Neurocomputing 398 (2020) 45–54

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Build a compact binary neural network through bit-level sensitivity and data pruning

Yixing Li[a,1,*], Shuai Zhang[b], Xichuan Zhou[b], Fengbo Ren[a]

[a] School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA
[b] School of Microelectronics and Communication Engineering, Chongqing University, Chongqing, China

ARTICLE INFO

Article history:
Received 28 February 2019
Revised 17 December 2019
Accepted 3 February 2020
Available online 11 February 2020

Communicated by Dr. Wen Wujie

Keywords:
Binary neural networks
Deep neural networks
Deep learning
Neural network compression

ABSTRACT

Due to the high computational complexity and memory storage requirement, it is hard to directly deploy a full-precision convolutional neural network (CNN) on embedded devices. The hardware-friendly designs are needed for resource-limited and energy-constrained embedded devices. Emerging solutions are adopted for the neural network compression, e.g., binary/ternary weight network, pruned network and quantized network. Among them, binary neural network (BNN) is believed to be the most hardware-friendly framework due to its small network size and low computational complexity. No existing work has further shrunk the size of BNN. In this work, we explore the redundancy in BNN and build a compact BNN (CBNN) based on the bit-level sensitivity analysis and bit-level data pruning. The input data is converted to a high dimensional bit-sliced format. In the post-training stage, we analyze the impact of different bit slices to the accuracy. By pruning the redundant input bit slices and shrinking the network size, we are able to build a more compact BNN. Our result shows that we can further scale down the network size of the BNN up to 3.9x with no more than 1% accuracy drop. The actual runtime can be reduced up to 2x and 9.9x compared with the baseline BNN and its full-precision counterpart, respectively.

1. Introduction

Vision-based applications can be found in many embedded devices for classification, recognition, detection and tracking tasks [1,2]. Specifically, convolutional neural network (CNN) has become the core architecture for those vision-based tasks [3]. Since it can outperform conventional feature selection-based algorithm in terms of accuracy, it becomes more and more popular. Advanced driver-assistance system (ADAS) can either use CNNs for guiding autonomous driving or alerting the driver of predicted risk [1]. It is obvious that ADAS depends on a low-latency system to get a timely reaction. Artificial intelligence (AI) applications also explode in smartphones, such as automatically tagging the photos, face detection and so on [2,4]. Apple has announced that the Apple Neural Engine on iPhone is aiming at partially moving their AI processing module on device [5]. If the users' requests are processed through sending them to the data center, there will be much overhead of the latency and power consumption caused by the commutation. As such, on-device AI processing is the future trend to balance power efficiency and latency. However, CNNs are known to have high computational complexity, which makes it hard to directly deploy on embedded devices. Therefore, compressed CNNs are in demand.

In the early stage, research work of hardware-friendly CNNs have focused on reducing the numerical precision down to 8–16 bits in the post-training stage [6], which either has a limited reduction or suffers from severer accuracy drop. Lately, in-training techniques have been brought up, achieving much higher compression ratio. BinaryConnect, BinaryNet, TernaryNet, XNOR-Net and LQ-Net [7–11] have pushed to reduce the weight to binary or ternary (−1, 0, +1) values. Network pruning [12] reduces the network size (the memory size for all the parameters) by means of reducing the number of connections. Regarding the network size, pruned network and TernaryNet can achieve 13x and 16x reduction [9,12], respectively. While BinaryConnect, BinaryNet, XNOR-Net and LQ-Net can achieve up to 32x reduction. In terms of computational complexity, BinaryNet and XNOR-Net both have binarized weights and activations, which can simply replace convolution operation with bitwise XNOR (Exclusive-NOR) and bit count operation. XNOR-Net has additional scaling factor filters in each layer, which brings overhead to both memory and computation cost. Overall, BinaryNet is recognized as the most efficient solution for hardware deployment by hardware community when

* Corresponding author.
E-mail addresses: yixingli@asu.edu (Y. Li), zs@cqu.edu.cn (S. Zhang), zxc@cqu.edu.cn (X. Zhou), renfengbo@asu.edu (F. Ren).

[1] The source code is available at https://github.com/PSCLab-ASU/C-BNN

https://doi.org/10.1016/j.neucom.2020.02.012
0925-2312/

considering its small network size and low computational complexity [13]. The computational complexity of LQ-Net [11], which can be configured to have 1-bit weights and 2-bit activations, will rank the 2nd right after BinaryNet. In the rest of paper, we use the term, Binary Neural Networks (BNNs) specifically to represent the top two compact binary weight neural networks – BinaryNet and LQ-Net. Although the study is possible to be generalized to any binarized weight neural nets (NNs), here we just constrain our study to the most two compact form of binary weight neural networks to show the impact to such compact NNs.

