Compact Neural Network via Stacking Hybrid Units

Weichao Lan, Yiu-Ming Cheung, *Fellow, IEEE*, Juyong Jiang, Zhikai Hu, and Mengke Li

***Abstract*—As an effective tool for network compression, pruning techniques have been widely used to reduce the large number of parameters in deep neural networks (NNs). Nevertheless, unstructured pruning has the limitation of dealing with the sparse and irregular weights. By contrast, structured pruning can help eliminate this drawback but it requires complex criteria to determine which components to be pruned. Therefore, this paper presents a new method termed BUnit-Net, which directly constructs compact NNs by stacking designed basic units, without requiring additional judgement criteria anymore. Given the basic units of various architectures, they are combined and stacked systematically to build up compact NNs which involve fewer weight parameters due to the independence among the units. In this way, BUnit-Net can achieve the same compression effect as unstructured pruning while the weight tensors can still remain regular and dense. We formulate BUnit-Net in diverse popular backbones in comparison with the state-of-the-art pruning methods on different benchmark datasets. Moreover, two new metrics are proposed to evaluate the trade-off of compression performance. Experiment results show that BUnit-Net can achieve comparable classification accuracy while saving around 80% FLOPs and 73% parameters. That is, stacking basic units provides a new promising way for network compression.**

***Index Terms*—Model compression, network pruning, compact networks, convolutional neural networks, generalization.**

I. Introduction

DEEP neural networks (DNNs) have obtained superior performance and become indispensable tools in computer vision community, such as image classification [1], [2], object detection [3], [4], and semantic segmentation [5], [6]. However, the widely-recognized properties of over-parameterization and redundancy result in huge consumption of memory footprint and computation cost, which hinder their practical applications. Considering the limited storage space and computation capacity of mobile and edge devices that are often resource-constraint, it is critically desired to reduce the number of parameters and floating-point operations (FLOPs) of DNNs for better deployment [7]. To obtain more efficient models, many techniques have been explored for compression and acceleration, including pruning [8], [9], quantization [10], low-rank decomposition [11], [12], knowledge distillation [13], [14], and compact model design [15], [16]. The first four methods are usually destructive for the original model because the parameters or structure will be adjusted, while compact model design method is constrictive that directly create efficient models. Among them, network pruning has been a mainstream branch in both academia and industry due to the advantages of saving resources, reducing latency and addressing privacy concerns, showing broad prospects in numerous applications [17], [18].

The pruning frameworks aim at eliminating the redundancy of deep models by removing the components with less importance. According to the types of removed components, there are mainly two kinds of pruning methods: unstructured pruning and structured pruning. Unstructured pruning removes the connections or neurons in weight matrices [8], [19], [20]. A common standard to determine which weights should be pruned is magnitude-based pruning that compares the weight amplitude with a threshold [19]. First, a predefined quality parameter is multiplied by the standard deviation of weights to calculate the threshold, and then the weights with lower magnitude than the threshold will be set to zero. After all layers are pruned, the model needs to be retrained so that the remaining weights can be adjusted to compensate for the removed ones. However, this kind of low-level pruning has the risk of being non-structural that may hinder the actual acceleration, owing to the irregular memory access mode. Some special software and hardware such as sparse CNN accelerators based on ASIC [21], [22] and FPGA [23], [24] can alleviate this problem, but also brings extra cost to the deployment of models.

To achieve more practical compression and acceleration, the well-supported structured pruning methods such as channel and filter pruning [25], [26], [27] have been explored recently. Although the structured pruning for convolution kernels and graphs can obtain hardware-friendly network, they usually need specific criteria or complex mechanisms to identify the irrelevant subset of the components for elimination, which will increase the memory requirement and computation cost. For example, the reconstruction-based methods try to minimize the reconstruction error of feature maps based on the pretrained model to achieve pruning [28], [29], consuming much memory to store the feature maps of pretrained models. On the other hand, most of the current methods on compression and acceleration are conducted

Manuscript received 7 July 2022; revised 29 July 2023; accepted 5 October 2023. Date of publication 10 October 2023; date of current version 5 December 2023. This work was supported in part by NSFC and Research Grants Council (RGC) Joint Research Scheme under Grant N_HKBU214/21, in part by the General Research Fund of RGC under Grants 12201321 and 12202622, and in part by Hong Kong Baptist University under Grant RC-FNRA-IG/18-19/SCI/03. Recommended for acceptance by J. Li. *(Corresponding author: Yiu-Ming Cheung.)*

Weichao Lan, Yiu-Ming Cheung, and Zhikai Hu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, China (e-mail: cswclan@comp.hkbu.edu.hk; ymc@comp.hkbu.edu.hk; cszkhu@comp.hkbu.edu.hk).

