# Unsupervised Network Quantization via Fixed-Point Factorization

Peisong Wang, Xiangyu He, Qiang Chen, Anda Cheng, Qingshan Liu, *Senior Member, IEEE*, and Jian Cheng, *Member, IEEE*

***Abstract*— The deep neural network (DNN) has achieved remarkable performance in a wide range of applications at the cost of huge memory and computational complexity. Fixed-point network quantization emerges as a popular acceleration and compression method but still suffers from huge performance degradation when extremely low-bit quantization is utilized. Moreover, current fixed-point quantization methods rely heavily on supervised retraining using large amounts of the labeled training data, while the labeled data are hard to obtain in the real-world applications. In this article, we propose an efficient framework, namely, fixed-point factorized network (FFN), to turn all weights into ternary values, i.e., {−1, 0, 1}. We highlight that the proposed FFN framework can achieve negligible degradation even without any supervised retraining on the labeled data. Note that the activations can be easily quantized into an 8-bit format; thus, the resulting networks only have low-bit fixed-point additions that are significantly more efficient than 32-bit floating-point multiply–accumulate operations (MACs). Extensive experiments on large-scale ImageNet classification and object detection on MS COCO show that the proposed FFN can achieve about more than 20× compression and remove most of the multiply operations with comparable accuracy. Codes are available on GitHub at https://github.com/wps712/FFN.**

***Index Terms*— Acceleration, compression, deep neural networks (DNNs), fixed-point quantization, unsupervised quantization.**

## I. Introduction

DEEP neural networks (DNNs) have recently shown significant improvements over traditional learning methods in many fields. On one hand, these breakthroughs promote the demands of applying the state-of-the-art DNN models to real-world applications. On the other hand, these models are hard to deploy in real-world systems due to the huge computational complexity and the storage footprint, especially for real-time but resource-limited systems. Take Alexnet [1] as an example, which involves 61M floating-point parameters and 725M high-precision operations at the inference stage. More powerful networks, such as VGG or ResNet, tend to have even higher computational complexity. Such an amount of resource consumption is prohibitive in real-world applications, for example, in automatic driving cars, intelligent surveillance cameras, and robotics. These devices usually have very limited resources. The huge parameters and computations of deep models may quickly exhaust the storage, memory, battery, and computing units of these devices. Thus, it is of central importance to reduce the complexity of networks, i.e., to lower the huge parameter size and reduce the computational complexity.

Under this circumstance, network compression and acceleration have attracted widespread attention among researchers throughout the world. Many distinguished approaches are proposed and improved by the community, such as network pruning [2]–[4], matrix/tensor decomposition [5]–[8], and network quantization [9]–[11]. Network pruning can dramatically reduce the parameter size, for example, the VGG16 can be reduced by 13× with no loss of accuracy [2]. However, the resulting sparse network is not hardware-friendly due to the cache-miss problem; thus, the very limited speedup can be achieved. In contrast, decomposition-based approaches can notably speed up the execution; however, the compression is not remarkable.

Among these approaches, fixed-point quantization is widely used, which partially alleviates the abovementioned two drawbacks, i.e., fixed-point quantization can not only dramatically reduce the model size but also is hardware-friendly. In the earlier works [12]–[14], network parameters are quantized into low-bit numbers with no loss of accuracy. However, weight quantization using higher than 2-bit still involves a large amount of multiply operations. More recently, binary [9], [10], [15], [16] or ternary quantization [17], [18] focuses on training networks from scratch with binary (+1 and −1) or ternary (+1, 0, and −1) weights. High performance is achieved on simple tasks, such as character recognition. However, on more complicated tasks, such as ImageNet classification or object detection, the accuracy degradation is not negligible.

In addition to performance degradation, current fixed-point quantization methods rely heavily on supervised training using

Manuscript received July 25, 2019; revised December 10, 2019 and March 28, 2020; accepted June 28, 2020. Date of publication July 24, 2020; date of current version June 2, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61906193, in part by the Strategic Priority Research Program of the Chinese Academy of Science under Grant XDB32050200, in part the Advance Research Program under Grant 31511130301, in part by National Key Research and Development Plan of China under Grant 2018AAA0103304, and in part by the Jiangsu Frontier Technology Basic Research Project under Grant BK20192004. *(Corresponding author: Jian Cheng.)*

Peisong Wang, Xiangyu He, Qiang Chen, Anda Cheng, and Jian Cheng are with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100049, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: peisong.wang@nlpr.ia.ac.cn; xiangyu.he@nlpr.ia.ac.cn; qiang.chen@nlpr.ia.ac.cn; jcheng@nlpr.ia.ac.cn).

Qingshan Liu is with the B-Data Laboratory, Nanjing University of Information Science and Technology, Nanjing 210044, China (e-mail: qsliu@nuist.edu.cn).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.3007749

large amounts of labeled data. This quantization scheme clearly has some prohibitive costs. First, massive training data are hard to obtain due to the privacy policy. Moreover, even if we have access to a large amount of data, data labeling is also a tedious and time-consuming job. Moreover, for some areas, such as medical images, domain knowledge is required for data labeling. Thus, unsupervised network quantization using limited unlabeled data is of central importance for applying quantization techniques to real-world tasks.

In this article, we introduce a novel ternary weight ($-1$, 0, and 1) quantization framework, named fixed-point factorized networks (FFNs), to accelerate and compress DNN models with only minor performance degradation. Instead of directly quantizing the weights of a given network, FFN utilizes a decomposition-based quantization scheme, i.e., to factorize the full-precision weight matrix into the multiplication of multiple low-bit matrices. The FFN can make full use of the pretrained full-precision models, leading to superior performances among others.

Besides the flexibility and high performance in the fully supervised quantization scheme, we also propose the unsupervised quantization scheme for FFN when no label or even no data are provided. We show that FFN trained with limited unlabeled data could outperform previous quantization methods trained supervisely using labels. Even without any training data, our FFN still works pretty well. We demonstrate the effects of the proposed FFN on various CNN architectures, including AlexNet [1], VGG-16 [19], and ResNet [20], on the ImageNet classification task. Performance on MS COCO object detection benchmark is also investigated. The main contributions can be summarized as follows.

1) A unified FFN framework is proposed for DNN acceleration and compression, which is flexible and accurate.
2) Based on fixed-point factorization, we propose a novel full-precision weight recovery method, which makes it possible to make full use of the pretrained models even for very deep architectures, such as deep residual networks (ResNets).
3) We investigate the weight imbalance problem generally existing in matrix/tensor decomposition-based DNN acceleration methods and propose an effective weight balancing technique to stabilize the fine-tuning process of DNN models.
4) An unsupervised version of FFN is developed to deal with the circumstance where limited unlabeled data or even no data are provided.

A preliminary work of this article has been presented in the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017 [21]. This article differs from the previously published article from the following aspects: 1) we provide more theoretical and experimental analysis on the fixed-point factorization approach and compare FFN with more recent quantization methods; 2) we extend the fully supervised FFN to the unsupervised circumstance, demonstrating the effectiveness of the proposed FFN framework; and 3) experiments on object detection benchmark are also included to demonstrate the generalization ability of FFN to other computer vision tasks.

## II. Related Work

DNNs suffer from huge storage and memory consumption as well as the low speed at inference time. Consequently, a bulk of works [22] have emerged to deal with the acceleration and compression of CNNs with tolerable performance degradation, including but not limited to low-rank decomposition, network pruning, quantization, and knowledge distillation. We also review network compression approaches with limited data.

### A. Low-Rank Decomposition

DNNs are usually overparameterized, and the redundancy can be removed using the low-rank approximation of the filter matrix as shown in the work of [5]. Since then, many low-rank-based methods have been proposed. Jaderberg [23] proposed to use filter low-rank approximation and data reconstruction to lower the approximation error. Zhang *et al.* [6] presented a novel nonlinear data reconstruction method, which allows asymmetric reconstruction to prevent error accumulation across layers. Their method achieves high speedup on the VGG-16 model with a minor increase on top-five error for ImageNet [24] classification. Low-rank tensor decomposition methods, such as CP-decomposition [25], the Tucker decomposition [7], and block term decomposition (BTD) [8], are also investigated and show high speedup and energy reduction.

### B. Network Pruning

Based on the assumption that many parameters in CNNs are unnecessary, network pruning methods are explored to remove unimportant ones to significantly expand the sparsity of deep models, lowering the computation and storage requirement. According to the granularity of pruning, these methods can be divided into two groups: unstructured pruning and structured pruning. Han *et al*. [2] first proposed to prune the deep CNNs in an unstructured way that compressed AlexNet by $9\times$ without a drop in accuracy. To avoid the potential risk of irretrievable network damage, [26] proposed a dynamic network surgery framework that can recover the incorrectly pruned connections. Aiming at working with the existing efficient BLAS libraries for dense matrix operations, structured sparsity methods are also investigated [3]. Luo *et al*. [4] proposed a filter-level sparsity method by utilizing the next layer's feature map to guide filter pruning in the current layer. By adding structured sparsity regularizer, [27] proposed to reduced trivial filters, channels, or even layers. In addition, [28] proposed to leverage the scaling factor of the batch normalization layer to evaluate the importance of the filters for channel pruning.

### C. Quantization

Fixed-point quantization-based methods are also investigated by several recent works [29], [30]. In BinaryConnect (BC), Courbariaux *et al.* [9] proposed to use binary

weights for forward and backward computation while keep a full-precision version of weights for gradients accumulation. Binary weight network (BWN) and XNOR-net were proposed in a more recent work [10], which was among the first ones to evaluate the performance of binarization on large-scale data sets, such as ImageNet [24], and yielded good results. To further compensate for the accuracy loss of binarization, [31] introduced 1-bit CNNs aided with a real-valued shortcut. On the other hand, multibit networks [32]–[34] decomposed a single convolution layer into multiple binary convolutions for higher accuracy. These methods train neural networks from scratch and can barely benefit from pretrained networks. Hwang and Sung [35] found a way by first quantizing pretrained weights using a reduced number of bits, followed by retraining. However, their method achieved good results only for longer bits on small data sets and heavily relied on carefully choosing the step size of quantization using exhaustive search. The scalability on large-scale data sets remained unclear. Besides fixed-point quantization, product quantization is also investigated in the work of [36] and [37] to compress and speed up DNNs at the same time.

### D. Knowledge Distillation

Different from the abovementioned compression and acceleration methods that do not change the original network structure, knowledge distillation aims at transferring the learned knowledge from a teacher network to a student network, which may be not relevant to the teacher network. It is common that the student network is much smaller than the teacher network; thus, the computation and storage can be reduced. Hinton *et al.* [38] proposed to use dark knowledge, i.e., the output of the softmax layer as labels to guide the training of a student network. Later, FitNet was proposed by transferring the knowledge learned from a teacher to a student, which is deeper and thinner than the teacher. Different from [38], FitNet also utilizes the feature maps from the hidden layers for knowledge transfer. Besides feature map approximation, [39] proposed to mimic the attention maps of the intermediate layers, which allows the student to more about the important features. Besides image recognition, knowledge distillation has also shown advantages in object detection [40], [41] and visual tracking [42].

