment. As the D-block-v2 performs less aggressive FLOP reduction, its mAP is only reduced by 0.1%, which is a good trade-off for a small scale FLOP reduction (15%).

Table 7.1: Comparison Between Different Light-weight Block.

Light-weight block	scale	mAP	ΔmAP\%	GFLOPs	ΔFLOPs/%
original	800	35.7	0	156	0
D-block-v1	800	34.3	1.4	135	15.4
D-block-v2	800	35.6	0.1	135	6.4
D-block-v3	800	35.1	0.6	89	15.4

Table 7.2: Configurations of Different Light-weight (LW) RetinaNet.

	Light-weight block	Detection backend	
		Classification	Bouding box
LW-RetinaNet-v1	D-block-v2	√	
LW-RetinaNet-v2	D-block-v3	√	
LW-RetinaNet-v3	D-block-v3	√	√

Table 7.3: Resource Utilization of Intel Arria 10 GX 1150 FPGA Implementation.

Resource Type	Utilization amount	Percentage
Frequency	210 MHz	-
Logic utilization	248K / 427K	58%
DSP utilization	1,184 / 1,518	78%
BRAM utilization	1,818 / 2,713	67%

The configurations for different versions of light-weight RetinaNet with D-block-v2 or D-block-v3 light-weight blocks are shown in Table 7.2. Specifically, Table 7.2 shows which light-weight block is applied to which branches of the backend in each version. The corresponding light-weight RetinaNet performance results are shown in Table 7.4. As scaling down input image size is the only method that proposed in existing work of FLOP-mAP trade-off for RetinaNet, 7.4 also cites the performance results of the original RetinaNet at different input scales from the original paper (Lin *et al.* (2017c)). For better comparison between the proposed method and input image scaling method, Fig. 7.5 visualizes the FLOPs and accuracy trade-off. Each data point in Fig. 7.5 corresponds to one row of the results in 7.4. The trending curve of light-weight RetinaNet is marked in red dot curve and that of original RetinaNet in blue dot curve. The upper-left corner is the preferred corner in the FLOP-mAP

Table 7.4: Comparison of Original RetinaNet and Proposed Light-weight RetinaNet.

	scale	mAP	AP50	AP75	APS	APM	APL	GFLOPs	ratio	runtime(s)	power efficiency (µJ/pixel)
RetinaNet	800	35.7	55	38.5	18.9	38.9	46.3	156	0	1.7	74
RetinaNet	700	35.1	54.2	37.7	18	39.3	46.4	119	1.3x	1.3	74
RetinaNet	600	34.3	53.2	36.9	16.2	37.4	47.4	88	1.8x	0.9	70
LW-RetinaNet-v1	800	35.4	54.4	38.2	18.3	38.7	46	135	1.1x	1.5	66
LW-RetinaNet-v2	800	35.1	54.3	37.7	17.9	38.4	45.7	114	1.4x	1.2	53
LW-RetinaNet-v3	800	34.6	53.1	37.3	15.7	38.7	44.6	89	1.8x	0.9	39

plane. As the red curve is constantly closer to the preferred corner, it indicates that the proposed method has a better FLOP-mAP trade-off than the conventional input image scaling method. The difference between these two methods results in a 0.1% mAP gap at the same number of FLOPs with low reduction ratio of 15%. However, when further reducing the number of FLOPs, the proposed method shows a trend of linear degradation, while the input image scaling method degrade in a more polynomial fashion. Fig. 7.5 clearly shows a divergence around 90 GFLOPs, where the input scaling method yields to 0.3% more accuracy drop than the proposed method.

The experimental setup in Chapter 7.3.1 is used to evaluate the runtime performance on FPGA-based edge devices. The resource utilization of the FPGA kernel mapped on an Intel Arria 10 GX 1150 FPGA board is shown in Table 7.3. The actual runtime shown in Table 7.4 is evaluated by accumulating the layerwise runtime. The reported power is the total board power that measured by actual testing on the FPGA board. Comparing the RetinaNet at the input scale of 600 to LW-RetinaNet-v3, LW-RetinaNet-v3 achieves an 0.3% mAP improvement over the original RetinaNet for the same runtime, and also is 1.8x more energy-efficient. One can observe that the actual runtime is approximately proportional to the FLOP count in Table 7.4, which also validates the feasibility of choosing FLOP count as the indicator to optimize the