Mengke Li is with the Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518060, China (e-mail: csmkli@comp.hkbu.edu.hk).

Juyong Jiang is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, China, and also with the Hong Kong University of Science and Technology (GZ), Guangzhou 510230, China (e-mail: csjuyongjiang@gmail.com).

Digital Object Identifier 10.1109/TPAMI.2023.3323496

© 2023 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

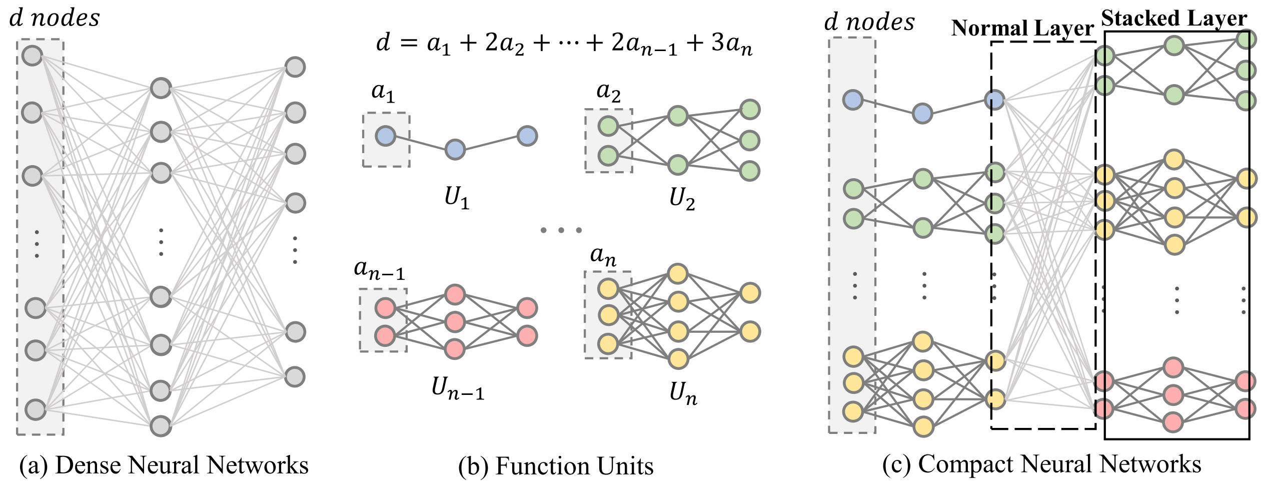

Fig. 1. Construct BUnit-Net via stacking designed basic units. (a) The dense neural network as baseline for comparison; (b) Different basic units with various input, hidden and output nodes; (c) The compact BUnit-Net with stacked units where the each unit is independent with each other. A compact BUnit-Net contains stacked layers and normal layers, where the compression will be achieved by stacked layer. The number of input node d in dense network is the summation of the input nodes of all units that are selected to stack.

on pretrained neural networks. For example, pruning methods try to remove the unimportant components of pretrained models and usually require extra iterations to recover the accuracy, while network quantization achieves compression through quantizing the parameters such as weights and activations in existing networks.

Under the circumstances, it is interesting to ask whether it is possible to have a human-designed compact networks that can achieve comparable or even better compression performance than pruned networks learned through training. Since the current pruning methods are mainly conducted on pretrained neural networks, the performance of compressed model is highly affected by the quality of the pretrained models, which will give rise to a restriction if the pretrained models are not well trained. In some cases, even fewer or no pretrained models are available. Another concern is that network pruning often involves addition processes such as fine-tuning, re-initializing and rewinding, which may become an obstacle to implement the network because the generalizability of a given pruning framework in different architectures is usually unclear. Therefore, simply using a more efficient architecture would be more effective than pruning a suboptimal network in many cases [30].