### E. Network Compression With Limited Data

Previous network compression approaches commonly rely on all training data and labels. Recently, many works explore network compression with limited data. Bhardwaj *et al.* [43] proposed a data-independent compression approach by first utilizing a small amount of metadata and the pretrained model to generate many synthetic images and then using these synthetic images for knowledge distillation. Chen *et al.* [44] further proposed to use GAN to generate training samples that are used for knowledge distillation. Different from these two approaches, which generate training images for network compression, Xu *et al.* [45] exploited massive unlabeled data to find new training samples for knowledge distillation. All these approaches utilize knowledge distillation to learn a compact model, which relies on a large amount of time-consuming backpropagation. Network quantization without fine-tuning based on limited data is also studied. He and Cheng [46] proposed the quasi-Lloyd-max and renormalization approach for 4-bit quantization. Banner *et al*. [47] studied posttraining 4-bit quantization for rapid deployment. Bit split is proposed in [48] for accurate posttraining quantization. Nagel *et al*. [49] proposed data-free quantization for MobileNet. These quantization approaches only study network quantization higher than 4 bit.

Unlike previous works, we explore fixed-point factorization on the weight matrix. In the proposed FFN architecture, we directly factorize the weight matrix into a fixed-point format in an end-to-end way. Using only a small amount of unlabeled data, our FFN can achieve negligible accuracy degradation with ternary weights.

## III. Preliminaries

A general DNN is usually composed of multiple fully connected (FC) layers and/or convolutional layers (CONV). FC layers can be treated as $1 \times 1$ CONV layers; thus, we will focus on CONV layers in this article for consistency.

Typically, the parameters of a CONV layer form a 4-D tensor $\mathcal{W} \in \mathbb{R}^{w \times h \times c \times m}$, where $w$ and $h$ represent the kernel width/height and $c$ and $m$ represent the number of input/output feature maps. Thus, a CONV layer consists of $m$ 3-D kernels of size $w \times h \times c$. During the computation of a CONV layer, the convolution can be transformed into matrix multiplication [50]. Especially, the 4-D tensor $\mathcal{W}$ is rearranged into a 2-D matrix $W \in \mathbb{R}^{m \times n}$ (here, $n = w * h * c$), where the $i$th row $\boldsymbol{w}_i$ represents the $i$th kernel. Accordingly, the input feature maps are also rearranged into a 2-D matrix, where each column represents a convolutional volume. More details can be found in [50].

The problem of fixed-point quantization of DNNs is to transform the floating-point weight matrix $W$ and/or input and output feature maps into fixed-point format. More specifically, for weight quantization, the problem is commonly to quantize each kernel vector $\boldsymbol{w}$ into fixed-point vector $\hat{\boldsymbol{w}}$, with an optional floating-point scaling factor $\alpha$

$$\boldsymbol{w} \approx \alpha \hat{\boldsymbol{w}}. \tag{1}$$

By the replacement shown in (1), the quantized model represented by $\alpha$ and $\hat{\boldsymbol{w}}$ can be trained using stochastic gradient descent (SGD). $\alpha$ can be calculated at each step of SGD or fixed during training and can also be trained as a network parameter using SGD. However, due to the discrete nature of the quantized weights, the updates (weight gradients multiplied by the learning rate) to $\hat{\boldsymbol{w}}$ during SGD are too small to make a change.

To solve the abovementioned problem, the full-precision weight vector $\boldsymbol{w}$ is also kept, which is used for the updates accumulation. More specifically, during the forward and backward propagations, the quantized weights $\hat{\boldsymbol{w}}$ are used for the gradient computation, while the updates to the weights are accumulated by the full-precision weights $\boldsymbol{w}$ ($\Delta \boldsymbol{w} = \alpha \Delta \hat{\boldsymbol{w}}$). During the next step of the SGD training process, the quantized

weights $\hat{\boldsymbol{w}}$ are updated by

$$\hat{\boldsymbol{w}} = q(\boldsymbol{w}/\alpha) \tag{2}$$

where, for uniform fixed-point quantization, the function $q$ is commonly a rounding up operation.

## IV. Approaches

The proposed FFN framework exploits weight matrix factorization for network acceleration and compression. Different from previous low-rank-based decomposition methods that use floating-point values for the factorized submatrices, our method aims at fixed-point factorization directly.

The FFN is presented in Section IV-A. For applications with limited unlabeled data, we further propose the unsupervised quantization scheme for FFN in Section IV-B. The convergence and complexity of the proposed FFN framework are analyzed in Sections IV-C and IV-D, respectively.

### A. Fixed-Point Factorized Networks

We will introduce the fixed-point factorization framework in detail. In this section, we mainly talk about the FFN at the fully supervised quantization circumstance, where all training data and labels are given. First, the direct fixed-point factorization method is introduced in Section IV-A1. In order to better fine-tune the quantized model after factorization, the pseudofull-precision weight recovery and weight balancing methods are discussed in detail in Sections IV-A2 and IV-A3, respectively.

*1) Fixed-Point Factorization of Weight Matrices:* Deep networks commonly consist of multiple convolutional and FC layers, which dominate the storage and computation of the networks. The basic computation of both FC layers as well as the convolutional layers is matrix multiplication [50], i.e., the output signal vector $\boldsymbol{s_o}$ is computed as

$$\boldsymbol{s}_o = \phi(W\boldsymbol{s}_i + \boldsymbol{b}) \tag{3}$$

where $\boldsymbol{s}_i$ is the input signal vector and $W \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ are the weight matrix and the bias term, respectively.

The FFN process is conducted on the weight matrix $W$. More precisely, FFN directly factorizes the weight matrix $W \in \mathbb{R}^{m\times n}$ into the weighted sum of the outer products of $k$ vector pairs with only ternary ($+1$, 0, and $-1$) entries, which is referred to as the semidiscrete decomposition (SDD) in the following format:

$$\begin{aligned}\underset{\hat{U},D,\hat{V}}{\text{minimize}} \parallel W - \hat{U}D\hat{V}^T \parallel_F^2 &\triangleq J(D,\hat{U},\hat{V}) \\ &= \underset{\{d_i\},\{\hat{\boldsymbol{u}}_i\},\{\hat{\boldsymbol{v}}_i\}}{\text{minimize}} \parallel W - \sum_i^k d_i\hat{\boldsymbol{u}}_i\hat{\boldsymbol{v}}_i^T \parallel_F^2 \end{aligned} \tag{4}$$

where $\hat{U} \in \{-1,0,+1\}^{m\times k}$, $\hat{V} \in \{-1,0,+1\}^{n\times k}$, and $D \in \mathbb{R}_+^{k\times k}$ is a nonnegative diagonal matrix. Note that throughout this article, we utilize the symbol $k$ to represent the rank of the SDD decomposition.

The advantage of fixed-point factorization over direct quantization is that the approximation accuracy can be guaranteed. By controlling the decomposition rank $k$, the approximation to $W$ can be accurate as possible. (Note that $k$ could be larger than both $m$ and $n$). This also provides a way to choose different $k$'s for different layers according to the redundancy of that layer. Thus, the FFN can be much more flexible and accurate than direct quantization methods.

The ternary constraints in (4) make the optimization of SDD an NP-hard problem. Kolda and O'Leary [51], [52] proposed to obtain an approximate solution greedily. Here, we utilize an iterative optimization procedure for the minimization problem of (4). Without loss of generality, we assume that the $i$th entries are to be optimized with the rest of entries given. By denoting $R_i = W - \sum_{j\neq i} d_j\hat{\boldsymbol{u}}_j\hat{\boldsymbol{v}}_j^T$, the optimization of the $i$th entries becomes

$$\underset{d_i,\hat{\boldsymbol{u}}_i,\hat{\boldsymbol{v}}_i}{\text{minimize}} \| R_i - d_i\hat{\boldsymbol{u}}_i\hat{\boldsymbol{v}}_i^T \|_F^2 \triangleq F(d_i,\hat{\boldsymbol{u}}_i,\hat{\boldsymbol{v}}_i). \tag{5}$$

For representation simplicity, we drop the index of $i$. Expanding (5) gives

$$\begin{aligned} F(d,\hat{\boldsymbol{u}},\hat{\boldsymbol{v}}) &= \|R - d\hat{\boldsymbol{u}}\hat{\boldsymbol{v}}^T\|_F^2 \\ &= \|R\|_F^2 - 2d\hat{\boldsymbol{u}}^T R\hat{\boldsymbol{v}} + d^2\|\hat{\boldsymbol{u}}\|_2^2\|\hat{\boldsymbol{v}}\|_2^2. \end{aligned} \tag{6}$$

By setting $(\partial F/\partial d) = 0$, the optimal value of $d$ is given by

$$d^* = \frac{\hat{\boldsymbol{u}}^T R\hat{\boldsymbol{v}}}{\|\hat{\boldsymbol{u}}\|_2^2\|\hat{\boldsymbol{v}}\|_2^2}. \tag{7}$$

By substituting $d^*$ into (6), we obtain the equivalent optimization problem as follows:

$$\underset{\hat{\boldsymbol{u}},\hat{\boldsymbol{v}}}{\text{maximize}} \frac{(\hat{\boldsymbol{u}}^T R\hat{\boldsymbol{v}})^2}{\|\hat{\boldsymbol{u}}\|_2^2\|\hat{\boldsymbol{v}}\|_2^2}. \tag{8}$$

We use an alternating optimization for the problem of (8), i.e., we fix $\hat{\boldsymbol{v}}$ to optimize $\hat{\boldsymbol{u}}$, then fix $\hat{\boldsymbol{u}}$ to optimize $\hat{\boldsymbol{v}}$, and so on. Solving (8) with fixed $\hat{\boldsymbol{u}}$ or $\hat{\boldsymbol{v}}$ is equivalent to the following optimization problem:

$$\underset{\hat{\boldsymbol{x}}}{\text{maximize}} \frac{(\hat{\boldsymbol{x}}^T\boldsymbol{t})^2}{\|\hat{\boldsymbol{x}}\|_2^2} \tag{9}$$

where $\hat{\boldsymbol{x}}$ could be either $\hat{\boldsymbol{u}}$ or $\hat{\boldsymbol{v}}$. More specifically, $\hat{\boldsymbol{x}} = \hat{\boldsymbol{u}}$ and $\boldsymbol{t} = R\hat{\boldsymbol{v}}$ for the optimization of $\hat{\boldsymbol{u}}$, or $\hat{\boldsymbol{x}} = \hat{\boldsymbol{v}}$ and $\boldsymbol{t} = R^T\hat{\boldsymbol{u}}$ for the optimization of $\hat{\boldsymbol{v}}$.

