# FATNN: Fast and Accurate Ternary Neural Networks*

Peng Chen, Bohan Zhuang, Chunhua Shen
Data Science & AI, Monash University, Australia

## Abstract

*Ternary Neural Networks (TNNs) have received much attention due to being potentially orders of magnitude faster in inference, as well as more power efficient, than full-precision counterparts. However, 2 bits are required to encode the ternary representation with only 3 quantization levels leveraged. As a result, conventional TNNs have similar memory consumption and speed compared with the standard 2-bit models, but have worse representational capability. Moreover, there is still a significant gap in accuracy between TNNs and full-precision networks, hampering their deployment to real applications. To tackle these two challenges, in this work, we first show that, under some mild constraints, computational complexity of the ternary inner product can be reduced by $2\times$. Second, to mitigate the performance gap, we elaborately design an implementation-dependent ternary quantization algorithm. The proposed framework is termed Fast and Accurate Ternary Neural Networks (FATNN). Experiments on image classification demonstrate that our FATNN surpasses the state-of-the-arts by a significant margin in accuracy. More importantly, speedup evaluation compared with various precision is analyzed on several platforms, which serves as a strong benchmark for further research. Source code and models are available at:* https://github.com/MonashAI/QTool

## 1. Introduction

Equipped with high-performance computing and large-scale datasets, deep convolution neural networks (DCNN) have become a cornerstone for most computer vision tasks. However, a significant obstacle for deploying DCNN algorithms to mobile/embedded edge devices with limited computing resources is the ever growing computation complexity—in order to achieve good accuracy, the models are becoming very heavy. To tackle this problem, much research effort has been spent on model compression. Representative methods include model quantization [49, 47], network pruning [23, 51] and neural architecture search for lightweight models [52, 27]. In this paper, we focus on model quantization, which reduces the model complexity by representing a network with low-precision weights and activations.

Network quantization aims to map the continuous input values within a quantization interval to the corresponding quantization level, and a low-precision quantized value is assigned accordingly. TNNs in which both the activations and weights are quantized to ternary, are particularly of interest because most of the calculations can be realized with bit operations, thus completely eliminating multiplications. However, there exists two limitations for conventional TNNs. The first limitation is the inefficient implementation of TNNs. Specifically, the ternary representation of $\{-1, 0, 1\}$ needs 2 bits to encode with one state wasted. As a result, with the conventional bitwise implementation of quantized networks [47, 44], the complexity of ternary inner product is the same with the standard 2-bit counterparts. Another limitation is the considerable accuracy drop compared with the full-precision counterparts due to the much more compact capacity.

To handle these drawbacks, we introduce a new framework, termed FATNN, where we co-design the underlying ternary implementation of computation and the quantization algorithm. In terms of implementation, we fully leverage the property of the ternary representation to design a series of bit operations to accomplish the ternary inner product with improved efficiency. In particular, FATNN reduces the computational complexity of TNNs by $2\times$, which solves the existing efficiency bottleneck. In contrast to previous works, nearly no arithmetic operations exist in the proposed implementation. Also, FATNN works efficiently on almost all kinds of devices (such as CPU, GPU, DSP, FPGA and ASIC) with basic bit operation instructions available. Furthermore, we design the compatible ternary quantization algorithm in accordance to the mild constraints derived from the underlying implementation. Early works with learned quantizers either propose to learn the quantized values [44] or seek to learn the quantization intervals [20, 13]. However, most of them assume the uniform quantizer step size, which might still be non-optimal on optimizing network

*PC and BZ contributed equally. Part of this work was done when all authors were with The University of Adelaide. CS is the corresponding author, email: chunhua@me.com

performance. To make the low-precision discrete values sufficiently fit the statistics of the data distribution, we propose to parameterize the step size of each quantization level and optimize them with the approximate gradient. Besides, we suggest calibrating the distribution between the skip connection and the low-precision branch to further improve the performance. The overall approach is usable for quantizing both activations and weights, and works with existing methods for back-propagation and stochastic gradient descent.

Our main contributions are summarized as follows:

- We propose a ternary quantization pipeline, in which we co-design the underlying implementation and the quantization algorithm. To our best knowledge, the proposed FATNN is the first solution being applied on general platforms specific to TNNs while previous acceleration methods only target on the dedicated hardware (FPGA or ASIC).
- We devise a fast ternary inner product implementation which reduces the complexity of TNNs by $2\times$ while keeping the bit-operation-compatible merit. We then design a highly accurate ternary quantization algorithm in accordance with the constraints imposed by the implementation.
- We evaluate the execution speed of FATNN and make comparison with other bit configurations on various platforms. Moreover, experiments on image classification task demonstrate the superior performance of our FATNN over a few competitive state-of-the-art approaches.