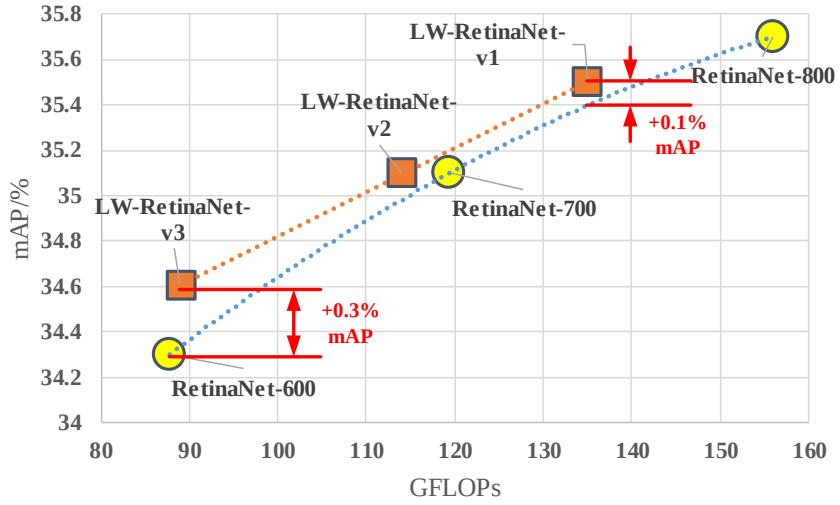


Figure 7.5: FLOPs and mAP Trade-off for Input Image Size Scaling Versus the Proposed Method.

heavy FLOP layers for speedup.

As any detection methods with FPN structure can result in an imbalanced FLOP distribution, the proposed method can be potentially applied to any such kind of detection network for a better FLOP-mAP trade-off with more energy-efficient edge inference.

7.4 Summary

This chapter presents a light-weight RetinaNet model that has a constantly better FLOP-mAP trade-off curve (linear degradation) than a naive input image scaling approach (polynomial degradation). The key is to substitute the heaviest bottleneck layer of blockwise-FLOP-imbalance RetinaNet with simplified building blocks, while keeping the rest of the network untouched. Experiment results show that, at a 1.8x FLOP reduction point, the light-weight RetinaNet achieves 0.3% mAP improvement and 1.8x more energy-efficiency on an FPGA-based edge node. The proposed method can be potentially applied to any FPN-based detection network that has im-

balanced blockwise FLOP distribution for an improved FLOP-mAP trade-off, with more energy-efficient inference at the edge.

Chapter 8

CONCLUSION AND FUTURE WORK

To enable timely IoT services on edge devices, this dissertation addresses the challenge from both hardware and algorithm perspectives. The main thrusts of this dissertation can be summarized as follows:

- **Hardware solutions**: For the FPGA-based solution, an optimized FPGA accelerator architecture is proposed for BCNNs (Chapter 3). BCNNs are ideal for efficient hardware implementations on FPGAs regardless of the size of workload. The bitwise operations in BCNNs allow for the efficient hardware mapping of convolution kernels using LUTs, which is the key to enable massive computing parallelism on an FPGA. Applying the optimal levels of architectural unfolding, parallelism, and pipelining based on the proposed throughput model is the key to maximizing the system throughput. For the ASIC-based solution, an ASIC accelerator for real-time and low-latency natural scene text interpretation (NSTI) on power constrained mobile devices is proposed (Chapter 4). The NSTI accelerator takes the cropped scene text image as input and output a salience map for pixelwise classification result. To target a real-time throughput and low latency, a B-CEDNet is adopted as the core architecture to enable massive spatial parallelism. A highly pipelined data flow control is applied to enable temporal parallelism. Moreover, all the binarized intermediate results and parameters are stored on chip to eliminate the power consumption and latency overhead of off-chip commutation. The proposed accelerator can be used in power-constrained edge devices to enable real-time augment reality applications for natural scene understanding.