Assuming that $\hat{\boldsymbol{x}}$ has exactly $s$ nonzeros, the solution is given by

$$\hat{x}_i = \begin{cases} -\text{sign}(t_i), & \text{abs}(t_i) \text{ in the top } s \text{ of abs}(\boldsymbol{t}) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where sign is the sign function and abs is the absolute value function. When $s$ traverses from 0 to the length of $\hat{\boldsymbol{x}}$, we can get the global optimum $\hat{\boldsymbol{x}}^*$ for (9). We summarize the iterative optimization procedure in Algorithm 1.

Given a pretrained model with full-precision weights, we can conduct fixed-point factorization for each layer using Algorithm 1. After fixed-point decomposition, the original full-precision weight matrix $W$ can be replaced by the factorized ones, i.e., $\hat{U}$, $\hat{V}$, and $D$. More formally, each convolutional layer of the pretrained model is replaced by three layers: 1) a ternary convolutional layer with $k$ filters of size $w \times h \times c$; 2) a "channelwise scaling layer," i.e., each of the $k$

**Algorithm 1** Improved SDD Decomposition

**Input:** weight matrix $W \in R^{m\times n}$
**Input:** non-negative integer $k$
**Output:** $\hat{U} \in \{+1, 0, -1\}^{m\times k}$
**Output:** $\hat{V} \in \{+1, 0, -1\}^{n\times k}$
**Output:** diagonal matrix $D \in \mathbb{R}_{+}^{k\times k}$
$d_i \leftarrow 0$ for $i = 1, \cdots, k$
Select $\hat{V} \in \{-1, 0, 1\}^{n\times k}$
**while** outer iteration not converges **do**
  **for** $i = 1, \cdots, k$ **do**
    $R \leftarrow W - \sum_{j\neq i} d_j \hat{\boldsymbol{u}}_j \hat{\boldsymbol{v}}_j^T$
    Set $\hat{\boldsymbol{v}}_i$ to the $i$-th column of $\hat{V}$
    **while** inner iteration not converges **do**
      compute $\hat{\boldsymbol{u}}_i \in \{-1, 0, 1\}^m$ given $\hat{\boldsymbol{v}}_i$ and $R$
      compute $\hat{\boldsymbol{v}}_i \in \{-1, 0, 1\}^n$ given $\hat{\boldsymbol{u}}_i$ and $R$
    **end while**
    Set $d_i$ to the average of $R \circ \hat{\boldsymbol{u}}_i \hat{\boldsymbol{v}}_i^T$ over the non-zero locations of $\hat{\boldsymbol{u}}_i \hat{\boldsymbol{v}}_i^T$
    Set $\hat{\boldsymbol{u}}_i$ as the $i$-th column of $\hat{U}$, $\hat{\boldsymbol{v}}_i$ the $i$-th column of $\hat{V}$ and $d_i$ the $i$-th diagonal value of $D$
  **end for**
**end while**

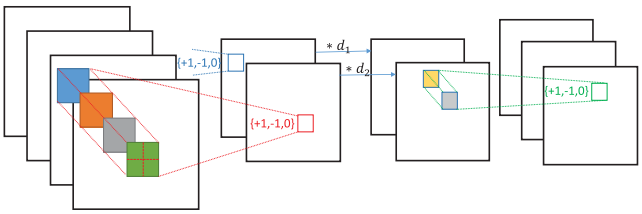



Fig. 1. New layers used in our FFN architecture. After the decomposition of $W \rightarrow \hat{U} D \hat{V}^T$, the original convolution represented by weight matrix $W$ is replaced by three layers, i.e., the first $h \times w$ convolution corresponds to $\hat{V}^T$, followed by the second scaling layer represented by $D$, and the third $1 \times 1$ convolution represented by $\hat{U}$.

feature maps is multiplied by a scaling factor; and 3) another ternary convolutional layer with $m$ filters of size $1 \times 1 \times k$. Fig. 1 shows the architecture of the new layers in the FFN network.

*2) Pseudofull-Precision Weight Recovery:* Retraining or fine-tuning fixed-point networks requires full-precision weights for gradient accumulation (see Section III for more details). However, after factorization, the full-precision weights are discarded, i.e., the original $W$ is replaced by $\hat{U}, \hat{V}$, and $D$ and, thus, cannot be used for gradient accumulation. A simple solution is to use the floating-point version of $\hat{U}$ and $\hat{V}$ as full-precision weights to accumulate gradients. However, this is far from satisfactory, as can be seen from Section V-A2.

To ease the retraining stage after factorization, we propose a novel full-precision weight recovery method based on pretrained weights. By discarding the ternary constraints in (4), we recovery the full-precision version of $\hat{U}$ and $\hat{V}$, indicated by $U$ and $V$, which can better approximate $W$. Note that at each step during the retraining process, the full-precision weights are quantized using a quantization function [(11) in our case] into fixed-point weights that are used for forward/backward computation. Thus, to recover the appropriate full-precision weights, it must be guaranteed that, initially, $U$ and $V$ will be quantized into $\hat{U}$ and $\hat{V}$ using the same quantization function. We turn this problem into an optimization problem as follows:

$$\begin{aligned} &\min_{U,V} \| W - UDV^T \|_F^2 \\ &\text{s.t. } |U_{ij} - \hat{U}_{ij}| < 0.5 \quad \forall i, j \\ &\qquad |V_{ij} - \hat{V}_{ij}| < 0.5 \quad \forall i, j. \end{aligned} \tag{11}$$

Here, the two constraints are introduced to ensure that $U$ and $V$ will be quantized to $\hat{U}$ and $\hat{V}$. The problem can be efficiently solved by the alternative method. During the retraining stage, the full-precision weights $U$ and $V$ are quantized according to the following equation:

$$q(A_{ij}) = \begin{cases} +1, & 0.5 < A_{ij} < 1.5 \\ 0, & -0.5 \leq A_{ij} \leq 0.5 \\ -1, & -1.5 < A_{ij} < -0.5. \end{cases} \tag{12}$$

Note that the full-precision weights $U$ and $V$ are only used for gradient accumulation during the fine-tuning stage, which can be discarded after retraining. Only the quantized weights $\hat{U}$ and $\hat{V}$ are used for prediction. The full-precision weight recovery method can be treated as an inversion of common fixed-point quantization methods. In fixed-point quantization, each floating-point element is quantized into its nearest fixed-point value, while, in our method, we first obtain the fixed-point weights through fixed-point factorization (4), and then, we need to determine from what values the fixed-point elements are quantized.

*3) Weight Balancing:* After weight matrix factorization and full-precision weight recovery, the quantized networks can be retrained using training data. However, it is noticed that the retraining process becomes unstable. We innovatively identify this phenomenon as the weight imbalance problem, which is caused by the nonuniqueness of the decomposition. It is worth noting that weight imbalance is a general problem in decomposition-based methods, such as in [6]. In this section, we thoroughly analyze the cause of this problem and propose a weight balancing approach to solve it.

The forward computation of an $L$-layer network can be represented by the following equations:

$$\begin{aligned} \boldsymbol{z}^{(l+1)} &= W^{(l)} \boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)} \\ \boldsymbol{a}^{(l+1)} &= \phi(\boldsymbol{z}^{(l+1)}). \end{aligned} \tag{13}$$

For backward computation, the error terms of layer $l$ and the corresponding gradients are updated as follows:

$$\boldsymbol{\delta}^{(l)} = ((W^{(l)})^T \boldsymbol{\delta}^{(l+1)}) \bullet \phi'(\boldsymbol{z}^{(l)}) \tag{14}$$

$$\nabla_{W^{(l)}} = \boldsymbol{\delta}^{(l+1)} (\boldsymbol{a}^{(l)})^T \tag{15}$$

where "$\bullet$" denotes the Hadamard product operator. The input, output, and error vectors for layer $l$ are represented by $\boldsymbol{a}^{(l)}$, $\boldsymbol{a}^{(l+1)}$, and $\boldsymbol{\delta}^{(l)}$, respectively.

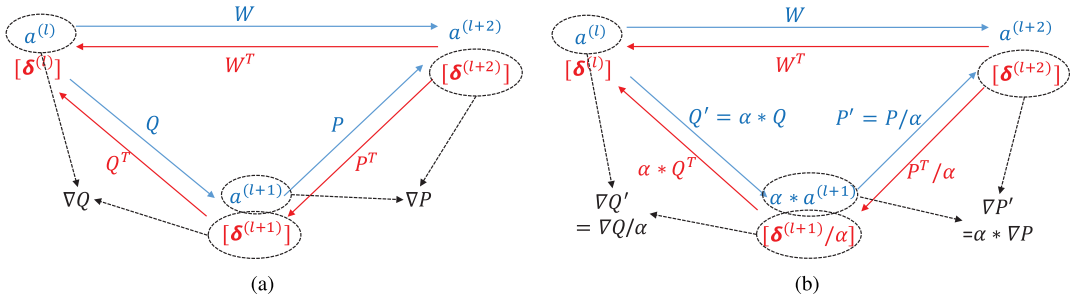



Fig. 2. Illustration of the cause of weight imbalance problem generally existing in decomposition-based methods. The blue lines represent the forward information propagation, while the red lines represent the backward error propagation. The black dashed lines indicate that the gradients of weights are computed from the multiplication of the output error term $\delta$ and input activation $a$. Left: by the decomposition of $W = PQ$, which means that the convolutional layer with weight matrix $W$ is factorized into two layers parameterized by $Q$ and $P$. Right: by setting $P' = P/\alpha$ and $Q' = \alpha * Q$, the decomposition also holds, i.e., $W = P'Q'$. By comparing the weights and gradients of $Q$ and $Q'$, as well as $P$ and $P'$, it can be concluded that the bigger the weights are, the smaller the gradients will become, which makes training unstable. (a) Forward and backward propagations. (b) Propagation after setting $Q' = \alpha * Q$ and $P' = P/\alpha$.

Considering the decomposition of $W = PQ$, which means that the convolutional layer with weight matrix $W$ is factorized into two layers parameterized by $Q$ and $P$, as shown in Fig. 2(a). By setting $P' = P/\alpha$ and $Q' = \alpha * Q$, the decomposition also holds, i.e., $W = P'Q'$, as shown in Fig. 2(b). By comparing the weights and the corresponding gradients of $Q$ and $Q'$, as well as $P$ and $P'$, we can find that the bigger the weights are, the smaller the gradients will become. This will make the training procedure unstable. Supposing $\alpha \gg 1$, during backward computation, $P$ will change drastically, while $Q$ almost stays unchanged. This requires one to set different learning rates for different layers, which is impractical especially for very DNNs.