## 2. Related Work

Model quantization aims to quantize the weights, activations and even backpropagation gradients into low-precision, to yield highly compact DCNN models compared to their floating-point counterparts. As a result, most of the multiplication operations in network inference can be replaced by more efficient addition or bitwise operations. In particular, BNNs [36, 17, 46, 4, 5, 42, 38, 14, 28], where both weights and activations are quantized to binary tensors, are reported to have potentially $32\times$ memory compression ratio, and up to $58\times$ speed-up on CPU compared with the full-precision counterparts. However, BNNs still suffer from sizable performance drop issue, hindering them from being widely deployed. To make a trade-off between accuracy and complexity, researchers also study ternary [22, 48] and higher-bit quantization [47, 8, 13, 50, 26]. In general, quantization algorithms aim at tackling two core challenges. The first challenge is to design accurate quantizers to minimize the information loss. Early works use handcrafted heuristic quantizers [47] while later studies propose to adjust the quantizers to the data, basically based on matching the original data distribution [47, 6], minimizing the quantization error [44, 9] or directly optimizing the quantizer with stochastic gradient descent [8, 20, 2]. Moreover, another challenge is to approximate gradient of the non-differentiable quantizer. To solve this problem, most studies focus on improving the training via loss-aware optimization [16], regularization [12, 9, 1], knowledge distillation [49, 30], entropy maximization [31, 33] and relaxed optimization [29, 43, 21, 41]. In addition to the quantization algorithms design, the implementation frameworks and acceleration libraries [18, 7, 39, 19] are indispensable to expedite the quantization technique to be deployed on energy-efficient edge devices. For example, TBN [40] focuses on the implementation of ternary activation and binary weight networks. daBNN [45] targets at the inference optimization of BNNs on ARM CPU devices. GXNOR-Net [11] treats TNNs as a kind of sparse BNNs and propose an acceleration solution on dedicated hardware platforms. RTN [24] leverages extra linear transformation before quantization for better performance of TNNs. Similar with GXNOR-Net, RTN demonstrates the benefit of their implementation on dedicated devices (FPGA and ASIC). However, there are few works targeting on improving the inference efficiency of TNNs on general purpose computing platforms. In this paper, we propose to co-design the underlying implementation and the quantization algorithm to achieve ideal efficiency and accuracy simultaneously.

## 3. Inference Acceleration

### 3.1. Preliminary

As the inner product is one of the fundamental operations in convolution neural networks, which consumes most of the execution time, we mainly focus on the acceleration of inner product in this paper. It is worth to firstly review how the inner product between two quantized vectors are computed in previous literature. For BNNs, in which both the weights and activation are binarized to $\{-1, 1\}$, the inner product between two length-$N$ vectors $\boldsymbol{x}, \boldsymbol{y} \in \{-1, 1\}^N$ can be derived using bit-wise operations:

$$\boldsymbol{x} \cdot \boldsymbol{y} = 2 \cdot \text{popcount}(\text{xnor}(\boldsymbol{x}, \boldsymbol{y})) - N, \tag{1}$$

where popcount counts the number of bits in a bit vector. Furthermore, for quantization with more bits, the input vectors can be decomposed with a linear combination of binary bases. For example, a $M$-bit vector $\boldsymbol{x}$ can be encoded as $\boldsymbol{x} = \sum_{m=0}^{m=M-1} \boldsymbol{x}_m \cdot 2^m$ where $\boldsymbol{x}_m \in \{-1, 1\}^N$ (in practical implementation, $-1$ and $1$ are represented by $0$ and $1$ respectively). Similarly, for another $K$-bit vector $\boldsymbol{y}$, we have $\boldsymbol{y} = \sum_{k=0}^{k=K-1} \boldsymbol{y}_k \cdot 2^k$, where $\boldsymbol{y}_k \in \{-1, 1\}^N$. Based on the decomposition, the binary inner product specified in Eq. (1) can be used to compute higher bit inner product. Generally, the inner product between two quantized vectors

can be formulated as

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} \alpha_m \beta_k (\boldsymbol{x}_m \odot \boldsymbol{y}_k), \tag{2}$$

where $\odot$ is specially used to denote the binary inner product in formulation of Eq. (1), $\boldsymbol{\alpha} \in \mathbb{R}^M$ and $\boldsymbol{\beta} \in \mathbb{R}^K$ are scales to encode $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Specifically, $\alpha_m = 2^m$ and $\beta_k = 2^k$ are used in uniform fixed-point quantization [47, 25] while $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ become trainable scales for non-uniform quantization [44, 2]. Considerable speedups can be achieved by Eq. (1) and Eq. (2) because all the calculation can be realized with bit operations [17, 10, 39], thus completely eliminating multiplications.

### 3.2. Motivations for Acceleration

If assuming the computational complexity of BNNs in Eq. (1) to be $O(N)$, the computational complexity for higher bit quantization in Eq. (2) becomes $O(M \cdot K \cdot N)$. We can find that higher bit quantization algorithms acquire better task accuracy at the cost of increased computational complexity. In particular, for the TNN case, 2 bits are required for the data representation of $\{-1, 0, 1\}$. Thus the computational complexity for ternary inner product is $O(4N)$, which is the same with the standard 2-bit counterpart, however, with one of the quantization levels wasted (2 bits can express 4 quantization levels at most). As a result, the implementation in Eq. (2) makes TNNs less appealing to standard 2-bit models in practical.

To fully unleash the potential of TNNs, we further observe that the binary inner product in Eq. (1) is the core for acceleration since its multiplication and accumulation are realized by the bit operators $\mathrm{xnor}$ and $\mathrm{popcount}$, respectively. As the input of the inner product in BNNs is restricted to $\{-1, 1\}$, the multiplication result is also within the set $\{-1, 1\}$ , which we call the "**non-overflow**" property. The multiplication result can be directly obtained via $\mathrm{xnor}$ between the input vectors and it owns the attribute that only two states exist, thus $\mathrm{popcount}$ can be used to realize accumulation by simply counting the number of state "1" (or state "$-1$"). As a result, Eq. (1) enables the same parallelism degree[1] for $\mathrm{xnor}$ and $\mathrm{popcount}$, with the ALU register fully utilized. Interestingly, we find the ternary quantized values $\{1, 0, -1\}$ also meets the "non-overflow" property. Thus, it is potential for the TNNs to be executed in the same parallelism degree manner for the multiplication (*i.e.*, $\mathrm{xnor}$) and accumulation (*i.e.*, $\mathrm{popcount}$) operations. Moreover, we can use this property to design a novel ternary inner product implementation with a reduced complexity of $O(2N)$.