Although, pruning is capable to work with reduced-precision CNNs (typically 4–8 bits) to argument the computational resource savings, it is not clear how much improvement it can get with BNN. Pruning methods can be categorized as magnitude-based and optimization-based pruning. For magnitude-based pruning, 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 cannot be applied with magnitude analysis. For optimization-based pruning, the pruned network can be resulted in a structured or non-structured way [14]. For the non-structured case, additional "flag bits" are needed for marking each prunable weight, which may even increase the memory consumption and is not beneficial to the inference speed. The only possible way which can improve runtime performance is to apply optimization-based structured pruning [15,16] on BNN, but no existing work has done any related study. In summary, how much redundancy the BNN still has is still unknown, and no existing solution has been proved to work effectively on the BNN.

In order to explore how much redundancy the BNN still has, and how much one can further compress a BNN, a new solution tailored for BNN compression is needed. For a data center based application which run large models, BNN may not be the best option. On the contrary, for on-device inference on resource constrained embedded system, further reducing the memory footprint and network size are critical to efficient computing. If there is a smart gate lock for a company's building, it probably needs a really large model since the dataset is large. However, if it is a smart door lock for a single house or apartment, a smaller model will be good enough for such a small dataset. Also, in this case, the BNN will make the smart lock to be more energy-efficient with speed enhancement.

In the previous work, [17] demonstrates the redundancy in the first layer of BNN and [18] uses approximated binary filters. The former one reduces the connection only in the first layer and the latter one encodes the original binary filters into approximated ones without any connection reduction. This work is the first one that explores and proves that there is still connection redundancy throughout the entire BNN. The proposed flow to reduce the network size is triggered by conversion and analysis of input data rather than the network body, which is rarely seen in previous work. A novel flow is proposed to prune out the redundant input bit slices and rebuild a compact BNN through bit-level sensitivity analysis.

Actually, the proposed method shares very similar idea with simplifying combinational logic circuits. In the logic circuits, all input/output data and intermediate results are all binary (0/1) values. If flip one input node value (change from 1 to 0 or 0 to 1) but none of the outputs change, it means that this input node is useless for building this system. In other words, you can remove this redundant input node and the function of the system will not change. By removing this useless input node, it helps to simplify combinational logic circuits design. Similarly, the binary neural network can be seen as the body of combinational logic circuits. If we flip the binary inputs of the binary neural network and its output (accuracy) doesn't change, we can infer these inputs are redundant, which can be removed. Accordingly, we can shrink the network size (reduce the number of parameters) since the function we would like to approximate is simplified. Experiment results show that the compression ratio of the network size is achieving up to 3.9x with no more than 1% accuracy drop.

The rest of the paper is organized as follows. Section 2 discusses the related work for network compression and explains why BNN is a more superior solution to be deployed on the hardware. In addition, Section 2 also explains why existing neural network compression methods cannot be applied on BNN. Section 3 demonstrates the experiments to validate the hypothesis that BNN has redundancy and proposes a novel flow to build a compact BNN. Experiment results and discussion are shown in Section 4. Section 5 concludes the paper.

2. Related work

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. In this section, we first discuss and evaluate the related work for network compression by emphasizing both factors. We also present a simple benchmark study to help the readers better understand the computational complexity in terms of hardware resource utilization of the existing work. It can reveal why BNN is a more superior solution to be deployed on the hardware.

2.1. Reduced-precision methods

BinaryConnect [7] 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 [8] 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 [10] 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 [9] 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 [11] 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.

2.2. Reduced-connection methods

Network pruning [12] 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 [6]. 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. We will explain in below why they both are not compatible with further compressing BNN.

For magnitude-based pruning [12], 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 [14], 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 [15,16] 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.

2.3. Other methods

Singular Value Decomposition (SVD) is one method that has been applied to BNN to compress its weight matrices [18]. 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 [18].

2.4. Comparison

We implement a $W_{(10,10)} \times A_{(10,10)}$ matrix multiplication 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, we don'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 [6], we set the precision of any full precision weights or activations to be 16 bits. For the pruned network, we set 84% of the elements

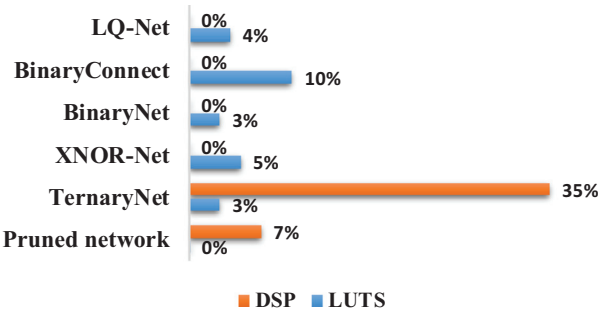


Fig. 1. Resource consumption of $W^{(10,10)} \times A^{(10,10)}$ multiplication on a Xilinx Virtex-7 FPGA for different architecture.

in W of the pruned network as zeros for a fair comparison. (Since pruned network can get up to 13x reduction [12] while BNN can get 32x, the size of the pruned network is $32/13 = 2.5$x larger. With 16-bit weights, the total number of non-zero weights of the pruned network is $2.5/16 = 16\%$ of that of the binarized weight cases.) For LQ-Net [11], we only refer to its most compact configuration in this paper, which has 1-bit weights and 2-bit activations. As shown in Fig. 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, we have explained in Section 2.2. For precision reduction, BNN has already reached the lower bound.