In the FFN framework, the weight matrix $W \in \mathbb{R}^{m\times n}$ is decomposed into $UDV^T$, where the values of $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{n\times k}$ are in the range of $-1.5\sim1.5$. However, the values of $D$ are much smaller, ranging from 0.00001 to 0.01. Inspired by the weight initialization approach proposed by [53], we develop the weight balancing approach from the following aspects.

First, $U$ and $V$ should be balanced into the appropriate scales by their corresponding scaling factors $\lambda_U$ and $\lambda_V$, which are proportional to the square root of their rows and columns.

Second, the diagonal matrix $D$ that corresponds to the channelwise scaling factors should be close to the identity matrix after balancing. Making $D$ close to the identity matrix will not affect the calculation of gradients. To achieve this, we set the mean value of the diagonal elements to be one.

The abovementioned two objectives can be realized by using (16), where $\tilde{U}$, $\tilde{V}$, and $\tilde{D}$ correspond to the balanced weight matrices. Note that $\varphi$ is introduced to guarantee that the two scaling factors $\lambda_U$ and $\lambda_V$ are in proportion to the square root of the rows and columns of $U$ and $V$

$$\begin{cases} \tilde{U} = \lambda_U * U = \dfrac{\varphi}{\sqrt{m+k}} * U \\ \tilde{V} = \lambda_V * V = \dfrac{\varphi}{\sqrt{n+k}} * V \\ \tilde{D} = \dfrac{D}{\lambda_U * \lambda_V} \\ \text{mean}(\tilde{D}) = 1. \end{cases} \tag{16}$$

After weight balancing, the balanced weights $\tilde{U}$, $\tilde{D}$, and $\tilde{V}$ can be used for retraining, which makes the retraining process more stable.

### B. FFN With Limited Unlabeled Data

In Section IV-A, we have presented the FFN framework under the fully supervised quantization circumstance, where all training data and labels are accessible. However, in many real-world applications, the label information is hard to obtain. Moreover, due to privacy policies, we may have no access to a large amount of data. In this section, we extend our FFN to the unsupervised quantization scheme.

*1) Response Matrix Factorization:* In this section, we will extend our FFN from weight matrix factorization to response matrix factorization as follows:

$$\begin{aligned} \min_{D,\hat{U},\hat{V}} \ & \| Y - \hat{U}D\hat{V}^T X \|_F^2 \\ \text{s.t. } & \hat{U} \in \{-1, 0, +1\}^{m\times k} \\ & \hat{V} \in \{-1, 0, +1\}^{n\times k} \\ & D \in \mathbb{R}_+^{k\times k} \end{aligned} \tag{17}$$

where $Y \in \mathbb{R}^{m\times t}$ represents the output response matrix of a given convolutional layer, and $X \in \mathbb{R}^{n\times t}$ represents the corresponding input matrix of the pretrained model. Here, $t$ represents the number of response vectors. Formally, we have $Y = WX$. Note that we only need to sample some responses for the optimization problem of (17).

Similar to fixed-point weight matrix factorization, we utilize an iterative greedy optimization procedure. At each step $i$, we solve the following problem:

$$\begin{aligned} \min_{d_i,\hat{\boldsymbol{u}}_i,\hat{\boldsymbol{v}}_i} \ & \| E_i - d_i\hat{\boldsymbol{u}}_i\hat{\boldsymbol{v}}_i^T X \|_F^2 \\ \text{s.t. } & \hat{\boldsymbol{u}}_i \in \{-1, 0, +1\}^m \\ & \hat{\boldsymbol{v}}_i \in \{-1, 0, +1\}^n \\ & d_i \in \mathbb{R}_+ \end{aligned} \tag{18}$$

where $E_i = Y - \sum_{j\neq i} d_j\hat{\boldsymbol{u}}_j\hat{\boldsymbol{v}}_j^T X$ represents the residual to be minimized at step $i$. For simplicity, we will discard the index $i$,

and the abovementioned optimization problem becomes

$$\underset{d,\hat{\boldsymbol{u}},\hat{\boldsymbol{v}}}{\text{minimize}} \| E - d\hat{\boldsymbol{u}}\hat{\boldsymbol{v}}^T X \|_F^2 \triangleq H(d, \hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}). \tag{19}$$

To minimize (19), we utilize an alternating solver to optimize $d$, $\hat{\boldsymbol{u}}$, and $\hat{\boldsymbol{v}}$.

*a) Subproblem of $d$:* In this case, $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$ are fixed. The minimization problem of (19) turns into the minimization of the following problem:

$$H(d) = \| E \|_F^2 - 2d\hat{\boldsymbol{v}}^T X E^T \hat{\boldsymbol{u}} + d^2 \| \hat{\boldsymbol{u}}\hat{\boldsymbol{v}}^T X \|_F^2. \tag{20}$$

Equation (20) has only one variable, and it is easy to get that

$$d = \frac{\hat{\boldsymbol{v}}^T X E^T \hat{\boldsymbol{u}}}{\| \hat{\boldsymbol{u}}\hat{\boldsymbol{v}}^T X \|_F^2}. \tag{21}$$

*b) Subproblem of $\hat{\boldsymbol{u}}$:* In this case, $d$ and $\hat{\boldsymbol{v}}$ are fixed. The minimization problem of (19) turns into the minimization of the following problem:

$$\begin{aligned} H(\hat{\boldsymbol{u}}) &= \| E \|_F^2 - 2d\hat{\boldsymbol{v}}^T X E^T \hat{\boldsymbol{u}} + d^2 \hat{\boldsymbol{v}}^T X X^T \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}} \\ &= \gamma\, \hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}} + \boldsymbol{t}^T \hat{\boldsymbol{u}} + \text{const} \end{aligned} \tag{22}$$

where $\gamma = d^2 \hat{\boldsymbol{v}}^T X X^T \hat{\boldsymbol{v}}$ and $\boldsymbol{t} = -2dEX^T \hat{\boldsymbol{v}}$.

The integer programming of (22) has $3^m$ feasible points; thus, an exhaustive search method is impractical to use. However, $\hat{\boldsymbol{u}} \in \{-1, 0, 1\}^m$; thus, the term $\hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}}$ equals to the number of nonzeros of $\hat{\boldsymbol{u}}$. In this way, the minimization of (22) can be solved by checking $m$ possibilities. Assuming that $\hat{\boldsymbol{u}}$ has exactly $s$ nonzeros, the solution is given by

$$\hat{u}_i = \begin{cases} -\text{sign}(t_i), & \text{abs}(t_i) \text{ in the top } s \text{ of abs}(\boldsymbol{t}) \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

where sign is the sign function and abs is the absolute value function. When $s$ traverses from 0 to $m$, we can get the global optimum $\hat{\boldsymbol{u}}^*$ for (22).

*c) Subproblem of $\hat{\boldsymbol{v}}$:* In this case, $d$ and $\hat{\boldsymbol{u}}$ are fixed. The minimization problem of (19) turns into the minimization of the following problem:

$$H(\hat{\boldsymbol{v}}) = \| E \|_F^2 - 2d\hat{\boldsymbol{u}}^T E X^T \hat{\boldsymbol{v}} + d^2 \hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}} \| X^T \hat{\boldsymbol{v}} \|_F^2. \tag{24}$$

It is not easy to minimize (24). Here, we utilize the cyclic coordinate descent algorithm to obtain an approximate solution. More specifically, during each descent step, we only update one element of $\hat{\boldsymbol{v}}$ but fix all the other elements. Assuming that $\hat{v}_j$ is unknown, the quadratic form of (24) can be transformed into the following formulation:

$$\| X^T \hat{\boldsymbol{v}} \|_F^2 = \| \hat{v}_j X_j^T + \bar{X}_j^T \bar{\hat{\boldsymbol{v}}}_j \|_F^2 \tag{25}$$

where $\hat{v}_j$ is the $j$th element of $\hat{\boldsymbol{v}}$, and $\bar{\hat{\boldsymbol{v}}}_j$ represents the other elements excluding $\hat{v}_j$. Similarly, $X_j^T$ is the $j$th column of $X^T$, and $\bar{X}_j^T$ represents the other columns excluding $X_j^T$. To optimize $\hat{v}_j$ and by setting $\boldsymbol{t} = 2dXE^T \hat{\boldsymbol{u}}$ and $\boldsymbol{s} = \bar{X}_j^T \bar{\hat{\boldsymbol{v}}}_j$, (24) can be written as

$$\begin{aligned} H(\hat{v}_j) &= \| E \|_F^2 - 2d\boldsymbol{t}^T \hat{\boldsymbol{v}} + d^2 \hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}} \| \hat{v}_j X_j^T + \bar{X}_j^T \bar{\hat{\boldsymbol{v}}}_j \|_F^2 \\ &= \gamma\, \hat{v}_j^2 + \eta \hat{v}_j + \text{const} \end{aligned} \tag{26}$$

where $\gamma = d^2 \hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}} X_j^T X_j$ and $\eta = (2d^2 \hat{\boldsymbol{u}}^T \hat{\boldsymbol{u}} X_j^T \boldsymbol{s} - t_j)$. $t_j$ represents the $j$th element of $\boldsymbol{t}$. Note that $\hat{v}_j \in \{-1, 0, 1\}$, and thus, the optimal value of $\hat{v}_j$ is given by

$$\hat{v}_j = \begin{cases} -\text{sign}(\eta), & \gamma - \text{abs}(\eta) < 0 \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

where sign is the sign function and abs is the absolute value function. By iteratively optimize $\hat{v}_j$ for $j = 1, \ldots, n$, we can get an approximate solution $\hat{\boldsymbol{v}}^*$ to (24).

*2) Response Factorization With Error Correction:* So far, we have presented our response matrix factorization framework for unsupervised quantization circumstances. From (17), it can be seen that for a given network, each layer is processed independently. This quantization scheme could result in accumulated errors when the whole network is quantized.

To solve this problem, we make a change to the optimization problem of (17) as follows:

$$\begin{aligned} \min_{D,\hat{U},\hat{V}} \ & \| Y - \hat{U} D \hat{V}^T \hat{X} \|_F^2 \\ \text{s.t. } & \hat{U} \in \{-1, 0, +1\}^{m \times k} \\ & \hat{V} \in \{-1, 0, +1\}^{n \times k} \\ & D \in \mathbb{R}_+^{k \times k} \end{aligned} \tag{28}$$

where $\hat{X}$ represents the quantized inputs (i.e., the outputs from the previous layer of the quantized model) to the layer that is currently processing. Using this quantization scheme, the approximation error of the previous layer can be absorbed during the quantization of the current layer, thus preventing the error from accumulating across layers.

### C. Convergence Analysis

In [52], it has been proved that the SDD algorithm converges linearly to the original matrix with respect to rank $k$. The convergence also holds for the improved SDD algorithm. In this section, we further show that the improved SDD decomposition of Algorithm 1 also converges in terms of the cost function ($J$) with fixed rank $k$.