[1]The same parallelism degree indicates the data amount processed is the same per instruction. More explanations are put in Section S1 in the supplementary file.

**Table 1** – The correspondence mapping between the quantized data space and the codec space employed in the proposed solution. Both 2'b01 and 2'b10 are taken to represent the "0" value. The design of the codec owns the attribute of "popcount(codec) = data +1".

| data | $-1$ | 0 | 0 | 1 |
|---|---|---|---|---|
| codec | 2'b00 | 2'b01 | 2'b10 | 2'b11 |

### 3.3. Ternary Network Acceleration

We now elaborate the design of the fast ternary inner product implementation. First, it is worth noting that the ternary values $\{-1, 0, 1\}$ will be represented by the corresponding codec in the practical implementation. We elaborately design the mapping between the logical level ternary values and their implementation level codec as illustrated in Table 1. Interestingly, we observe a property that the number of "1" in each value's codec equals the value plus one. Therefore, we can compute the inner product between two ternary vectors $\boldsymbol{x}, \boldsymbol{y} \in \{-1, 0, 1\}^N$ as follows:

$$\boldsymbol{x} \cdot \boldsymbol{y} = \mathrm{popcount}(\mathrm{TM}(\boldsymbol{x}, \boldsymbol{y})) - N, \tag{3}$$

where $\mathrm{TM}(\cdot)$ indicates the ternary multiplication, $N$ is the vector length. In Eq. (3), we first compute the inner product in the codec space by $\mathrm{popcount}(\mathrm{TM}(\boldsymbol{x}, \boldsymbol{y}))$, which consists of pure bit operations. Then we simply subtract $N$ from the result to transform it into the inner product of the logical level ternary vectors.

Second, we illustrate the way to design $\mathrm{TM}(\cdot)$ with pure bit operations[2]. For easy understanding, the true value table of ternary multiplication is listed in Table 2. Since two bits are required to encode the ternary input value and the ternary multiplication result, there are 16 possibilities with respect to the codec. From Table 2, we observe that most rows in the table, except the bold ones, still follow the rule of $\mathrm{xnor}$:

$$xnor = \sim (x_c \wedge y_c), \tag{4}$$

where $x_c$ and $y_c$ are the codec representation of the element in $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Therefore, the ternary multiplication can be realized by $\mathrm{xnor}$ along with the exception cases fixed.

After reviewing the $\mathrm{xnor}$ correct cases and exception cases, we summarize that the exception ones only happen when both of the operands are $0$. Therefore, the ternary multiplication result can be fixed by locating the "zero operand" cases and forcing the result to be $0$ (we force the result to be $0$ if "zero operand" is detected no matter whether it is the $\mathrm{xnor}$ incorrect case or not). Overall, our implementation of the ternary multiplication consists of three steps: 1): Obtain the intermediate result by the $\mathrm{xnor}$

[2]We follow the C/C++ grammar in the equations. For example, & means "AND", $\sim$ indicates "NOT", $\wedge$ represents "XOR".

**Table 2** – True value table of ternary multiplication.

| $x_c$ | $\cdot$ | $y_c$ | $=$ | $z_c$ |
|---|---|---|---|---|
| $-1$(2'b00) | | $-1$(2'b00) | | 1(2'b11) |
| $-1$(2'b00) | | 0(2'b01) | | 0(2'b10) |
| $-1$(2'b00) | | 0(2'b10) | | 0(2'b01) |
| $-1$(2'b00) | | 1(2'b11) | | $-1$(2'b00) |
| 0(2'b01) | | $-1$(2'b00) | | 0(2'b10) |
| **0(2'b01)** | | **0(2'b01)** | | **0(2'b01)** |
| **0(2'b01)** | | **0(2'b10)** | | **0(2'b01)** |
| 0(2'b01) | | 1(2'b11) | | 0(2'b01) |
| 0(2'b10) | | $-1$(2'b00) | | 0(2'b01) |
| **0(2'b10)** | | **0(2'b01)** | | **0(2'b10)** |
| **0(2'b10)** | | **0(2'b10)** | | **0(2'b10)** |
| 0(2'b10) | | 1(2'b11) | | 0(2'b10) |
| 1(2'b11) | | $-1$(2'b00) | | $-1$(2'b00) |
| 1(2'b11) | | 0(2'b01) | | 0(2'b01) |
| 1(2'b11) | | 0(2'b10) | | 0(2'b10) |
| 1(2'b11) | | 1(2'b11) | | 1(2'b11) |

operation. 2): Identify the 0 operand. 3): Fix the exceptions and obtain the ternary multiplication result. The first step is easily achieved by Eq. (4). For the second step, we here introduce an auxiliary variable $auxi$ which is a predefined constant in codec $2'b01$. In fact, the auxiliary variable $auxi$ indicates a zero value variable (one codec of the 0 value is $2'b01$), which acts as a mask to fetch specific bits in operands. Then we propose to identify the 0 operand using the following bit operations:

$$switch = ((y_c >> 1)\ \&\ auxi)\ |\ ((y_c << 1)\ \&\ \sim auxi), \tag{5}$$

$$mask = switch \wedge y_c, \tag{6}$$

With the shift and mask operations, Eq. (5) actually results in the exchange of the two sequence bits in the operand. After that, Eq. (6) generates the mask information by distinguishing whether the operand is 0 (in codec $2'b01$ or $2'b10$) or not. Specifically, the $mask$ variable in Eq. (6) will be $2'b11$ if the 0 operand is detected and $2'b00$ otherwise. After identifying the 0 operand, we can easily fix the exceptions and obtain the final ternary multiplication result as:

$$\mathrm{TM}(\cdot) = (mask\ \&\ auxi)\ |\ ((\sim mask)\ \&\ xnor). \tag{7}$$

If the $mask$ is $2'b11$ (the 0 operand is detected), then Eq. (7) reduces to $auxi$ which equals 0. In contrast, if the $mask$ is $2'b00$, then Eq. (7) becomes $xnor$, where the correctness can be examined in Table 2. In practise, we use the weight parameter ($y_c$) to generate $mask$ which is determined after training.