Since CNNs are believed to have huge redundancy, we hypothesize that the BNN also has redundancy and it is able to get a more compact BNN. To our best knowledge, there is only one related work pruned the first layer of a BNN with the observation of barely any accuracy drop [17]. Since they only compress the first layer, the impact on the entire network is fairly limited. On the contrary, we have analyzed and conducted the experiments to prove reducing the input precision is a valid method to trigger the compression of the entire BNN.

We are the first to explore the BNN redundancy across the entire network by the bit-level analysis of the input data. We will validate our hypothesis step by step in the next section.

In the following paragraphs, BNN is referring to non-compressed binarized CNN, which is our baseline model. The reconstructed BNN and CBNN is referring to the reconstructed model we used for sensitivity analysis and the final compact BNN with shrunk network size, respectively.

3. Build a compact BNN

First, we need to reconstruct and train a new model for sensitivity analysis. Section 3.1 demonstrates the model reconstruction of BNN and shows the redundancy exists in BNN through statistical analysis of the non-binary first layer. Then, with the reconstructed BNN, Section 3.2 will further prove the redundancy exists throughout the entire BNN and decide the prunable bit slices through bit-level sensitivity analysis in post-training stage. Finally, Section 3.3 presents the guide to rebuilding a more compact BNN (CBNN) that triggered by input data pruning.

3.1. BNN reconstruction

In this section, we first illustrates reformatting the input and modifying the first layer for the BNN reconstruction in Section 3.1.1. Then we shows the redundancy exists in BNN through

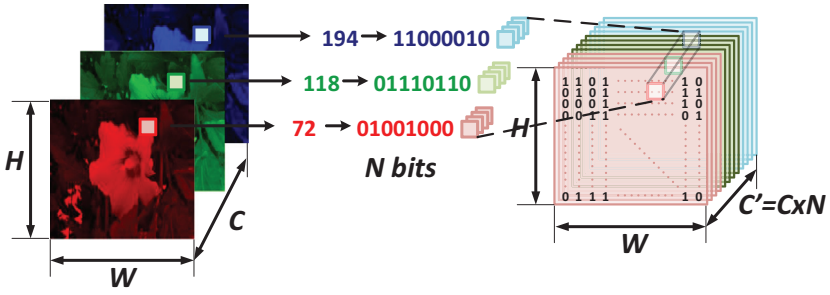


Fig. 2. Conversion from fixed-point input to bit-sliced binary input.

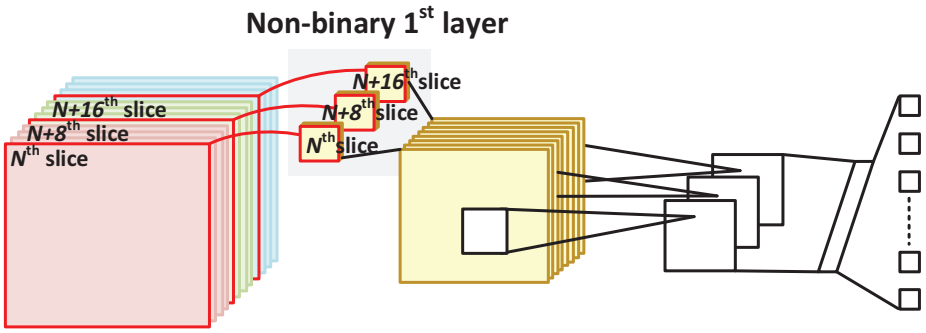


Fig. 3. Corresponding relationship between input bit slices and non-binary first layer.

statistical analysis of the non-binary first layer in Section 3.1.2. In Section 3.1.3, the training method is presented.

3.1.1. 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. 2. 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 format is defined as the *int2b* function.

$$D_{(W,H,C')}{}^{b} = int2b(D_{(W,H,C)}, N), \tag{1}$$

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.