*1) Convergence of Inner Iteration With Respect to $F$:* The inner iteration of Algorithm 1 solves the minimization problem of $F$ (5). According to the definition, $F$ measures the sum of squared distances between the target matrix $R_i$ and the SDD component $d_i \hat{\boldsymbol{u}}_i \hat{\boldsymbol{v}}_i^T$. Thus, $F$ is bounded from below ($F \geq 0$). The inner iterations of Algorithm 1 is exactly coordinate descent on $F$, in which each coordinate descent step has an optimal solution. Thus, $F$ will monotonically decrease. By now, we have demonstrated that $F$ is bounded from below, and each inner iteration will monotonically reduce $F$. According to the convergence of a bounded monotonic sequence, the value of $F$ will converge.

*2) Convergence of Outer Iteration:* The outer iteration of Algorithm 1 solves the minimization problem of $J$ (4) with respect to $D$, $\hat{U}$, and $\hat{V}$. According to the definition, $J$ measures the sum of squared distances between the original matrix $W$ and the SDD decomposition $\hat{U} D \hat{V}^T$. Thus, $J$ is bounded from below ($J \geq 0$). We will prove that $J$ will monotonically decrease.

Algorithm 1 decomposes the minimization problem of $J$ with respect to $D$, $\hat{U}$, and $\hat{V}$ into $k$ steps, by minimizing $J$ with respect to $\{(d_i, \hat{\boldsymbol{u}}_i, \hat{\boldsymbol{v}}_i); i = 1, \ldots, k\}$. To demonstrate that the outer iteration will reduce the value of $J$, we instead by demonstrating that each of the $k$ steps will reduce the value of $J$. At the $i$th step, only the $i$th SDD components will be updated. We denote the $i$th SDD components after update by $(d_i', \hat{\boldsymbol{u}}_i', \hat{\boldsymbol{v}}_i')$. Similarly, the value of $J$ after the $i$th step is denoted by $J'$. Then, we have

$$\begin{aligned} J' &= \| \left( W - \sum_{j \neq i} d_j \hat{\boldsymbol{u}}_j \hat{\boldsymbol{v}}_j^T \right) - d_i' \hat{\boldsymbol{u}}_i' \hat{\boldsymbol{v}}_i'^T \|_F^2 \\ &= \| R_i - d_i' \hat{\boldsymbol{u}}_i' \hat{\boldsymbol{v}}_i'^T \|_F^2 \\ &\leq \| R_i - d_i \hat{\boldsymbol{u}}_i \hat{\boldsymbol{v}}_i^T \|_F^2 \\ &= \| \left( W - \sum_{j \neq i} d_j \hat{\boldsymbol{u}}_j \hat{\boldsymbol{v}}_j^T \right) - d_i \hat{\boldsymbol{u}}_i \hat{\boldsymbol{v}}_i^T \|_F^2 = J. \end{aligned} \tag{29}$$

By now, we have demonstrated that $J$ is bounded from below, and each outer iteration will monotonically reduce $J$. According to the convergence of a bounded monotonic sequence, the value of $J$ will converge.

### D. Complexity Analysis

The computing complexity of the proposed FFN framework is thoroughly analyzed in this section. The analysis is based on convolutional layers, which dominates most of the operations in a convolutional network.

For a convolutional layer with kernels of size $w \times h \times c \times n$, the operations of multiplication and addition are given by

$$C_{\text{mul}} = C_{\text{add}} = W' * H' * (w * h * c * n) \tag{30}$$

where $W'$ and $H'$ represent the height and width of the output feature maps. By comparison, the operations required by FFN architecture are

$$\begin{aligned} C_{\text{mul}} &= W' * H' * k \\ C_{\text{add}} &= (1 - \alpha) * W' * H' * (w * h * c + n) * k \\ &\approx (1 - \alpha) * W' * H' * (w * h * c * n) \end{aligned} \tag{31}$$

where $\alpha$ denotes the proportion of zeros after fixed-point factorization. For convolutional layers, $c$, $n$, and $k$ are usually in the same order of magnitude. Thus, the multiply operations of FFN can be dramatically reduced because $w * h * c * n \gg k$. At the same time, the addition operations required by FFN can also be reduced by about $(1 - \alpha)$ times. As can be seen from the experiments sections, the sparsity $\alpha$ after quantization is around 0.5, which means that only about half of operations are required, compared with previous binary quantization methods. We refer to Section V-E for more detail.

## V. Experiments

In this section, we comprehensively evaluate the proposed method on ILSVRC-12 [24] image classification and MS COOC object detection benchmarks. We first examine the effects of each individual component of FFN, i.e., fixed-point factorization, full-precision weight recovery, and weight balancing, in Section V-A. Then, in Section V-B, the performance of FFN with fully supervised quantization scheme is evaluated based on AlexNet [1], VGG-16 [19], and ResNet [20]. Next, in Section V-C, we evaluate the performance of the proposed FFN under the circumstance when very limited unlabeled or even no data are provided. We also evaluate the FFN framework on object detection task in Section V-D. Finally, the experimental robustness and efficiency analysis are given in Section V-E.

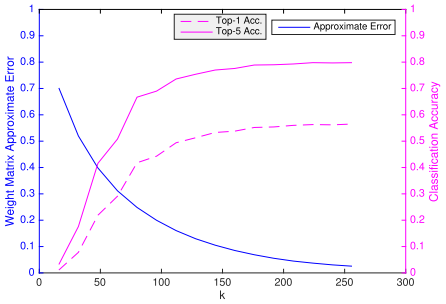



Fig. 3. Weight approximation error and classification accuracy on ImageNet when choosing different $k$'s for the second convolutional layer of AlexNet.

### A. Effectiveness of Each Part

In this section, the effectiveness of each part of our unified FFN framework is thoroughly analyzed by controlled experiments. For fast evaluation, we only utilize the relatively smaller AlexNet [1] model for experiments.

*1) Fixed-Point Factorization:* In Section IV-A1, we have demonstrated that the FFN can infinitely approximate the full-precision weight matrix $W$ by choosing large decomposition rank $k$. It also allows us to utilize different $k$'s for different layers, making the FFN framework much more accurate and flexible than direct quantization methods. In this section, we experimentally evaluate the relationship between the classification accuracy and the weight approximation error under different $k$'s. Here, the weight matrix approximate error is defined as

$$r = \frac{\| W - \hat{U} D \hat{V}^T \|_F^2}{\| W \|_F^2}. \tag{32}$$

Experiments are based on the second convolutional layer of AlexNet, which dominates most of the operations during the inference phase. This layer consists of two groups. The kernel size for each group is $5 \times 5 \times 48 \times 128$. We choose the same decomposition rank $k$ for these two groups. The approximation error is averaged for evaluation.

Fig. 3 shows the relationship between the approximation error (the blue line) and the accuracy (the pink lines) on the ImageNet classification task, for varying $k$. Fig. 3 shows that as $k$ increases, the approximation error decreases to zero, and the accuracy approaches the original AlexNet. The results indicate that the proposed FFN framework can be accurate enough as we want, as long as we use large enough $k$. However, using too large $k$ may also increase the storage requirement and computing time. An appropriate way is to choose $k$ where the

TABLE I
RESULTS OF DIFFERENT SETTINGS ON ALEXNET

| Model | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|
| AlexNet [1] | 57.1 | 80.2 |
| FFN-SDD | 32.9 | 57.0 |
| FFN-Recovered | 57.0 | 80.1 |
| FFN-W/O-FWR | 53.4 | 77.2 |
| FFN-W/O-WB | 51.9 | 76.6 |
| FFN | 55.5 | 79.0 |

marginal gain becomes inconspicuous. In Fig. 3, we notice that when $k$ is larger than 128, the accuracy increment becomes subtle, i.e., setting the rank similar to the number of output channels could be an appropriate choice.

The classification accuracy after all layers are quantized is given in the second row of Table I (denoted as FFN-SDD). By comparing with the original AlexNet model, it can be seen that the accuracy of FFN drops drastically without retraining. Thus, a later fine-tune stage is necessary for higher accuracy. However, our fixed-point factorized method can indeed provide a reasonable initialization.

*2) Full-Precision Weight Recovery:* In this section, we evaluate the effect of the proposed full-precision weight recovery method. During the retraining stage, full-precision weights are used for gradients accumulation. Thus, the retraining procedure could be affected by the initial values of the full-precision weights. We first evaluate the performance of the recovered weights on AlexNet to demonstrate that the recovered weights can actually represent the original weights. The accuracy is shown in the third row of Table I (FFN-Recovered), which is very close to that of the original AlexNet model.

To further demonstrate the effect of the weight recovery strategy on the retraining procedure, we compare the accuracy of FFN with or without full-precision weight recovery in Table I. Without full-precision weight recovery (FFN-W/O-FWR), the top-five accuracy decreases by about 1.8%.

*3) Weight Balancing:* Weight balancing can make the retraining process more stable, which is demonstrated in Table I. We compare the results of using weight balancing (FFN) and that without weight balancing (FFN-W/O-WB). The results on AlexNet show that the weight balancing technique greatly helps the retraining, leading to a 3.6%/2.4% improvement in the top-one/top-five classification accuracy.

To further understand the gradients imbalance problem and how the weight balancing method works, we investigate the gradients' distribution of the second layer of AlexNet during the retraining stage, as shown in Fig. 4. The three rows from top to bottom correspond to the gradient distribution of $\hat{V}$, $D$, and $\hat{U}$, respectively. The left and right columns represent the gradient distribution before and after applying our weight balancing method.

From the left column of Fig. 4, we discover that without weight balancing, the gradient distribution of the three decomposed layers differs significantly from each other. The gradients to $D$ could be thousands of times larger than the gradients to $\hat{U}$ and $\hat{V}$. Moreover, the gradients to $\hat{U}$ tend to be several times larger than those of $\hat{V}$. The experimental results

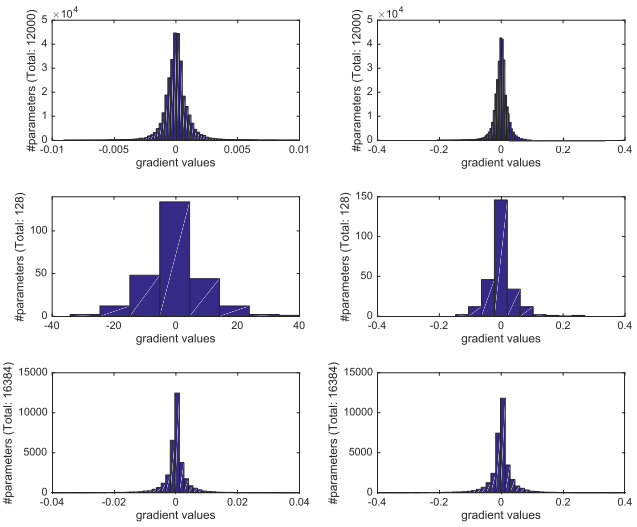



Fig. 4. Gradient distribution of the second convolutional layer of AlexNet before (left column) and after (right column) weight balancing. Three rows correspond to $\hat{V}$, $D$, and $\hat{U}$, respectively.

are consistent with the analysis in Section IV-A3 and Fig. 2, i.e., after decomposition, the matrix with larger elements tends to have smaller gradients, and the matrix with more elements also tends to have smaller gradients.