**Remark 1** We can derive from Eq. (3) that the computational complexity of the proposed ternary inner product is $O(2N)$. In other words, FATNN can reduce the computational complexity of TNNs from $O(4N)$ to $O(2N)$, which significantly improves the efficiency (memory consumption is not changed). Therefore, even though TNNs do not make full use of the 2-bit representational capacity, they fortunately enjoy the faster implementation than the standard 2-bit models. Besides, our solution can adapt to general purpose computing platforms, such as CPU, GPU and DSP. Note that, the extra bit operations introduced in Eqs. (5), (6) and (7) have negligible runtime overhead for deployment, as these extra bit operations are much faster than the accumulation operation in Eq. (3) [3]. We further provide extensive benchmark results in the experiment section to justify our analysis.

**Constraints on the algorithm.** From the formulation discussed above, it can be learned that the designed ternary implementation has certain requirements on the quantization algorithm. The constraints are summarized as follows:

- The ternary values for the network are limited to $\{-1, 0, 1\}$. One and only one additional high precision coefficient is allowed to adjust the scale of the quantized values. More than one scale coefficients will break Eq. (3). It indicates methods such as TTN [48], in which two trainable variables $(W_p, W_n)$ are learned, cannot be applied to our method.
- A special case exists for the activation quantization when the ReLU non-linearity is applied, which leads to a non-negative data range. In this situation, we advise to modify the ternary values to $\{0, 1, 2\}$ for activations. The revision does not conflict with the first constraint as in the inference procedure, it results in an additional constant on the output (simply add a copy of weights on the result which are fixed after training).

## 4. Constrained Ternary Quantization

### 4.1. Quantization Function Revisited

The quantization functions can be categorized into uniform [13, 47] and non-uniform [44, 20] ones. In particular, LSQ [13] proposes to parameterize the uniform step size and achieves state-of-the-art performance. In this section, we show that the principle of parameterized step size can be applied to solving the non-uniform ternary quantization.

Let us first consider the ternary weight quantization. In particular, we parameterize the three step sizes by introducing two learnable parameters $\{\alpha_1, \alpha_2\}$. In this way, the full-precision data is partitioned into three levels with the quantization thresholds $\{-\alpha_1/2, \alpha_2/2\}$, where each step size can be adjusted accordingly during training. Then we define the weight quantizer $Q^w(p; \alpha_1, \alpha_2)$ for data $p$ parameterized by the scale factors $\{\alpha_1, \alpha_2\}$. Specifically, $Q^w(p; \alpha_1, \alpha_2)$ performs quantization by applying

[3] Refer to the implementation details in Section S1 in the supplementary material.

three point-wise operations in order: normalization, saturate and round.

**Normalization**: Since we do non-uniform quantization, the tensor elements are firstly normalized by the scale factors $\alpha_1$ and $\alpha_2$, respectively. This operation aims to map the data from the floating-point domain to the quantized domain. **Saturate**: Once normalized, the tensor elements that are out of the range of the quantized domain are clipped accordingly: $\text{clip}(p; \beta_1, \beta_2) = \min(\max(p, \beta_1), \beta_2)$, where the scalar clipping limits $\{\beta_1, \beta_2\}$ are independent with the full-precision data range. **Round**: We discretize the normalized and tailored tensor elements to nearest integers using bankers rounding denoted by $\lfloor \cdot \rceil$. Putting the above point-wise operations together, the weight quantization function can be written as:

$$Q^w(p) = \begin{cases} \lfloor \text{clip}(p/\alpha_1, -1, 0) \rceil & \text{if } p < 0 \\ \lfloor \text{clip}(p/\alpha_2, 0, 1) \rceil & \text{otherwise} \end{cases}, \quad (8)$$

For simplicity, Eq. (8) can be re-written into a unified form:

$$Q^w(p) = \lfloor \text{clip}(p/\alpha_1, -1, 0) \rceil + \lfloor \text{clip}(p/\alpha_2, 0, 1) \rceil. \quad (9)$$

Since the bankers rounding is non-differentiable, we use the straight through estimator (STE) [3] to approximate the gradient through the round function $\lfloor \cdot \rceil$ as a pass through operation, and differentiating all other operations normally [32].

For activation quantization, there exists two cases. On the one hand, if $p$ is in the real domain (e.g., use PReLU or Tanh as non-linearity), the activation quantizer is the same as $Q^w(\cdot)$. On the other hand, if $p$ is in the non-negative domain (applicable to ReLU activations), the quantized values should be adapted accordingly. In this case, the activation quantizer becomes:

$$Q^a(p) = \lfloor \text{clip}(p/\alpha_1, 0, 1) \rceil + \lfloor \text{clip}(p - \alpha_1)/\alpha_2, 0, 1) \rceil. \quad (10)$$

Note that we learn independent step sizes for weights and activations of each quantized layer.