3.1.2. 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, we assign the first layer with 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. In the 2-D convolutions, the N^{th} slices of first-layer weights are only multiplied with N^{th} input bit slices. In an extreme case, if all the weights of N^{th} slices are 0s, no information of N^{th} input bit slices are propagated into the 2^{nd} layer. When the constraint is relaxed a little bit, if the N^{th} slices of weights associated with N^{th} input bit slices are all closed to 0s, the corresponding N^{th} input data bit slices that multiplied by those small values will have trivial contribution to the computation in the rest of layers. It can also be interpreted as these input features are filtered out. As shown in Fig. 3, input bit slices and first layer weight slices have one-to-one correlations. Thus, we group the weight associated with the N^{th} input bits separately for statistical analysis. In Fig. 4, 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 4th input bit slice, the weight magnitude spreads out in a wider range. Therefore, we can hypothesize 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 [8], 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 [6]. 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, we reconstruct the BNN model and refer it as the reconstructed BNN. In our experiments, we simply take the input bit-slices as floating-point 0s and 1s. Thus, the computations in the 1st layer are standard floating-point operations (multiplications and additions). Although the computational complexity is the same, the new structure helps to reduce the redundancy in BNN, which will be elaborated in the following sections.

3.1.3. Binary constrained training

For BinaryNet-based experiment, we adopt the training method proposed by Hubara et al. [8]. The objective function is shown in Eq. 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

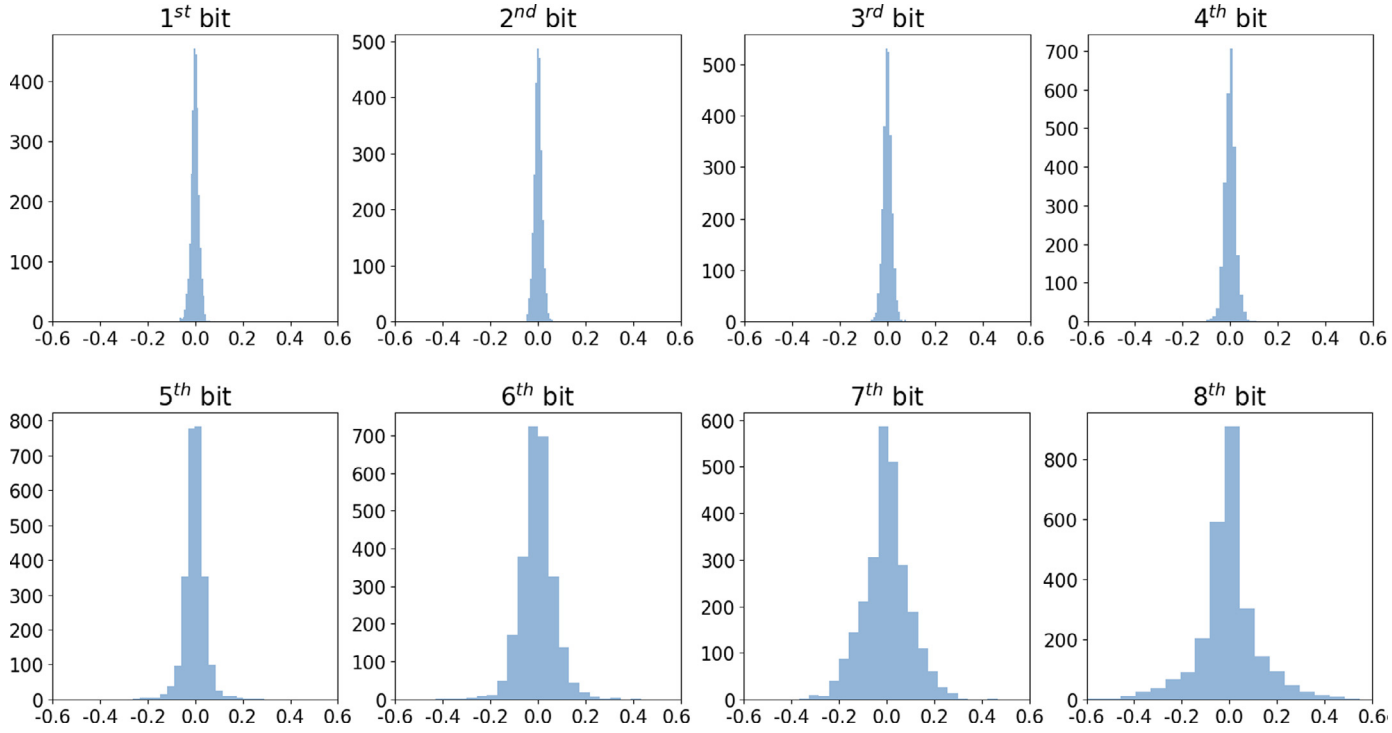


Fig. 4. Histogram of distributions of weight magnitude associated with different input bits.