While after weight balancing (the right column of Fig. 4), most gradients lie between $-0.1$ and 0.1 for all layers. The weight balancing technique allows us to use the same learning rate for all layers, which is very important for network retraining, especially for very deep networks, such as ResNet.

### B. Supervised Experiments on ILSVRC-12

In this section, the performance of the proposed FFN framework on the ImageNet classification task is evaluated under a fully supervised quantization scheme. The quantization procedure is first to approximate the original weight matrices using the proposed fixed-point factorization, full-precision weight recovery, and weight balancing method. Then, the quantized networks are retrained on the ImageNet classification data set to retain accuracy. Most of the commonly used CNN models, such as AlexNet [1], VGG-16 [19], ResNet-50 [20], and ResNet-101 [20], are used for evaluation.

*1) AlexNet:* We first conduct experiments on AlexNet [1]. This network has about 61M parameters, most of which reside in the last three FC layers. To achieve a higher compression rate, we utilize smaller decomposition rank $k$ for the FC layers. More specifically, for the convolutional layers with kernels in the shape of $w \times h \times c \times n$, we choose decomposition rank $k = \min(w * h * c, n)$, while, for the last three FC layers, $k$ is set to be 2048, 3072, and 1000, respectively.

For comparison, the performance of previous quantization methods is also given. Note that previous methods, such as BC [9] and BWN [10], report their results with batch normalization [54]. For fair comparison, we also report our results with batch normalization using the same settings.

1) *BC [9]:* Using binary weights, reported by [10].
2) *BWN [10]:* Using binary weights and floating-point scaling factors.

TABLE II
COMPARISON ON ALEXNET (BATCH NORMALIZATION [54] IS USED)

| Model | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|
| AlexNet-BN [59] | 60.1 | 81.9 |
| BC [9] | 35.4 | 61.0 |
| BWN [10] | 56.8 | 79.4 |
| BWNH [15] | 58.5 | 80.9 |
| TWN [17] | 54.5 | 76.8 |
| TTQ [18] | 57.5 | 79.7 |
| TWN-ADMM [55] | 58.2 | 80.6 |
| LDR [14] | - | 75.1 |
| FFN | 60.6 | 81.9 |

TABLE III
COMPARISON ON VGG-16

| Model | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|
| VGG-16 [19] | 71.1 | 89.9 |
| LDR [14] | - | 89.0 |
| FFN | 70.8 | 90.1 |

3) *BWN via Hashing (BWNH) [15]:* Using binary weights and floating-point scaling factors.
4) *Ternary Weight Network (TWN) [17]:* Using ternary weights and floating-point scaling factors.
5) *Trained Ternary Quantization (TTQ) [18]:* Using ternary weights and asymmetric floating-point scaling factors.
6) *TWN-ADMM [55]:* TWN trained using alternating direction method of multipliers (ADMM).
7) *LDR [14]:* Logarithmic data representation, 4-bit logarithmic activation, and 5-bit logarithmic weights.
8) *APPRENTICE [56]:* TWN via knowledge distillation.
9) *TGA [57]:* TWN by simultaneously optimizing weight and quantizer.
10) *QNet [58]:* TWN using differentiable nonlinear function.

The results are shown in Table II. The suffix $BN$ indicates that batch normalization [54] is used. From the results, we can see that under a fully supervised quantization scheme, the accuracy of the quantized networks is very close to that of the full-precision counterparts. For the top-one accuracy of AlexNet, our method (denoted by FFN) outperforms the previous best result by 2.1%. These results show that FFN can achieve comparable accuracy to full-precision baselines and dramatically outperforms current state-of-the-art methods.

*2) VGG-16:* The VGG-16 [19] model consists of 13 convolutional layers and three FC layers, which is much wider and deeper than AlexNet. It is also more challenging to quantize the VGG-16 model. We use the same policy for choosing the rank $k$ as in Section V-B1, and we set $k = 3138, 3072$, and 1000 for three FC layers, respectively, resulting in about the same number of parameters as the original VGG-16 model.

The quantization results as well as the baselines are shown in Table III. It is easy to conclude that after ternary quantization, our method even outperforms the full-precision VGG-16 model by 0.2% on top-five accuracy. The results demonstrate the effectiveness of the proposed FFN framework.

*3) ResNets:* To further demonstrate the advantages of the proposed FFN framework, we conduct experiments on the challenging ResNet architecture, i.e., ResNet-50 and ResNet-101. Unlike AlexNet and VGG-16, the ResNet is much harder to quantize because it has taken computation and storage into consideration during the design of network architecture. This is achieved by utilizing bottleneck architecture [20], which dramatically reduces the redundancy.

TABLE IV
COMPARISON ON RESNET-50 AND RESNET-101

| Model | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|
| ResNet-50 [20] | 75.4 | 92.6 |
| FFN-ResNet-50* | 72.7 | 90.9 |
| FFN-ResNet-50 | 73.8 | 91.7 |
| ResNet-101 [20] | 77.0 | 93.5 |
| FFN-ResNet-101 | 75.6 | 92.8 |

TABLE V
COMPARISON WITH CURRENT STATE-OF-THE-ART METHODS ON RESNET-50 CLASSIFICATION TASK

| Method | | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|---|
| APPRENTICE [56] | Full-precision | 76.2 | - |
| | Quantization | 73.9 | - |
| | Drop | -2.3 | - |
| TGA [57] | Full-precision | 76.1 | 92.9 |
| | Quantization | 74.0 | 91.7 |
| | Drop | -2.1 | -1.2 |
| QNet [58] | Full-precision | 76.4 | 93.2 |
| | Quantization | 75.2 | 92.6 |
| | Drop | -1.2 | -0.6 |
| FFN | Full-precision | 75.4 | 92.6 |
| | Quantization | 73.8 | 91.7 |
| | Drop | -1.6 | -0.9 |

The ResNet architecture only has one FC layer. There is a global average pooling layer before the FC layer; thus, the ResNet has much fewer parameters than AlexNet [1] and VGG-16 [19]. To make the number of parameters unchanged, we set $k = ((w * h * c) * n/w * h * c + n)$ for all layers, i.e., keeping the same number of parameters for every layer. In this setting, our method still achieves promising results. As shown in Table IV, the top-five accuracy of the proposed FFN (denoted by FFN-ResNet-50*) drops by 1.7%.

Choosing a higher $k$ for convolutional layers as done for AlexNet and VGG-16 could further reduce the classification error. More specifically, for the convolutional layers with 4-D weights of size $w \times h \times c \times n$, we choose decomposition dimension $k = \min(w * h * c, n)$. The results are shown in Table IV (denoted by FFN-ResNet-50 and FFN-ResNet-101). It is clear that choosing a higher $k$ can remarkably boost the accuracy, i.e., for the ResNet-50, the top-five accuracy can be improved from 90.9% to 91.7%. The improvement also illustrates the flexibility of the proposed FFN method, i.e., for networks with less redundancy, a relatively higher $k$ could be utilized to guarantee the classification accuracy. In this way, our FFN achieves only 0.9% and 0.7% top-five accuracy drops for ResNet50 and ResNet101, which shows that the performance of FFN is comparable to the full-precision networks.

To further compare FFN with the current state of the art, we report the accuracy before and after quantization based on ResNet-50 in Table V. From the results, we can see that under a supervised quantization scheme, the proposed

**Algorithm 2** Response Factorization With Error Correction

**Input:** Pretrained networks with weight matrices $\{W^l\}_{l=1}^L$
**Input:** A batch of unlabeled data $X^0$
**Output:** Ternary weight matrices $\{\hat{U}^l\}_{l=1}^L$ and $\{\hat{V}^l\}_{l=1}^L$
**Output:** Diagonal matrices $\{D^l\}_{l=1}^L$

```
Init D^l, Û^l and V̂^l using Algorithm 1
Forward propagation of pretrained model to get Y^l
Forward propagation of quantized model to get X̂^{l-1}
for l = 1, ··· , L do
  while outer iteration not converges do
    for i = 1, ··· , k do
      E ← Y^l − Σ_{j≠i} d_j û_j v̂_j^T X̂^l
      while inner iteration not converge do
        Update d_i given û_i and v̂_i according to (21)
        Update û_i given d_i and v̂_i according to (23)
        for j = 1, ··· , n do
          Update the jth element of v̂_i given d_i and û_i
          according to (27)
        end for
      end while
    end for
  end while
end for
```

TABLE VI
UNSUPERVISED QUANTIZATION RESULTS OF VARIOUS MODELS ON IMAGENET. ALL FFN MODELS ARE QUANTIZED INTO TERNARY WEIGHTS. THE SUFFIX "INT8" INDICATES 8-BIT FIXED-POINT QUANTIZATION OF ACTIVATIONS, IN ADDITION TO TERNARY WEIGHT QUANTIZATION

| Model | | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|---|
| AlexNet | Full-precision | 60.1 | 81.9 |
| | FFN | 58.8 | 80.4 |
| | FFN-Int8 | 58.6 | 80.2 |
| VGG-16 | Full-precision | 72.4 | 91.1 |
| | FFN | 69.9 | 89.6 |
| | FFN-Int8 | 69.8 | 89.5 |
| ResNet-50 | Full-precision | 75.4 | 92.6 |
| | FFN | 72.6 | 91.0 |
| | FFN-Int8 | 72.3 | 90.6 |
| ResNet-101 | Full-precision | 77.0 | 93.5 |
| | FFN | 72.4 | 91.2 |
| | FFN-Int8 | 72.1 | 90.7 |

FFN can achieve comparable or higher accuracy than current state-of-the-art approaches.

### C. Unsupervised Experiments on ILSVRC-12

In this section, we thoroughly evaluate the performance of FFN on the ImageNet classification task under an unsupervised quantization scheme, where only limited unlabeled data are provided, or even no data are provided. We use the same evaluation approach as in Section V-B.

*1) Unsupervised Quantization Results:* In this section, we evaluate the performance of the unsupervised FFN approach, i.e., the proposed response factorization with the error correction method, as illustrated in Algorithm 2. No fine-tuning is conducted after quantization.

Table VI compares our unsupervised ternary quantization methods with full-precision counterparts on various models.