**Remark 2** Above non-uniform step size quantization algorithm can be easily integrated in the general deep learning training framework. It is worth noting that the algorithm meets all the constraints derived in Section 3.3, which only have requirements on the quantized values while having no requirements on the quantization thresholds. We further visualize the non-uniform step sizes in Section S2 in the supplementary file to provide more insights.

The proposed algorithm is featured in learning the non-uniform quantization thresholds, which is different from two categories of methods. Compared with methods using a uniform step size [47, 8, 13], our FATNN enjoys more flexible step sizes to better fit the statistics of the data distribution. Moreover, different from non-uniform LQ-Net [44] that updates the quantizer via closed-form approximation and QIL [20] that introduces extra non-linear transformation and hyperparameters with careful tuning during optimization, our FATNN directly parameterizes and updates the quantization intervals by efficient stochastic gradient descent, which shows better performance in Section 5.

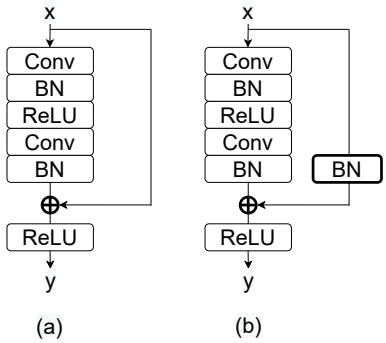


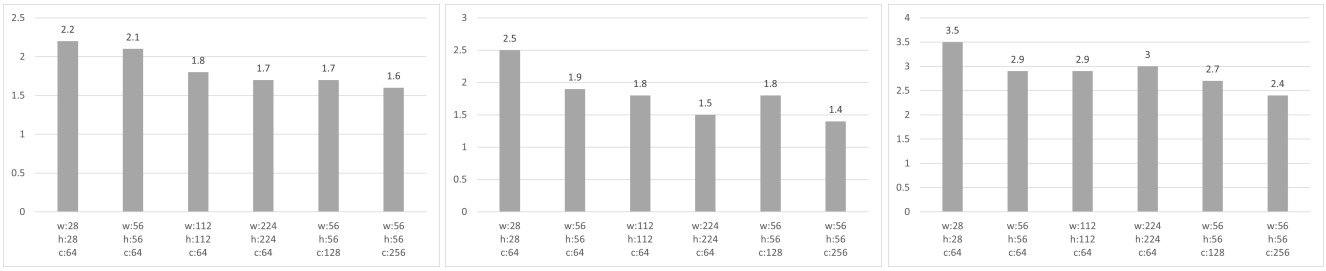

**Figure 1** – (a). Classical residual block. (b). Calibration between different branches by inserting an extra batch normalization (BN) layer into the identity mapping path.

## 4.2. Calibration of Residual Block

It has been shown in the quantization literature that high-precision skip connections are essential to reduce the accumulated quantization error and ease the difficulty in propagating gradients through a low-precision network due to the non-differentiable quantization function. However, few work pays attention to the influence of quantization on the branch fusion in the residual architecture.

As illustrated in Figure 1 (a), the distributions of the two inputs of the element-wise addition, the non-negative block input $\boldsymbol{x}$ (from the preceding residual block) and the output of the batch normalization layer, are different. The large covariate shift between the two inputs results in enlarged variance and value range for the output tensor. It might not cause a problem for the full-precision network, as the representational capability with 32-bit precision is enough to encode all the information. However, it can lead to increased information loss for the low-precision network, because of the deteriorated quantization error when quantizing the enlarged range of value into a set of limited discrete values (*e.g.*, ternary quantization).

To solve this problem, we propose to calibrate the distribution between the skip connection and the low-precision branch. As illustrated in Figure 1(b), the calibration can be done by inserting an extra batch normalization layer in the identity-mapping skip connection path. The calibration helps to align the two inputs of the element-wise addition, reducing the quantization error when the output tensor is quantized in the succeeding block. Note that the extra memory consumption is limited as the batch normalization layers




**(a)** Speedup on Qualcomm 821 GPU **(b)** Speedup on Qualcomm 835 GPU **(c)** Speedup on Nvidia 2080Ti GPU

**Figure 2** – Layer-wise speedup of ternary networks ('ter') versus 2-bit counterparts ('2-bit') on different platforms.

only introduce a few parameters (*i.e.*, two scalar parameters for each after training). We leverage the BN fusion when the preceding layer is a convolutional layer. Otherwise, extra computation is introduced by the BN layer, typically in the calibration branch. But the runtime overhead is negligible for deployment since the computational cost of BN is orders of magnitude lower than that of the convolutional layer. In practice, we find the execution time of all "unfused" BN layers only occupies less than 1% of that of the whole network.

# 5. Experiments

## 5.1. Acceleration

In this paper, we propose a fast ternary inner product implementation with $O(2N)$ complexity, which is potentially $2\times$ faster than the implementation in previous works [47, 44]. To further justify its effectiveness in practice, we develop the acceleration code (in C++ and OpenCL) for binary, ternary and 2-bit convolutions[4]. Measurements of the actual execution time on different devices are conducted. We cover both the embedded-side devices (Qualcomm 821 and Qualcomm 835[5]) and server-side devices (Nvidia 2080Ti) to demonstrate the flexibility of our solution. We keep the experimental setting the same across all devices.