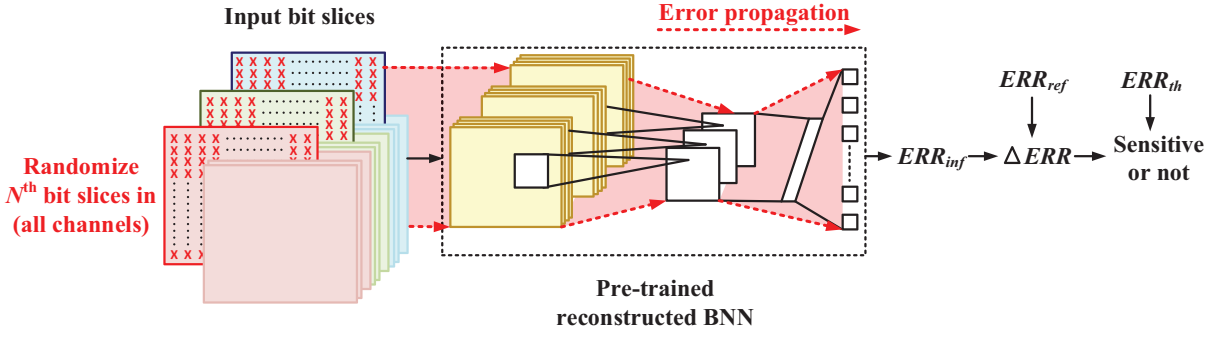


Fig. 5. Sensitivity analysis of the reconstructed BNN with distorted input.

propagation, and. the binarized weights $W_l^b = clip(W_l)$ [8] are used in the forward propagation. As Tang et al. propose in [19], 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. For LQ-Net-based experiment, we use exactly the same training method proposed in the original paper [11].

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

3.2. Sensitivity analysis

We use the training method in Section 3.1 to train a reconstructed BNN model with the bit-sliced input and non-binary first layer. In the post-training stage, we demonstrate the method to show the redundancy throughout the entire network and evaluate the sensitivity of the bit-sliced input to the accuracy performance.

As shown in Fig. 5, 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 why we use binary random bit slices other than pruning is that, pruning will reduce the size of the network. We want 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. 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, we can also analyze the sensitivity of a stack of slices by using the same method. Then we can find a collection of insensitive bit slices 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.

3.3. Rebuild a compact network

In the most popular CNN architectures, such as AlexNet [20], VGG [21] and ResNet [22], 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 we haven't explored how to build an accurate model to optimize the network compression ratio, we emphasize the entire flow (presented in Section 3) that proves and reduces the redundancy of the entire BNN, and enables speedup in the inference stage with the CBNN. In Section 4, we will present and discuss the performance results corresponding to each subsection in Section 3.

4. Result and discussion

We will first walk through the flow presented in Section 3 with experimental results on the CIFAR-10 classification task in Section 4.1. Section 4.2 will present additional results on SNVH, Chars74K, GTSRB and ImageNet datasets.

For the experiment setup, we build relative smaller models (AlexNet-scale) 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 model, here we test relative larger models (ResNet-scale) 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 [23]. 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) [24]. 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 [25]. 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) [26]. This dataset includes 43-class traffic signs. We resize the traffic sign images to 32 × 32. It has 39,209 training data and 12,630 testing data.

ImageNet [27]. ImageNet is a large scale dataset which has more than 14 million hand-annotated images. Here, we use its subset – 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 450 × 450 pixels.

4.1. Experiment on CIFAR-10

Subsections of 4.1 show the experimental results corresponding to the methodology in Section 3.1–3.3, respectively.

4.1.1. BNN reconstruction

Following the input data conversion method in Section 3.1, 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 $\text{CIFAR}^b_{(32,32,24)}$.

We have plotted an image in CIFAR-10 dataset with different bit-level distortion shown in Fig. 6. This image belongs to the "horse" category. In Fig. 6, 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. 6(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 7th bit slice gets distorted. If we further distort CIFAR^b in multiple bit slices from the 1^{st} to N^{th} bit slices, the corresponding images are shown in Fig. 6(b) and (c). The images in Fig. 6(c) are different from Fig. 6(b) that they don't maintain 8-bit precision. Instead, we directly prune the 1^{st} to N^{th} bit slices of the images in Fig. 6(c). From visualization, in both Fig. 6(b) and (c), the turning points is at 5^{th} bit.

Distorted images in Fig. 6(a) and (b) are used for sensitivity analysis in reconstructed BNN (Recon. BNN). Pruned images in Fig. 6(c) are used for training CBNN.

As illustrated in Section 3.1, the proposed structure of the reconstructed BNN is different from the original BNN in both input format and the first layer. Table 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 [8], with full precision input and a binarized 1^{st} layer. Here we define a CNN with bit-sliced input, binarized weights and activations as FBNN. FBNN has bit slices input but BNN does not. By training with the method in Section 3.1, 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 [19]. By introducing bit slices input and non-binary 1^{st} layer to reconstruct the BNN (as we proposed in Section 3.1), the accuracy drop can be compensated as shown in Table 1. We can 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.

4.1.2. Sensitivity analysis of the reconstructed BNN

With a pre-trained reconstructed BNN presented in the last section, now we can do bit-level sensitivity analysis as stated in Section 3.2.