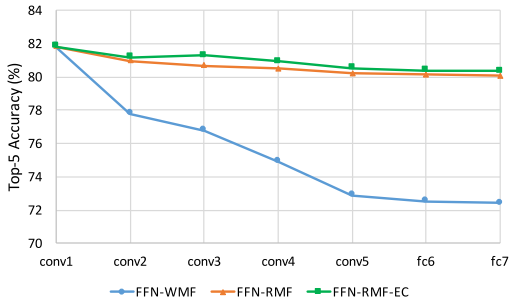



Fig. 5. Accuracy curve of AlexNet on ImageNet after layerwise fixed-point factorization.

Without fine-tuning, our FFN achieves results comparable to the full-precision models. The top-five accuracy drops 1.5%, 1.7%, 1.2%, and 2.3% on AlexNet, VGG-16, ResNet-50, and ResNet-101, respectively. The degradation is acceptable considering that no label and fine-tuning are needed.

By comparing the results to current state-of-the-art methods shown in Table II, it can be seen that our unsupervised FFN outperforms most of the current fully supervised quantization methods.

In order to eliminate the floating-point operations, we further quantize all activations into a fixed-point format using 8-bit quantization. The results are also given in Table VI. The suffix "Int8" indicates 8-bit fixed-point quantization of activations. From the results, it can be concluded that the 8-bit activation quantization has little impact on the accuracy.

*2) Error Correction:* To further show the effect of the proposed response factorization as well as error correction, we have conducted extensive experiments based on the AlexNet model. All layers except the first one are quantized sequentially. The results are shown in Fig. 5, which shows the classification accuracy versus the number of quantized layers. We summarize the controlled experiments as follows.

1) *FFN-WMF:* FFN using weight matrix factorization.
2) *FFN-RMF:* FFN using response matrix factorization without error correction.
3) *FFN-RMF-EC:* FFN using response matrix factorization with error correction.

From Fig. 5, we can see that FFN with weight matrix factorization alone can achieve about 72.4% top-five accuracy, which shows the powerful reconstruction ability of the proposed fixed-point factorization method. However, degradation is still not acceptable.

To demonstrate the advantages of response factorization over the weight matrix factorization, we give the results of FFN using response matrix factorization without error correction, where multilayers are processed simultaneously. With the help of response factorization, FFN-RMF consistently outperforms FFN-WMF by a large margin for all layers.

We notice that when coupled with error correction, the FFN-RMF-EC could outperform FFN-WMF by more than 0.3%, which indicates that the accumulative error due to multilayer approximation can be reduced by the error correction mechanism.

TABLE VII

IMAGENET CLASSIFICATION ACCURACY OF ALEXNET AND VGG16 AFTER RESPONSE MATRIX FACTORIZATION BASED ON DIFFERENT CALIBRATION DATA

| Model | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|
| AlexNet-BN | 60.1 | 81.9 |
| FFN-ImageNet | 58.8 | 80.4 |
| FFN-ImageNet-TestSet | 58.8 | 80.3 |
| FFN-VOC | 58.6 | 80.1 |
| VGG16 | 72.4 | 91.1 |
| FFN-ImageNet | 69.9 | 89.6 |
| FFN-ImageNet-TestSet | 69.8 | 89.4 |
| FFN-VOC | 68.3 | 88.5 |

*3) FFN Without Training Data:* So far, we have demonstrated the effects of the unsupervised FFN approach with unlabeled data. Furthermore, we apply our method under circumstances where no training data are available.

Note that as indicated in (17), our response matrix factorization method actually learns a mapping between the inputs and the outputs while ignoring the real content. Thus, instead of conducting fixed-point factorization based on target images, we could use other heterogeneous images that are publicly available, for example, images collected from the website. Here, we evaluate the unsupervised FFN on the ImageNet classification task; however, no ImageNet data are used during fixed-point factorization. Instead, we consider two cases. First, a small number of unlabeled images, which come from similar distribution as training data, are available. For this case, we use images from the test set of ImageNet for experiments, denoted by FFN-ImageNet-TestSet. Second, no training images or similar images are available. We use images from the Pascal VOC data set [60] for the factorization learning process, denoted by FFN-VOC. The evaluation results based on AlexNet and VGG16 are reported in Table VII.

From Table VII, we can see that using images from training data set or from similar distribution as the training data set could achieve very similar accuracy performance. Using images from very different distributions as training data results in a bit accuracy degradation, however, the accuracy is still comparable to that using original training data. Thus, we can conclude that even though no data are available, the proposed FFN still works well.

### D. Experiments on Object Detection

In Sections V-B and V-C, we have thoroughly evaluated the proposed method on the image classification task. It is worth noting that our method could be readily used to other computer vision tasks, which commonly involves DNNs. In this section, we evaluate the performance of the proposed unsupervised FFN on object detection and instance segmentation tasks.

We use Mask RCNN [61] for demonstration, which is current state-of-the-art object detection and instance segmentation approach. The challenging MS COCO data set is used for evaluation. The model is trained on 80k training images and 35k of validation images (trainval35k) and is evaluated on the remaining 5k validation images (minival). Input images are resized to 800 pixels in the shorter edge. We choose the ResNet-18 architecture as the backbone, which is quantized using the proposed FFN approach. Note that only the backbone is ternarized using the proposed unsupervised quantization scheme. The results are shown in Table VIII. It can be concluded that there are 2.8 points and 2.7 points degradation in box AP and mask AP, respectively, which demonstrates the generalization ability of the proposed FFN framework. It is worth noting that the accuracy could be further improved if we fine-tune the factorized networks using labels. The results on Mask RCNN have shown that the proposed FFN also works on object detection tasks and more fine-grained instance segmentation tasks.

TABLE VIII

OBJECT DETECTION (BOUNDING BOX AP) AND INSTANCE SEGMENTATION (MASK AP) RESULTS ON COCO MINIVAL SET

| Model | | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| Box | Full-precision | 33.1 | 54.3 | 35.2 | 18.2 | 35.7 | 43.7 |
| | FFN | 30.3 | 50.1 | 31.9 | 16.0 | 33.1 | 40.1 |
| Mask | Full-precision | 30.7 | 51.2 | 32.4 | 15.3 | 33.2 | 42.7 |
| | FFN | 28.0 | 47.1 | 29.3 | 13.5 | 30.6 | 39.1 |

### E. Robustness and Efficiency Analysis

In this section, we thoroughly analyze the robustness of the proposed weight matrix factorization (4) and response matrix factorization (17). Then, the computational complexity and storage requirement of the proposed FFN are analyzed and compared with the original networks as well as BWNs.

*1) Robustness of Factorization:* In this section, we explore the robustness of the optimization procedure of the proposed ternary factorization. We use an iterative optimization strategy for the optimization of (4) and (17), i.e., the weight matrix factorization and response matrix factorization. Note that each subproblem of (4) and (17) can steadily reduce the quantization error of the corresponding optimization problem.

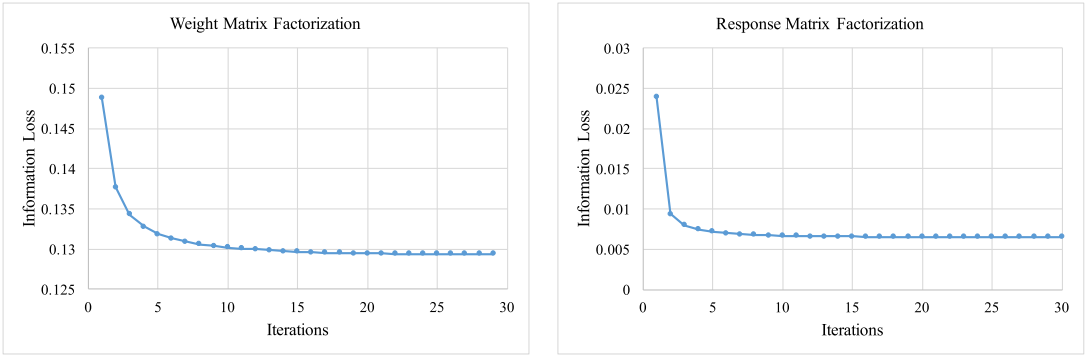
To show that the iterative optimization procedure can steadily reduce the quantization error, we first define the information loss for weight matrix factorization ($L_w$) and response matrix factorization ($L_r$), corresponding to (4) and (17), respectively. The information losses are defined as follows:

$$L_w = \frac{\| W - \hat{U} D \hat{V}^T \|_F^2}{\| W \|_F^2} \tag{33}$$

$$L_r = \frac{\| Y - \hat{U} D \hat{V}^T X \|_F^2}{\| Y \|_F^2}. \tag{34}$$

Then, we check the information loss during the iterative optimization procedure, and the results are shown in Fig. 6. From Fig. 6, we can see that the information loss for both weight matrix factorization and response matrix factorization can be steadily reduced with more iterations.

*2) Inference Efficiency:* FFN utilizes ternary weights, and we empirically find that about half of weights are zeros. Fig. 7 shows the sparsity of all layers within the residual blocks of ResNet-101. The sparsity is calculated based on the ratio of zeros in $\hat{U}$ and $\hat{V}$ after fixed-point factorization. The sparsity ranges from 40% to 70%, and different layers have different




(a) (b)

Fig. 6. Information loss for the optimization of (a) weight matrix factorization and (b) response matrix factorization. For both cases, the information loss can be steadily reduced with more optimization iterations.

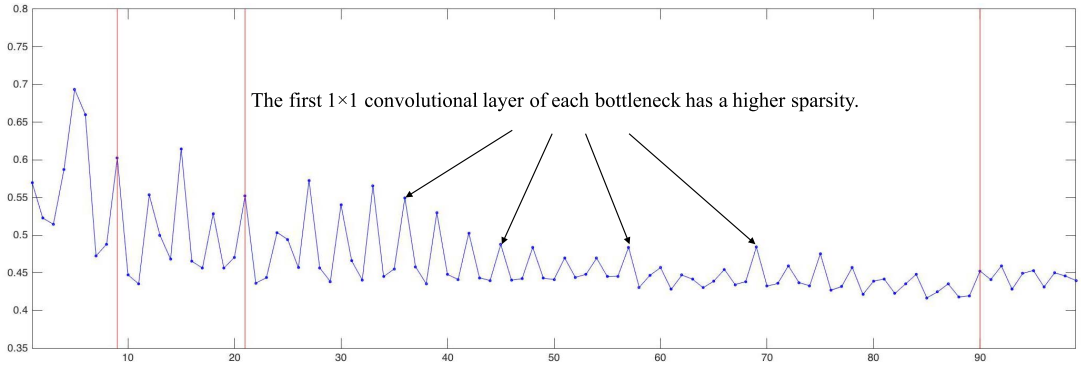



Fig. 7. Sparsity of all layers within the residual blocks of ResNet-101 after ternary quantization. The sparsity is calculated based on the ratio of zeros in $\hat{U}$ and $\hat{V}$ after fixed-point factorization. Each consecutive three points correspond to the sparsity of three layers of a bottleneck. As a baseline, we have checked the sparsity of full-precision ResNet-101, for which each layer has a sparsity of 0%.

sparse ratios. It can also be seen that lower layers, i.e., layers from the first residual block, have higher sparsity than upper layers. We also notice that, within each bottleneck structure (each consecutive three points correspond to the sparsity of three layers of a bottleneck), the first $1 \times 1$ convolutional layer tends to have higher sparsity.