To this end, we therefore propose to directly construct compact neural networks by independently stacking designed basic units (BUnit-Net) from a new perspective. We first design the basic unit as a small neural network with at least one hidden layer, and then combine these basic units following a certain stacking strategy to construct an entire network. The whole framework of our proposed method is illustrated in Fig. 1. In the framework, a group of nodes (i.e., neurons) forms a basic function unit just like the nervous tissues in human brain with different functions. Since there is no connection between each unit due to the independence, our BUnit-Net simply achieves weight compression without additional mechanisms for judging importance of weights. Different from the sparse and irregular weight tensors generated by unstructured pruning, the weights of each unit in BUnit-Net are regular and dense so that they can be easily processed without special devices. Moreover, the dependence on pretrained model will also be sidestepped because BUnit-Net is directly trained from scratch. Besides, BUnit-Net can be built as numerous network backbones such as convolutional neural networks (CNNs) and fully-connected networks by designing different basic units, with strong generalizability. In the experiments, we first explore the effect of different units through visualizing feature maps. Then, we conduct extensive experiments on multiple datasets (i.e., MNIST [31], CIFAR [32], Tiny ImageNet [33] and ImageNet-2012 [34]) with popular network structures (i.e., MLP, VGG [35], ResNet [2], and MobileNetV2 [36]). Experimental results demonstrate the efficacy of BUnit-Net for network compression on all datasets we have evaluated on. When comparing the performance of different compressed methods, we not only report the commonly-used metrics such as accuracy, FLOPs and parameters drop ratio, but also propose two new metrics to evaluate the trade-off between accuracy and model size (i.e., number of FLOPs and parameters), which are helpful to select the proper methods under different requirements in real-applications. As far as we know, this work is the first attempt to construct compact networks under human-designed stacking structure, showing great potential on compressing network.

We summarize our main contributions as follows:

1) We propose BUnit-Net that directly constructs compact neural networks by combining and stacking well-designed basic units. It achieves compression through the independence of each unit. Using different kinds of basic units, BUnit-Net can be applied on various models.
2) Different basic units and combining strategy are explored through several experiments to verify the flexibility and efficacy of BUnit-Net.
3) We define two novel metrics to measure the trade-off between accuracy and resource consumption including

the number of FLOPs and parameters, which are helpful in decision-making when choosing an appropriate compressed model for different aims.

The rest of this article is organized as follows. Related work is presented in Section II. Section III introduces the details of our proposed BUnit-Net. In Section IV, the two proposed metrics are introduced in detail, and the experiment results including comparison with different compression methods are provided. Finally, the limitations of our proposed method and the conclusions of this paper are given in Section V.

II. Related Work

A. Network Pruning

Network pruning has been widely used to reduce the complexity and computation of CNNs because of its effectiveness and simplicity. The earliest pruning works date back to 1990 s that use Hessian approximation to compute the saliency of parameters [37], [38]. And later, researchers continue exploring various pruning methods. Existing methods can be roughly divided into unstructured pruning and structured pruning in terms of the types of network components to be pruned.

Unstructured pruning involves removing weights or neurons that increases the sparsity of the network by replacing the connections or neurons with zero in the weight matrix. Han et al. [8] described a three-stage deep compression framework including network pruning, quantization and Huffman coding, where the weights were pruned by comparing the magnitude with a preset threshold. However, in such pruning technique, a weight will remain zero in the subsequent retraining process once it is removed which may cause large accuracy loss. Guo et al. [39] then proposed a dynamic network pruning framework, introducing a recovery operation in order to restore the connection that has been wrongly pruned. Zhang et al. [40] converted weight pruning as a constrained non-convex optimization problem.

Structured pruning removes the entire channels or filters, breaking the limitation that unstructured pruning heavily relies on technical hardware or software library to achieve compression and speedup because of the sparse weights. The channel pruning methods have developed different strategies such as variational technique [26] and genetic algorithm [41] to choose the unimportant channels. CGNet [25] identifies the salient channels and skip the remaining parts with less importance by the designed squeeze-excitation modules. However, the skipped regions in CGNet are irregular that require special toolkit to achieve practical compression. Recent work [42] has introduced channel independence to measure the importance. Furthermore, filter-level pruning is also proved to be efficacious. Along this line, Li et al. [29] first calculated the norms of filters for evaluation. Later, ThiNet [43] utilizes the features of the next layer to guide the pruning of the current layer. Further, [44] introduces group convolution scheme to save storage. Also, He et al. [27] proposed to rank the importance of convolution filters based on geometric median to overcome the shortcoming of norm-based pruning. The sparsity regularization measurements are explored in [45], [46]. A separate sub-network is introduced in GaterNet [47] to generate gates for adaptively selecting the activated filters. AutoPruner [48] automatically identifies the less important filters in an end-to-end manner. [49] measures the importance of latent representations to identify the pruned filters. Yeom et al. [50] combines interpretability and filter pruning based on layer-wise relevance propagation (LRP) [51].

Since the traditional pruning methods often involve hyper-parameters that require time-consuming human design, how to pruning automatically has become a hot issue in recent years [52], [53]. Besides, Neural Architecture Search (NAS) [54] has also been combined with pruning in these years such as Metapruning method [55].