### 5.1.1 Layer-wise Speedup

First, we report the layer-wise speedup in Figure 2, where convolution layers (kernel size = $3\times3$, padding = 1, stride = 1, the same number of input and output channels and batch size = 1) with six different shape configurations are tested. For the first four cases on each platform in Figure 2, we fix the channel number to be 64 and increase the resolution from 28 to 224. For the last two cases, we fix the resolution to be 56 and double the channel number from 64 to 256. We report the relative acceleration ratios between ternary and 2-bit models on Qualcomm 821, 835 and Nvidia 2080Ti platforms in Figure 2. From Figure 2, we observe that, compared with the 2-bit quantization, the proposed ternary quantization is much faster. Specifically, the ternary convolutional layer is $1.6\times$ to $2.2\times$ faster on the Qualcomm 821 platform, $1.4\times$ to $2.5\times$ faster on the Qualcomm 835 platform and $2.4\times$ to $3.5\times$ faster on the Nvidia 2080Ti platform. The variance of the speedup on different platforms is caused by the macro-architectures of different devices. More detailed results and analysis can be found in Section S3 in the supplementary file.

### 5.1.2 Overall Speedup

Besides the layer-wise analysis, it is also interesting to get a knowledge of the overall speed for some classical networks. We present the whole execution time of all quantized layers (first and last layers are excluded while non-linear and skip connection layers are included) for ResNet-18 and ResNet-34 on ImageNet in Table 3. We set the batch size to 1. From Table 3, we can learn that both the 2-bit and ternary models run faster than the theoretical speedup versus the binary counterpart (2 and 4 respectively). Moreover, the speedup of the ternary models compared with the 2-bit ones (last column) again demonstrates that our proposed FATNN can run about $2\times$ faster than the 2-bit models or conventional TNNs.

### 5.1.3 Ablation Study

In Remark 1, we claim the influence of the extra bit operations is negligible. To verify the impact, we test the exact execution time of the six cases introduced in Section 5.1.1 on different devices in Table 4. The total execution time of the implementation described in Section 3.3 is taken as the baseline. We then measure the cost with respect to the extra bit operations and list the absolute time and relative ratios in Table 4. From Table 4, we observe that the influence is highly relative to the devices and layer

[4]We include the implementation details in Section S1 of the supplementary material.

[5]We conduct experiments on Google Pixel and Xiaomi 6 phones which equip with the Qualcomm 821/835 chips respectively. Any platform with the same chip is expected to produce the similar result.

**Table 3** – Execution time (ms) and speedup ratios for overall quantized layers. "bin" means binary and "ter" is ternary. 'Q821' and 'Q835' indicate the platform of Qualcomm 821 and Qualcomm 835, respectively. We run 5 times and report the results with mean and standard deviation.

| Device | Network | bin | ter | 2-bit | bin vs. ter | bin vs. 2-bit | ter vs. 2-bit |
|---|---|---|---|---|---|---|---|
| Q835 | ResNet-18 | 12.1±0.2 | 20.1±0.2 | 43.0±0.7 | 1.7 | 3.6 | 2.1 |
| | ResNet-34 | 25.3±0.3 | 42.1±0.5 | 87.0±1.5 | 1.7 | 3.4 | 2.1 |
| Q821 | ResNet-18 | 15.7±0.3 | 25.2±0.3 | 52.3±1.0 | 1.6 | 3.3 | 2.1 |
| | ResNet-34 | 32.3±0.6 | 51.4±0.6 | 105.2±2.1 | 1.6 | 3.3 | 2.0 |

**Table 4** – Influence of the extra bit operations in the FATNN implementation. 'base' indicates the total layer-wise execution time ($\mu s$) of the ternary convolutional layer. 'extra' represents the layer-wise execution time of the extra bit operations described in Eqs. (5)-(7). We report both absolute time and relative ratios. We run 5 times and report the mean results. 'Q821', 'Q835' and '2080Ti' indicate the platforms of Qualcomm 821, Qualcomm 835 and Nvidia 2080Ti, respectively.

| Device | | case1 | case2 | case3 | case4 | case5 | case6 |
|---|---|---|---|---|---|---|---|
| Q821 | base | 594 | 1190 | 3425 | 12373 | 3631 | 13202 |
| | extra | 61 (10.3%) | 56 (4.7%) | 36 (1.1%) | 21 (0.2%) | 95 (2.6%) | 676 (5.1%) |
| Q835 | base | 485 | 1007 | 2847 | 11023 | 2606 | 12122 |
| | extra | 155 (32.0%) | 150 (14.9%) | 471 (16.5%) | 177 (1.6%) | 5 (0.2%) | 268 (2.2%) |
| 2080Ti | base | 12 | 15.5 | 29 | 92.5 | 35.5 | 88.5 |
| | extra | 1 (8.3%) | 1 (6.5%) | 3 (10.3%) | 2.5 (2.7%) | 1.5 (4.2%) | 1.5 (1.7%) |

shapes. On Qualcomm 821, Qualcomm 835 and Nvidia 2080Ti, the relative ratios of the extra bit operations range from $0.2\%$ to $10.3\%$, $0.2\%$ to $32.0\%$ and $1.7\%$ to $10.3\%$ on the evaluated cases, respectively. Besides, the occurrence of the $32.0\%$ ratio happens only on the small input shape ($\mathrm{c} = 64, \mathrm{w} = \mathrm{h} = 28$) while most of the larger input sizes have relative small ratios. Together with the overall speedup in Section 5.1.2, we can conclude that the extra bit operations have a small impact on the total execution time.

## 5.2. Quantization Accuracy

We perform experiments on the ImageNet [37] dataset. The ImageNet contains about 1.2 million training and 50K validation images of 1,000 object categories. To verify the effectiveness of the proposed auxiliary learning strategy, we compare with various representative quantization approaches, including uniform approaches LSQ [13] and DoReFa-Net [47] and non-uniform methods LQ-Net [44], HWGQ [6] and QN [43]. Ternary-oriented quantization methods, such as RTN [24], TBN [40] and LSB [34] are also included. Comparisons on other datasets, such as CIFAR-10, can be found in Section S4 in the supplementary file.