Fig. 7 shows that the sparsity of the factorized weights is about 50%. As a baseline, we have checked the sparsity of full-precision ResNet-101, for which each layer has a sparsity of 0. The sparsity for binary-based methods is also 0. Thus, the computational complexity is about half of binary-based methods, such as BC [9].

Table IX shows the detailed computation and storage complexity of AlexNet, VGG-16, ResNet-50, and ResNet-101. For the computation complexity, multiplication and addition operations needed by the inference of one input image are given, which are indicated by "Mul" and "Add," respectively. As for storage consumption, we report the number of bytes needed to store the whole models. To make a comparison, we report the computation and storage complexity of the original full-precision models, the binary quantized models, as well as our FFN quantized models. Note that we do not make a difference between different binary approaches, such as BWN, because most of the binary approaches, including BWN, have almost the same resource consumption.

TABLE IX

OPERATIONS AND STORAGE REQUIREMENTS. MUL AND ADD REPRESENT THE NUMBER OF MULTIPLY AND ADDITION OPERATION. BYTES INDICATES THE NUMBER OF BYTE NEEDED TO STORE THE WEIGHTS. ALL NUMBERS ARE ACCOUNTED FOR CONVOLUTIONAL LAYERS AND FC LAYERS

| Model | | AlexNet | VGG-16 | ResNet-50 | ResNet-101 |
|---|---|---|---|---|---|
| Original | Mul | 725M | 15471M | 4212M | 7801M |
| | Add | 725M | 15471M | 4212M | 7801M |
| | Bytes | 244M | 528M | 97.3M | 178M |
| Binary | Mul | 0.66M | 13.5M | 10.6M | 16.2M |
| | Add | 725M | 15471M | 4212M | 7801M |
| | Bytes | 7.7M | 16.6M | 3.1M | 5.5M |
| FFN | Mul | 0.66M | 11.7M | 4.4M | 8.6M |
| | Add | 392M | 8631M | 1907M | 5025M |
| | Bytes | 11.5M | 25.8M | 4.9M | 10.0M |

From Table IX, it can be concluded that both binary methods and the proposed FFN can dramatically remove multiplication operations. For some architecture, such as ResNet-50 and ResNet-101, the FFN has about half multiplications compared with binary methods, which is due to the small factorization rank of FFN. For additions, binary approaches have the same addition operations as full-precision counterparts because there is no sparsity of binary approaches. By contrast, the FFN only needs about half additions due to the $\sim$ 50% sparsity

of factorized weights. The disadvantage of using ternary weights is that it needs a little more storage than binary weights. Especially, our ternary method has about 1.5-bit weight representation because of the sparsity. Thus, the storage consumption of FFN is about 1.5× of binary approaches.

## VI. Conclusion

We introduce a novel fixed-point factorized framework, named FFN, for DNN acceleration and compression. To make full use of the pretrained models, we propose a novel full-precision weight recovery method, which makes the fine-tuning more efficient and effective. Moreover, we present a weight balancing technique to stabilize the retraining process. An unsupervised version of FFN is also developed to deal with the circumstance where limited unlabeled data or even no data are provided. Extensive experiments on the ImageNet classification and object detection on MS COCO show that, even without labels, the proposed FFN with ternary weights can achieve comparable accuracy to the full-precision counterparts, resulting in about more than 20× compression and removing most of the multiply operations.

## References


[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[3] R. Abbasi-Asl and B. Yu, "Structural compression of convolutional neural networks," 2017, *arXiv:1705.07356*. [Online]. Available: http://arxiv.org/abs/1705.07356

[4] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 5068–5076.

[5] M. Denil *et al.*, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.

[6] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, Oct. 2015.

[7] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*. [Online]. Available: http://arxiv.org/abs/1511.06530

[8] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proc. ACM Multimedia Conf.*, 2016, pp. 541–545.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Computer Vision* (Lecture Notes in Computer Science), vol. 9908. Cham, Germany: Springer, 2016, pp. 525–542.

[11] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, and J. Cheng, "Two-step quantization for low-bit neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4376–4384.

[12] T. Dettmers, "8-bit approximations for parallelism in deep learning," 2015, *arXiv:1511.04561*. [Online]. Available: http://arxiv.org/abs/1511.04561

[13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Jul. 2015, pp. 1737–1746.

[14] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," 2016, *arXiv:1603.01025*. [Online]. Available: http://arxiv.org/abs/1603.01025

[15] Q. Hu, P. Wang, and J. Cheng, "From hashing to CNNs: Training binary weight networks via hashing," in *Proc. AAAI*, Feb. 2018, pp. 3247–3254.

[16] P. Wang, X. He, G. Li, T. Zhao, and J. Cheng, "Sparsity-inducing binarized neural networks," in *Proc. AAAI*, 2020, p. 12192–12199.

[17] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016, *arXiv:1605.04711*. [Online]. Available: http://arxiv.org/abs/1605.04711

[18] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*. [Online]. Available: http://arxiv.org/abs/1612.01064

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[21] P. Wang and J. Cheng, "Fixed-point factorized networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4012–4020.

[22] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 1, pp. 64–77, Jan. 2018.

[23] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*. [Online]. Available: http://arxiv.org/abs/1405.3866

[24] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[25] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using finetuned CP-decomposition," 2014, *arXiv:1412.6553*. [Online]. Available: http://arxiv.org/abs/1412.6553

[26] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.

[27] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[28] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2755–2763.

[29] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan, "Training binary multilayer neural networks for image classification using expectation backpropagation," 2015, *arXiv:1503.03562*. [Online]. Available: http://arxiv.org/abs/1503.03562

[30] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.

[31] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *Proc. 15th Eur. Conf.*, Munich, Germany, Sep. 2018, pp. 747–763.

[32] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 344–352.

[33] S. Zhu, X. Dong, and H. Su, "Binary ensemble neural network: More bits per network or more networks per bit?" in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4923–4932.

[34] J. Fromm, S. Patel, and M. Philipose, "Heterogeneous bitwidth binarization in convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4010–4019.

[35] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+1, 0, and- 1," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2014, pp. 1–6.

[36] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.

[37] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized CNN: A unified approach to accelerate and compress convolutional networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4730–4743, Oct. 2018.

[38] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: http://arxiv.org/abs/1503.02531

[39] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," 2016, *arXiv:1612.03928*. [Online]. Available: http://arxiv.org/abs/1612.03928

[40] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 742–751.

[41] T. Wang, L. Yuan, X. Zhang, and J. Feng, "Distilling object detectors with fine-grained feature imitation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4933–4942.
[42] G. Zhu, J. Wang, P. Wang, Y. Wu, and H. Lu, "Feature distilled tracking," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 440–452, Feb. 2019.
[43] K. Bhardwaj, N. Suda, and R. Marculescu, "Dream distillation: A data-independent model compression framework," 2019, *arXiv:1905.07072*. [Online]. Available: http://arxiv.org/abs/1905.07072
[44] H. Chen *et al.*, "Data-free learning of student networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3514–3522.
[45] Y. Xu *et al.*, "Positive-unlabeled compression on the cloud," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 2561–2570.
[46] X. He and J. Cheng, "Learning compression from limited unlabeled data," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 752–769.
[47] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 7948–7956.
[48] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *Proc. 37nd Int. Conf. Mach. Learn. (ICML)*, Jul. 2020, pp. 243–252.
[49] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1325–1334.
[50] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Proc. 10th Int. Workshop Frontiers Handw. Recognit.*, 2006, pp. 1–8.
[51] T. G. Kolda and D. P. O'Leary, "A semidiscrete matrix decomposition for latent semantic indexing information retrieval," *ACM Trans. Inf. Syst.*, vol. 16, no. 4, pp. 322–346, Oct. 1998.
[52] T. G. Kolda and D. P. O'Leary, "Algorithm 805: Computation and uses of the semidiscrete matrix decomposition," *ACM Trans. Math. Softw.*, vol. 26, no. 3, p. 415–435, Sep. 2000, doi: 10.1145/358407.358424.
[53] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *J. Mach. Learn. Res.*, vol. 9, pp. 249–256, May 2010.
[54] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167
[55] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3466–3473.
[56] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–15.
[57] Z. He and D. Fan, "Simultaneously optimizing weight and quantizer of ternary neural network using truncated Gaussian approximation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, p. 11.
[58] J. Yang *et al.*, "Quantization networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Dec. 2019, pp. 7308–7316.
[59] M. Simon, E. Rodner, and J. Denzler, "ImageNet pre-trained models with batch normalization," 2016, *arXiv:1612.01452*. [Online]. Available: http://arxiv.org/abs/1612.01452
[60] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.
[61] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2961–2969.

**Peisong Wang** received the B.E. degree in software engineering from Shandong University, Jinan, China, in 2013, and the Ph.D. degree in computer science from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2018.

He is currently an Assistant Researcher with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences. His current research interests include computer vision and network acceleration and compression.

**Xiangyu He** received the B.E. degree in information security from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. He is currently pursuing the Ph.D. degree with the Institute of Automation, Chinese Academy of Sciences, Beijing.

His current research interests include deep learning, image retrieval, and high-performance computing.

**Qiang Chen** received the B.S. degree from Beihang University, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree with the Institute of Automation, Chinese Academy of Sciences, Beijing.

His current research interests include object detection and image segmentation.

**Anda Cheng** received the B.S. degree from the School of Information Science, Northeastern University (NEU), Shenyang, China, in 2017. He is currently pursuing the Ph.D. degree with the Institute of Automation, Chinese Academy of Sciences, Beijing, China.

His main research interests include semantic segmentation and metalearning.

**Qingshan Liu** (Senior Member, IEEE) received the Ph.D. degree from the National Laboratory of Pattern Recognition (NLPR), Chinese Academy of Sciences, Beijing, China, in 2003.

He was with NLPR. From April 2006 to August 2011, he was with the Department of Computer Science, Computational Biomedicine Imaging and Modeling Center, Rutgers University, New Brunswick, NJ, USA. He is currently a Professor with the B-Data Laboratory, Nanjing University of Information Science and Technology, Nanjing, China. His current research interests include image and vision analysis, computer vision, and pattern recognition.

**Jian Cheng** (Member, IEEE) received the B.S. and M.S. degrees in mathematics from Wuhan University, Wuhan, China, in 1998 and 2001, respectively, and the Ph.D. degree in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2004.

He is currently a Professor with the Institute of Automation, Chinese Academy of Sciences. His current major research interests include deep learning, computer vision, and chip design.