First, we analyze the sensitivity of a single bit slice. The results are shown in Table 2. The data shows in Table 2 is the average over 10 trials. In addition to the reconstructed BNN, we also evaluate the bit-level sensitivity of the input with its full-precision counterpart, which is denoted as FNN. With FNN, we intend to show that the data itself has redundancy, which can be reflected in both binary domain or fixed-point domain with the same pattern. We take the architecture in the first row 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. We define the turning point of error in sensitivity analysis as the point where ΔERR flips the sign or increases abruptly. The turning point here is the 5^{th} bit.

Second, we analyze the sensitivity of bit slices stacks. Each stack contains 1st to Nth bit slices in each color channel. The results are shown in Table 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.

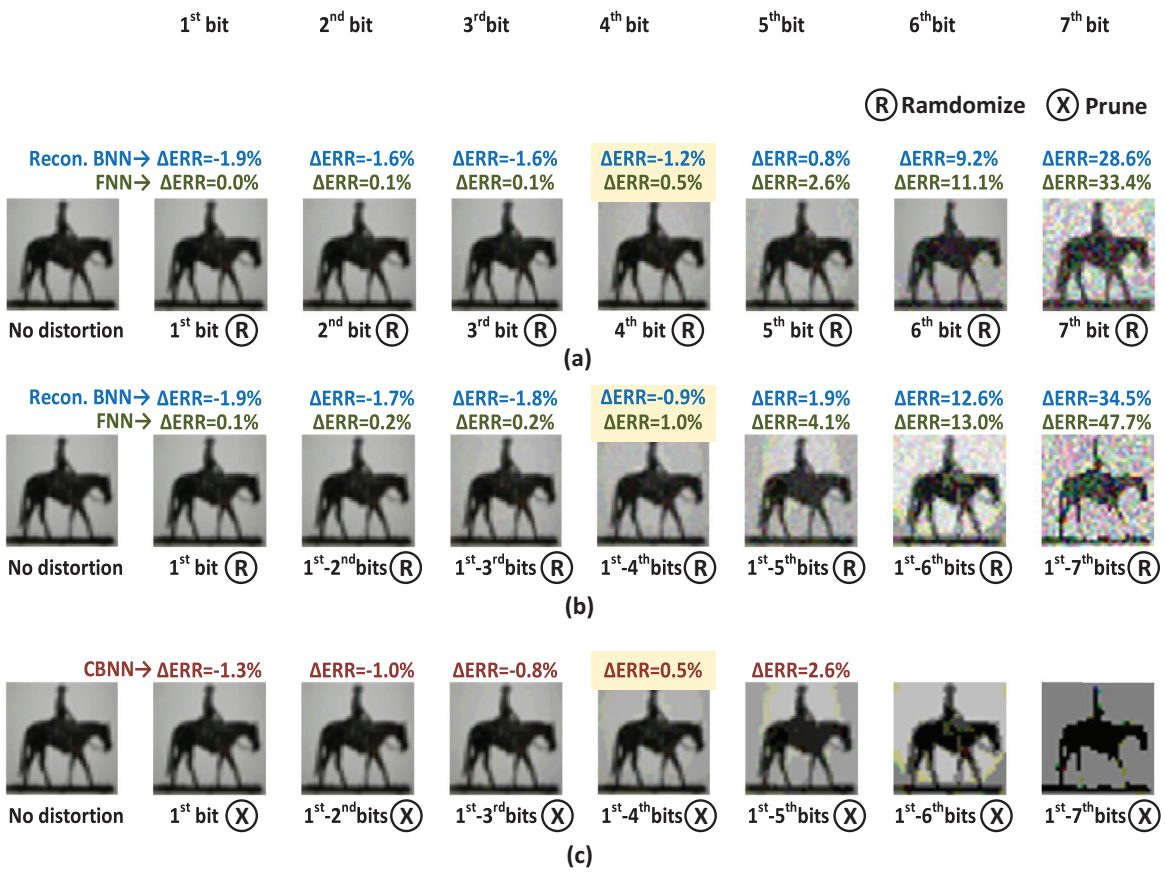


Fig. 6. Visualization of a horse image in CIFAR-10 with different bit-level distortion in spatial domain and frequency domain.

Table 1
Performance comparison with different input format and 1st 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

Table 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

Table 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

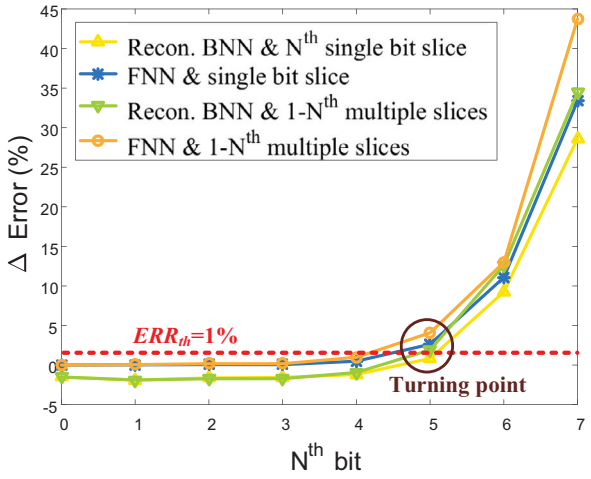


Fig. 7. Error rate of randomizing one or multiple bit slices in sensitivity analysis.