### 5.2.1 Evaluation on ImageNet

**Experimental setup.** For ImageNet classification, all the images are re-scaled with the shorter edge to be 256. Training images are then randomly cropped into resolution of $224 \times 224$. After that, the images are normalized using the mean and standard deviation. No additional augmentations are performed except the random horizontal flip. Validation images follow a similar procedure except the random crop is replaced with the center crop and no flip is applied. We conduct experiments on the ResNet models [15]. As in previous works [47, 44], we do not quantize the first and last layers. Following LSQ [13] and IR-net [35], we leverage weight normalization along with the standard batch normalization during training to make the optimization more stable. If not specially mentioned, the initial learning rate is set to 1e-2 and the cosine annealing decay is employed. Other default hyper-parameters include: SGD optimizer with a momentum of $0.9$, a weight decay of 2e-5, and a maximum training epoch of 90. The quantization related parameters $\alpha_1$ and $\alpha_2$ are initialized to 1.0 for weights and activations in all quantized layers. We initialize the quantized network with the pretrained full-precision weights at the beginning of the quantization.

**Performance analysis.** Firstly, we compare with the current best performed quantization algorithm LSQ, which learns a uniform step size and thus is directly comparable to our FATNN. To make a fair comparison, we implement LSQ under exactly the same training process with FATNN. We report the quantization results in Table 5. From Table 5, we observe steady Top-1 accuracy improvement of our FATNN over LSQ on all comparing architectures in the ternary case. This result strongly justifies the effectiveness of the proposed distribution-aware quantization strategy. To be emphasized, our FATNN boosts the implementation speed of LSQ by a factor of two according to Table 3 while still achieving the state-of-the-art accuracy.

Secondly, to make a comprehensive comparison, we

**Table 5** – Accuracy (%) comparisons between our FATNN and other algorithms on ImageNet. "A/W" in the second column indicates the bit configuration for activations and weights respectively. "ter" denotes ternary. Results for LSQ are based on our own implementation. Results for algorithms, including TBN, RTN, LSB, LQ-Net, HWGQ, DoReFa-Net and QN, are directly cited from the original papers.

| Method | A/W | ResNet-18 | | ResNet-34 | | ResNet-50 | |
|---|---|---|---|---|---|---|---|
| | | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 |
| | 32/32 | 69.8 | 89.1 | 73.3 | 91.4 | 76.1 | 92.9 |
| **FATNN** (Ours) | ter/ter | **66.0** | **86.4** | **69.8** | **89.1** | **72.4** | **90.6** |
| LSQ [13] | ter/ter | 64.7 | 85.6 | 69.0 | 88.8 | 71.2 | 90.1 |
| RTN [24] | ter/ter | 64.5 | - | - | - | - | - |
| TBN [40] | ter/1 | 55.6 | 79.0 | 58.2 | 81.0 | - | - |
| LSB [34] | ter/1 | 62.0 | 83.6 | - | - | - | - |
| LSQ [13] | 2/1 | 64.9 | 85.8 | 69.1 | 88.8 | 71.0 | 90.0 |
| LQ-Net [44] | 2/1 | 62.6 | 84.3 | 66.6 | 86.9 | 68.7 | 88.4 |
| HWGQ [6] | 2/1 | 59.6 | 82.2 | 64.3 | 85.7 | 64.6 | 85.9 |
| DoReFa-Net [47] | 2/1 | 53.4 | - | - | - | - | - |
| QN [43] | 2/1 | 63.4 | 84.9 | - | - | - | - |

also include the results of the "2-bit activation and binary weight" networks which have a similar computational complexity $O(2N)$ with our FATNN. We can learn from Table 5 that our FATNN is able to achieve steady accuracy gain over the non-uniform quantization algorithms, such as LQ-Net, HWGQ and QN. For example, FATNN outperforms LQ-Net by 3.4% on the Top-1 accuracy on ResNet-18. This justifies the superiority of our gradient-based approach to learn the step sizes. Moreover, compared to the uniform quantization algorithms, for example DoReFa-Net and LSQ, we also achieve the best performance on various architectures. This further shows that the employed algorithm can better fit the statistics of the data distribution to reduce the quantization error.

**Table 6** – Impact of the distribution-aware ternary quantization algorithms with ResNet-18 on ImageNet. 'N' indicates the non-uniform step size quantization algorithm. 'C' implies the distribution calibration between the skip connection and the low-precision branch.

| Method | A/W | N | C | Top-1 | Top-5 |
|---|---|---|---|---|---|
| | 32/32 | | | 69.8 | 89.1 |
| | 32/32 | | ✓ | 69.6 | 88.9 |
| LSQ [13] (baseline) | ter/ter | | | 64.7 | 85.6 |
| **FATNN** (Ours) | ter/ter | ✓ | | 65.4 | 86.2 |
| | ter/ter | ✓ | ✓ | **66.0** | **86.4** |

### 5.2.2 Ablation Study

In this section, we further investigate the effect of the distribution-aware quantization strategies proposed in Section 4, including the non-uniform quantization functions and the calibration of residual block, to the final performance respectively. We conduct the experiments with ResNet-18 on ImageNet and report the results in Table 6. We observe that the branch calibration strategy deteriorates the performance in the full-precision setting, leading to $0.2\%$ Top-1 accuracy drop. In contrast, the calibration strategy brings $0.6\%$ performance improvement to our ternary FATNN in terms of the Top-1 accuracy. It empirically justifies that calibrating the residual block is particularly designed for the low-precision network. Moreover, the non-uniform step size quantization of FATNN further brings $0.7\%$ Top-1 accuracy increase, which justifies that the parameterized quantization thresholds can fit the statistics of the data distribution effectively.