- **Algorithm solutions**: For BNN compression solutions, we first propose a novel flow to explore the redundancy of BNN and remove the redundancy by bit-level sensitivity analysis and data pruning (Chapter 5). In order to build a compact BNN, one should follow these three steps. Specifically, first reconstruct a BNN with bit-sliced input and non-binary 1st layer. Then, inject randomly binarized bit slices to analyze the sensitivity level of each bit slice to the classification error rate. After that, prune P accuracy insensitive bit slices out of total N slices and rebuild a CBNN with depth shrunk by (N/P) times in each layer. Alternatively, one can adopt the proposed pruning method tailored to BNNs, which uses flipping frequency as an indicator of sensitivity to accuracy (Chapter 6). The experiments illustrate that BNNs can be further pruned by using weight flipping frequency as an indicator of sensitivity to accuracy. For an application-specific solution, we propose a light-weight RetinaNet model that has a constantly better FLOP-mAP trade-off curve (linear degradation) than a naive input image scaling approach (polynomial degradation) (Chapter 7). The key is to substitute the heaviest bottleneck layer of blockwise-FLOP-imbalance RetinaNet with simplified building blocks, while keeping the rest of the network untouched. The proposed method can be potentially applied to any FPN-based detection network that has imbalanced blockwise FLOP distribution for an improved FLOP-mAP trade-off, with more energy-efficient inference at the edge.

The proposed solutions also open up several new directions for future work. The idea of substituting the heaviest bottleneck layer with simplified building blocks has been proven to be feasible for a better FLOP-mAP tradeoff in blockwise-FLOP-imbalance RetinaNet. If this method can be generalized to a large portion of networks, it will be a useful criteria in both manually tuning the network hyperparameters or automated neural network architecture search. Moreover, it is also worthy exploring

how to formulate a metric to measure the redundancy of each layer. A layer is not just an independent layer. It is a part of a network. Its contiguous layers or even the entire network topology can effect the redundancy level of a certain layer. All these factors are worthy exploring in the future work.

REFERENCES

Andri, R., L. Cavigelli, D. Rossi and L. Benini, "Yodann: An architecture for ultralow power binary-weight cnn acceleration", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**, 1, 48–60 (2018).

Badrinarayanan, V., A. Kendall and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", IEEE transactions on pattern analysis and machine intelligence **39**, 12, 2481–2495 (2017).

Biookaghazadeh, S., M. Zhao and F. Ren, "Are fpga suitable for edge computing?", in "{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)", (2018).

Bissacco, A., M. Cummins, Y. Netzer and H. Neven, "Photoocr: Reading text in uncontrolled conditions", in "Computer Vision (ICCV), 2013 IEEE International Conference on", pp. 785–792 (IEEE, 2013).

Bong, K., S. Choi, C. Kim, S. Kang, Y. Kim and H.-J. Yoo, "14.6 a 0.62 mw ultra-low-power convolutional-neural-network face-recognition processor and a cis integrated with always-on haar-like face detector", in "Solid-State Circuits Conference (ISSCC), 2017 IEEE International", pp. 248–249 (IEEE, 2017).

Carreira-Perpinán, M. A. and Y. Idelbayev, ""learning-compression" algorithms for neural net pruning", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 8532–8541 (2018).

Chen, Y.-H., T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks", IEEE Journal of Solid-State Circuits **52**, 1, 127–138 (2017).

Cheng, Y., F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections", in "Proceedings of the IEEE International Conference on Computer Vision", pp. 2857–2865 (2015).

Computing, F., "Fog computing and the internet of things: Extend the cloud to where the things are", (2016).

Courbariaux, M., Y. Bengio and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations", in "Advances in neural information processing systems", pp. 3123–3131 (2015).

De Campos, T. E., B. R. Babu, M. Varma *et al.*, "Character recognition in natural images.", VISAPP (2) **7** (2009).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", in "Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on", pp. 248–255 (IEEE, 2009).

Desoli, G., N. Chawla, T. Boesch, S.-p. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh *et al.*, "14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems", in "Solid-State Circuits Conference (ISSCC), 2017 IEEE International", pp. 238–239 (IEEE, 2017).

Dua, A., *Hardware Acceleration of Video Analytics on FPGA Using OpenCL*, Ph.D. thesis, Arizona State University (2019).

Farabet, C., B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision.", in "CVPR Workshops", pp. 109–116 (2011).

Girshick, R., I. Radosavovic, G. Gkioxari *et al.*, "Detectron", `https://github.com/facebookresearch/detectron` (2018).