Table 4
Performance of CBNNs on CIFAR-10.

Arch.	1-N^{th} bits	ERR%	ΔERR%	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

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 we 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. Therefore, these bits are useless in the training stage. This validates the hypothesis that the BNN still has redundancy. In Fig. 7, 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. 7) align well. In order to make the entire process be automatic, we can simple set an error-tolerant threshold ERR_{th} to determine how many bits are prunable. Here, ERR_{th} is set to 1%. We can conclude 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.

4.1.3. Rebuild a compact BNN (CBNN)

Since 4 out of 8 bit slices are prunable, we can rebuild a compact BNN with the depth of each layer shrunk by half. The performance of CBNN is shown in Table 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, we use 16 bits 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 [6]. We also show the alternatives of pruning 1-N^{th} ($N = 1, 2, \ldots, 5$) bit slices and shrink the layerwise depth by 1/8 to 5/8. 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 1st layer is larger than that of BNN, we cannot achieve the ideal network size compress ratio (4x) 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.

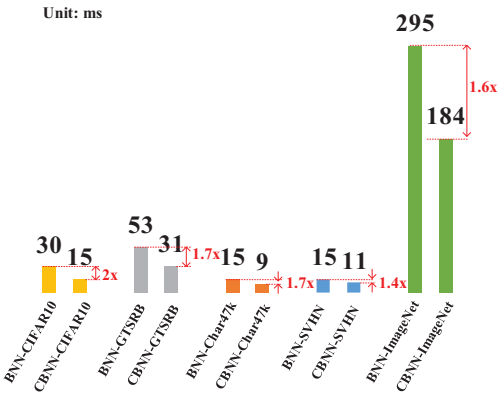


Fig. 8. Runtime comparison of different network compression technique.

4.2. Experiment on SVHN/Chars74K/GTSRB/ImageNet datasets

In this section, we will skip the sensitivity analysis and just show the result comparison between the baseline and the final CBNNs we get in the same procedure.

For SVHN and Char74K datasets, we use a baseline architecture that has half of the depth in each layer as the one for CIFAR-10. For GTSRB, we use a baseline architecture that 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, we use the same ResNet-18 architecture as Zhang et al.'s work.

In Table 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 1-3^{rd} bits and pruning 1-4^{th} 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.

4.3. Runtime evaluation

We evaluate the actual runtime performance of CBNNs by Nvidia GPU Titan X. The batch size is fixed as 128 in all the experiments. We use the same XNOR-based GPU kernel as [8] for CBNN implementation. The computational time is calculated by averaging over 10 runs.

Fig. 8 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 Tables 4 and 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 [28], combining pruning, quantization and Huffman coding technique, an FNN can achieve up to 4x speedup [8]. 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.

Table 5
Performance results of CBNNs on SVHN, Chars47k, GTSRB and ImageNet datasets.

Arch.	Dataset	1-N^{th} bits	ERR%	ΔERR%	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	3.0	1x
		1	37.1	0.0	115	1.3x	2.6	1.3x
		2	37.2	0.1	113	1.8x	2.3	1.8x
		3	**38.5**	**1.5**	**94**	**2.5x**	**1.9**	**2.6x**

5. Conclusion

In this paper, we propose a novel flow 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.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

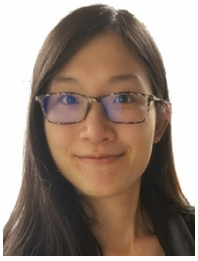
Acknowledgment

The work by Arizona State University is supported by an NSF grant (IIS/CPS-1652038) and an unrestricted gift (CG#1319167) from Cisco Research Center. The work by Chongqing University is funded by Science and Technology on Analog Integrated Circuit Laboratory (No. 6142802WD201807). The GPUs used for this research were donated by the NVIDIA Corporation.