## 6. Conclusion

In this paper, we have proposed a fast ternary neural network, named FATNN. Specifically, we emphasize that the underlying implementation and the quantization algorithm are highly correlated and should be co-designed. From the implementation perspective, we exploit the "non-overflow" property to design a novel ternary inner product with fully bit operations. As a result, our FATNN can achieve $2\times$ less complexity than the conventional TNNs. Moreover, we have designed an efficient quantization algorithm in accordance with the constraints of the implementation. Extensive experiments on image classification task demonstrate that FATNN improves the previous TNNs in both execution time and quantization accuracy. Besides, the proposed quantization algorithm does not pose any constraints on the network architecture design and task loss function. It implicates that the proposed method can be leveraged on other tasks, such as object detection and segmentation. It is also valuable to explore the ternary quantization on recent Transformer architecture, which is left in the future work. Thus, we advocate to rethink the value of TNNs and hope that FATNN can serve as a strong benchmark for further research.

## References


[1] Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[2] Chaim Baskin, Eli Schwartz, Evgenii Zheltonozhskii, Natan Liss, Raja Giryes, Alex M Bronstein, and Avi Mendelson. Uniq: Uniform noise injection for non-uniform quantization of neural networks. *arXiv preprint arXiv:1804.10969*, 2018.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Joseph Bethge, Marvin Bornstein, Adrian Loy, Haojin Yang, and Christoph Meinel. Training competitive binary neural networks from scratch. *arXiv preprint arXiv:1812.01965*, 2018.

[5] Joseph Bethge, Haojin Yang, Christian Bartz, and Christoph Meinel. Learning to train a binary neural network. *arXiv preprint arXiv:1809.10463*, 2018.

[6] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 5918–5926, 2017.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. TVM: An automated end-to-end optimizing compiler for deep learning. In *USENIX Symp. Operating Systems Design & Implementation*, pages 578–594, 2018.

[8] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[9] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Learning low precision deep neural networks through regularization. *ArXiv*, abs/1809.00095, 2018.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 3123–3131, 2015.

[11] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, 100:49–58, 2018.

[12] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 11408–11417, 2019.

[13] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *Proc. Int. Conf. Learn. Repren.*, 2020.

[14] Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. Network sketching: Exploiting binary structure in deep cnns. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 5955–5963, 2017.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 770–778, 2016.

[16] Lu Hou and James T Kwok. Loss-aware weight quantization of deep networks. In *Proc. Int. Conf. Learn. Repren.*, 2018.

[17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 4107–4115, 2016.

[18] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[19] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[20] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4350–4359, 2019.

[21] Cong Leng, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with ADMM. *CoRR*, abs/1707.09870, 2017.

[22] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2017.

[24] Yuhang Li, Xin Dong, Sai Qian Zhang, Haoli Bai, Yuanpeng Chen, and Wei Wang. Rtn: Reparameterized ternary network. In *AAAI*, 2020.

[25] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *Proc. Int. Conf. Mach. Learn.*, pages 2849–2858, 2016.

[26] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 344–352, 2017.

[27] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[28] Zechun Liu, Wenhan Luo, Baoyuan Wu, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Binarizing deep network towards real-network performance. *arXiv preprint arXiv:1811.01335*, 2018.

[29] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[30] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *Proc. Int. Conf. Learn. Repren.*, 2018.

[31] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 5456–5464, 2017.

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Proc. Adv. Neural Inf. Process. Syst. Workshops*, 2017.

[33] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *Proc. Int. Conf. Learn. Repren.*, 2018.

[34] Hadi Pouransari, Zhucheng Tu, and Oncel Tuzel. Least squares binary quantization of neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn. Workshops*, 2020.

[35] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

[36] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. Eur. Conf. Comp. Vis.*, pages 525–542, 2016.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *Int. J. Comp. Vis.*, 115(3):211–252, 2015.

[38] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Proc. AAAI Conf. on Arti. Intel.*, pages 2625–2631, 2017.

[39] Yaman Umuroglu, Nicholas Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[40] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *Proc. Eur. Conf. Comp. Vis.*, September 2018.

[41] Peisong Wang, Qinghao Hu, Yifan Zhang, Chunjie Zhang, Yang Liu, and Jian Cheng. Two-step quantization for low-bit neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4376–4384, 2018.

[42] Haojin Yang, Martin Fritzsche, Christian Bartz, and Christoph Meinel. Bmxnet: An open-source binary neural network implementation based on mxnet. In *Proc. of the ACM Int. Conf. on Multimedia.*, pages 1209–1212. ACM, 2017.

[43] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, June 2019.

[44] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[45] Jianhao Zhang, Yingwei Pan, Ting Yao, He Zhao, and Tao Mei. dabnn: A super fast inference framework for binary neural networks on arm devices. In *Proc. ACM Int. Conf. Multimedia*, pages 2272–2275, 2019.

[46] Tianli Zhao, Xiangyu He, Jian Cheng, and Jing Hu. Bitstream: Efficient computing architecture for real-time low-power inference of binary neural networks on CPUs. In *Proc. ACM Int. Conf. Multimedia*, pages 1545–1552, 2018.

[47] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[48] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. In *Proc. Int. Conf. Learn. Repren.*, 2017.

[49] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[50] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Structured binary neural network for accurate image classification and semantic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[51] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 875–886, 2018.

[52] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *Proc. Int. Conf. Learn. Repren.*, 2017.