Goodfellow, I., Y. Bengio, A. Courville and Y. Bengio, *Deep learning*, vol. 1 (MIT press Cambridge, 2016).

Goyal, P., P. Dollár, R. Girshick *et al.*, "Accurate, large minibatch sgd: Training imagenet in 1 hour", arXiv preprint arXiv:1706.02677 (2017).

Guo, P., H. Ma, R. Chen, P. Li, S. Xie and D. Wang, "Fbna: A fully binarized neural network accelerator", in "2018 28th International Conference on Field Programmable Logic and Applications (FPL)", pp. 51–513 (IEEE, 2018).

Han, S., H. Mao and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", International Conference on Learning Representations (2016).

Han, S., J. Pool, J. Tran and W. Dally, "Learning both weights and connections for efficient neural network", in "Advances in neural information processing systems", pp. 1135–1143 (2015).

He, K., G. Gkioxari, P. Dollár *et al.*, "Mask r-cnn", in "Proceedings of the IEEE international conference on computer vision", pp. 2961–2969 (2017).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 770–778 (2016).

Howard, A. G., M. Zhu, B. Chen *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv preprint arXiv:1704.04861 (2017).

Huang, J., V. Rathod, C. Sun *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 7310–7311 (2017).

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized neural networks", in "Advances in neural information processing systems", pp. 4107–4115 (2016).

Inference, G.-B. D. L., "A performance and power analysis", Whitepaper, November (2015).

Ioffe, S. and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", arXiv preprint arXiv:1502.03167 (2015).

Jacob, B., S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 2704–2713 (2018).

Jaderberg, M., A. Vedaldi and A. Zisserman, "Deep features for text spotting", in "European conference on computer vision", pp. 512–528 (Springer, 2014).

Jouppi, N. P., C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit", in "2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)", pp. 1–12 (IEEE, 2017).

Krizhevsky, A., G. Hinton *et al.*, "Learning multiple layers of features from tiny images", Tech. rep., Citeseer (2009).

Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in "Advances in neural information processing systems", pp. 1097–1105 (2012).

LeCun, Y., Y. Bengio and G. Hinton, "Deep learning", nature **521**, 7553, 436 (2015).

LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE **86**, 11, 2278–2324 (1998).

Li, Y., Z. Liu, K. Xu, H. Yu and F. Ren, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks", in "Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 290–291 (ACM, 2017).

Li, Y., Z. Liu, K. Xu, H. Yu and F. Ren, "A gpu-outperforming fpga accelerator architecture for binary convolutional neural networks", ACM Journal on Emerging Technologies in Computing Systems (JETC) **14**, 2, 18 (2018).

Lin, J.-H., T. Xing, R. Zhao, Z. Zhang, M. B. Srivastava, Z. Tu and R. K. Gupta, "Binarized convolutional neural networks with separable filters for efficient hardware acceleration.", in "CVPR Workshops", pp. 344–352 (2017a).

Lin, M., Q. Chen and S. Yan, "Network in network", arXiv preprint arXiv:1312.4400 (2013).

Lin, T.-Y., P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature pyramid networks for object detection", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 2117–2125 (2017b).

Lin, T.-Y., P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection", in "Proceedings of the IEEE international conference on computer vision", pp. 2980–2988 (2017c).

Lin, T.-Y., M. Maire, S. Belongie *et al.*, "Microsoft coco: Common objects in context", in "European conference on computer vision", pp. 740–755 (Springer, 2014).

Lin, X., C. Zhao and W. Pan, "Towards accurate binary convolutional neural network", in "Advances in Neural Information Processing Systems", pp. 345–353 (2017d).

Liu, W., D. Anguelov, D. Erhan *et al.*, "Ssd: Single shot multibox detector", in "European conference on computer vision", pp. 21–37 (Springer, 2016).

Liu, Z., Y. Li, F. Ren, W. L. Goh and H. Yu, "Squeezedtext: A real-time scene text recognition by binary convolutional encoder-decoder network", Thirty-Second AAAI Conference on Artificial Intelligence (2018a).

Liu, Z., M. Sun, Y. Zhou *et al.*, "Rethinking the value of network pruning", arXiv preprint arXiv:1810.05270 (2018b).