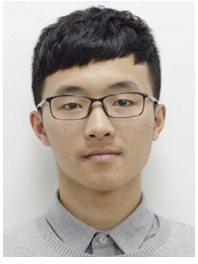
References

[1] Y. Tian, P. Luo, X. Wang, X. Tang, Pedestrian detection aided by deep learning semantic tasks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5079–5087.
[2] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S.Z. Li, T. Hospedales, When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2015, pp. 142–150.
[3] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[4] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: a unified embedding for face recognition and clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 815–823.
[5] D. Lumb, Apple 'neural engine' chip could power ai on iphones, 2017. https://www.engadget.com/2017/05/26/apple-neural-engine-chip-could-power-ai-on-iphones/.
[6] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, 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, ACM, 2016, pp. 16–25.
[7] M. Courbariaux, Y. Bengio, J.P. David, Binaryconnect: training deep neural networks with binary weights during propagations, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 3123–3131.
[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in: Proceedings of the Advances in Neural Information Processing Systems, 2016, pp. 4107–4115.
[9] C. Zhu, S. Han, H. Mao, W.J. Dally, Trained ternary quantization, in: 5th International Conference on Learning Representations, (ICLR), 2017.
[10] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-NET: imagenet classification using binary convolutional neural networks, in: Proceedings of the European Conference on Computer Vision, Springer, 2016, pp. 525–542.
[11] D. Zhang, J. Yang, D. Ye, G. Hua, LQ-NETS: Learned Quantization for Highly Accurate and Compact Deep Neural Networks, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 365–382.
[12] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.
[13] Y. Li, Z. Liu, K. Xu, H. Yu, F. Ren, A 7.663-Tops 8.2-W energy-efficient FPGA accelerator for binary convolutional neural networks, in: Proceedings of the FPGA, 2017, pp. 290–291.
[14] M.A. Carreira-Perpinan, Y. Idelbayev, "Learning-compression" algorithms for neural net pruning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8532–8541.
[15] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, Y. Wang, StructADMM: A Systematic, High-Efficiency Framework of Structured Weight Pruning for DNNs, 2018. arXiv:1807.11091.
[16] J.-H. Luo, J. Wu, W. Lin, Thinet: a filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.
[17] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, D. Wang, FBNA: a fully binarized neural network accelerator, in: Proceedings of the 2018 28th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2018, pp. 51–513.
[18] J.-H. Lin, T. Xing, R. Zhao, Z. Zhang, M.B. Srivastava, Z. Tu, R.K. Gupta, Binarized convolutional neural networks with separable filters for efficient hardware acceleration, in: Proceedings of the CVPR Workshops, 2017, pp. 344–352.
[19] W. Tang, G. Hua, L. Wang, How to train a compact binary neural network with high accuracy? in: Proceedings of the AAAI, 2017, pp. 2625–2631.
[20] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[21] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014. arXiv:1409.1556.

[22] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[23] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Technical Report (2009).
[24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning, Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011) 5.
[25] T.E. De Campos, B.R. Babu, M. Varma, et al., Character recognition in natural images, Proceedings of the International Conference on Computer Vision Theory and Applications 27.
[26] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, The German traffic sign recognition benchmark: a multi-class classification competition, in: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN), 2011, pp. 1453–1460.
[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: a large-scale hierarchical image database, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2009, pp. 248–255.
[28] S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding, arXiv:1510.00149.



Yixing Li received B.Eng and M.S. degrees in microelectronics from South China University of Technology, Guangzhou, China, in 2012 and 2015, respectively, and is currently working towards her Ph.D. degree in School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. Her current research interests focus on hardware-friendly machine learning algorithm, hardware-software co-optimization for data analytics, and computer vision applications.



Shuai Zhang received the B.S. degree from Chongqing University (CQU), Chongqing, China, in 2018. He is currently a Graduate Student with the School of Microelectronics and Communication Engineering, CQU. His current research interests include computer version and neural networks.

Xichuan Zhou received the B.S. and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 2005 and 2010, respectively. He is currently a Professor and the Vice Dean of the School of Microelectronics and Communication Engineering with Chongqing University, Chongqing, China. He has authored or co-authored over 30 papers on peer-reviewed journals, such as the IEEE Transactions on Biomedical Engineering, the IEEE Transactions on Computational Biology and Bioinformatics, the IEEE Geoscience and Remote Sensing Letters, the IEEE Transactions on Electron Devices, and the IEEE Transactions on Circuits and System I. His current research interests include parallel computing, circuits and systems, and machine learning. Dr. Zhou was a recipient of the Outstanding Young-and-Middle-Age Faculty Award of Chongqing in 2016.

Fengbo Ren received the B.Eng. degree from Zhejiang University, Hangzhou, China, in 2008 and the M.S. and Ph.D. degrees from University of California, Los Angeles, in 2010 and 2014, respectively, all in electrical engineering. In 2015, he joined the faculty of the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University (ASU). His Ph.D. research has involved in designing energy-efficient VLSI systems, accelerating compressive sensing signal reconstruction, and developing emerging memory technology. His current research interests are focused on hardware acceleration and parallel computing solutions for data analytics and information processing, with emphasis on bringing energy efficiency and signal intelligence into a wide spectrum of today's computing infrastructures, from data center server systems to wearable and Internet-of-things devices. He is a member of the Digital Signal Processing Technical Committee and VLSI Systems and Applications Technical Committee of the IEEE Circuits and Systems Society. He received the Broadcom Fellowship in 2012, the National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2017, and the Google Faculty Research Award in 2018.