Luo, J.-H., J. Wu and W. Lin, "Thinet: A filter level pruning method for deep neural network compression", in "Proceedings of the IEEE international conference on computer vision", pp. 5058–5066 (2017).

Molchanov, P., A. Mallya, S. Tyree, I. Frosio and J. Kautz, "Importance estimation for neural network pruning", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 11264–11272 (2019).

Moons, B. and M. Verhelst, "A 0.3–2.6 tops/w precision-scalable processor for real-time large-scale convnets", in "VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on", pp. 1–2 (IEEE, 2016).

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", in "NIPS workshop on deep learning and unsupervised feature learning", vol. 2011, p. 5 (2011).

Ouyang, J., S. Lin, W. Qi, Y. Wang, B. Yu and S. Jiang, "Sda: Software-defined accelerator for large-scale dnn systems", in "2014 IEEE Hot Chips 26 Symposium (HCS)", pp. 1–23 (IEEE, 2014).

Pham, P.-H., D. Jelaca, C. Farabet, B. Martini, Y. LeCun and E. Culurciello, "Neuflow: Dataflow vision processing system-on-a-chip", in "Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on", pp. 1044–1047 (IEEE, 2012).

Qiu, J., J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network", in "Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 26–35 (ACM, 2016).

Rahmani, M., A. Ghanbari and M. M. Ettefagh, "Hybrid neural network fraction integral terminal sliding mode control of an inchworm robot manipulator", Mechanical Systems and Signal Processing **80**, 117–136 (2016).

Rahmani, M., A. Ghanbari and M. M. Ettefagh, "A novel adaptive neural network integral sliding-mode control of a biped robot using bat algorithm", Journal of Vibration and Control **24**, 10, 2045–2060 (2018).

Rastegari, M., V. Ordonez, J. Redmon and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", in "European Conference on Computer Vision", pp. 525–542 (Springer, 2016).

Redmon, J., S. Divvala, R. Girshick *et al.*, "You only look once: Unified, real-time object detection", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 779–788 (2016).

Redmon, J. and A. Farhadi, "Yolo9000: better, faster, stronger", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 7263–7271 (2017).

Redmon, J. and A. Farhadi, "Yolov3: An incremental improvement", arXiv preprint arXiv:1804.02767 (2018).

Ren, S., K. He, R. Girshick *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks", in "Advances in neural information processing systems", pp. 91–99 (2015).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556 (2014).

Stallkamp, J., M. Schlipsing, J. Salmen and C. Igel, "The german traffic sign recognition benchmark: a multi-class classification competition", in "Neural Networks

(IJCNN), The 2011 International Joint Conference on", pp. 1453–1460 (IEEE, 2011).

Stillmaker, A. and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm", Integration **58**, 74–81 (2017).

Suda, N., V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks", in "Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 16–25 (ACM, 2016).

Tang, W., G. Hua and L. Wang, "How to train a compact binary neural network with high accuracy?", in "AAAI", pp. 2625–2631 (2017).

Tu, F., S. Yin, P. Ouyang, S. Tang, L. Liu and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **25**, 8, 2220–2233 (2017).

Wang, T., D. J. Wu, A. Coates and A. Y. Ng, "End-to-end text recognition with convolutional neural networks", in "Pattern Recognition (ICPR), 2012 21st International Conference on", pp. 3304–3308 (IEEE, 2012).

Yu, J., "Xnor-net", `https://github.com/jiecaoyu/XNOR-Net-PyTorch` (2019).

Yu, R., A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 9194–9203 (2018).

Zhang, C., P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks", in "Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 161–170 (ACM, 2015).

Zhang, D., J. Yang, D. Ye and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks", in "Proceedings of the European Conference on Computer Vision (ECCV)", pp. 365–382 (2018a).

Zhang, T., K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad and Y. Wang, "Adam-admm: A unified, systematic framework of structured weight pruning for dnns", arXiv preprint arXiv:1807.11091 (2018b).

Zhang, X., S. Das, O. Neopane and K. Kreutz-Delgado, "A design methodology for efficient implementation of deconvolutional neural networks on an fpga", arXiv preprint arXiv:1705.02583 (2017).

Zhao, R., W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas", in "Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 15–24 (ACM, 2017).

Zhu, C., S. Han, H. Mao and W. J. Dally, "Trained ternary quantization", arXiv preprint arXiv:1612.01064 (2016).