B. Compact Model Design

Compact model design methods focus on changing the basic operation and redesigning the structure to reduce the complexity and model size. It is a widely used method to construct a lightweight network with different cheaper operations. In the Network in Network (NIN) [56] model, the architecture of embedded network is developed that uses 1×1 convolution to increase capacity while decreasing computational complexity. Besides 1×1 convolution, SqueezeNet [15] utilizes group convolution for further speedup. Howard et al. [16] designed MobileNet using branching strategy, where each branch contained only one channel called depth-wise convolution. The further work termed MobileNetV2 [36] introduces residual and linear bottleneck structure. ShuffleNet [57] combines group convolution and channel shuffle operation that shuffles channels before the next convolution operation in order to realize information transmission between multiple groups. Furthermore, EspNetV2 [58] proposes deep expandable and separable convolution to improve efficiency. Jeon et al. [59] proposed active shift operation to save memory, and then the sparse shift layer (SSL) applied in FE-Net [60] eliminates the meaningless shift operation. In addition to manually design light-weight models, NAS technique [54] that can construct network automatically has also attracted attention recently. EfficientNet [61] first utilizes NAS to search an efficient backbone, and then the backbone is scaled to deal with different inputs, where the scaling is conducted on three dimensions (network width, depth and resolution) at the same time. Based on NAS, [62] and [63] have improved MobileNetV2 with comparable performance. However, these methods still require large resource consumption that is difficult to deploy on practical applications.

C. Other Compression Methods

In addition to pruning and compact models, other methods are also explored to compress networks and speed up the inference, including quantization [10], low-rank decomposition [11], [12] and knowledge distillation (KD) [13], [14]. Network quantization method achieves compression by quantizing the weights or activations to low-bit representations such as binary and ternary neural networks [64], [65], [66]. In this way, the 32-bit floating point parameters can be converted to low-bit or even 1 b to reduce resource consumption. Low-rank decomposition approximates the large weight matrices by the product of several low-dimensional matrices to accelerate computation. The most common used decomposition techniques include Singular Value Decomposition (SVD) [11], [67] and Tucker Decomposition [68]. With respect to KD, the whole framework usually contains two networks, where the knowledge in the larger network (teacher) is utilized to supervise the training of the smaller one (student). The theoretical basis of KD comes from [69] that first proposed to train the student through the output of softmax layer of teacher. Later, the distillation method is continuously improved from different aspects, such as using intermediate feature maps [13], [14] or multiple teachers [70], [71].

III. Proposed Method

A. Preliminaries

Before describing our proposed method, we formally give some notations and symbols. In a CNN, the weight of a standard convolutional layer is a four-dimensional tensor $\mathbf{W} \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$, where $d \times d$ is the kernel size of the convolution filter, c_{in} and c_{out} is the number of input and output channels. For a three-dimensional input $\mathbf{X}_{in} \in \mathbb{R}^{w_{in} \times h_{in} \times c_{in}}$ with width w_{in} and height h_{in}, the convolutional layer can transform it into another three-dimensional tensor $\mathbf{X}_{out} \in \mathbb{R}^{w_{out} \times h_{out} \times c_{out}}$. Let $\text{Conv}(\cdot)$ denote the operations in a convolutional layer including convolution and activation, the transformation can be formulated as

$$\mathbf{X}_{out} = \text{Conv}(\mathbf{X}_{in}, \mathbf{W}). \tag{1}$$

For a given convolutional layer, the weight $\mathbf{W}$ contains $\{d \times d \times c_{in} \times c_{out}\}$ parameters, and the convolution operation requires $\{d \times d \times c_{in} \times w_{out} \times h_{out} \times c_{out}\}$ FLOPs. Because a CNN model is usually composed by a large number of convolutional layers, the cost of memory and computation will be huge that makes it a challenge to deploy CNNs on resource-constraint devices.

B. Basic Function Unit and Stacked Layers

As illustrated in Fig. 1, one of the most important components of the proposed BUnit-Net is the designed basic unit. We take CNN as an example to demonstrate our method.

Without loss of generality, we initially design the unit as a small convolutional neural network with at least one hidden layer, where the nodes can be adjusted flexibly. Suppose the unit contains three layers and the node numbers in each layer is $\{c'_{in}, c_h, c'_{out}\}$, respectively. The two weight matrices in a unit can be represented as $\mathbf{W}_l \in \mathbb{R}^{d \times d \times c'_{in} \times c_h}$ and $\mathbf{W}_r \in \mathbb{R}^{d \times d \times c_h \times c'_{out}}$. Then, for a given input $\mathbf{X}'_{in} \in \mathbb{R}^{w_{in} \times h_{in} \times c'_{in}}$, the output $\mathbf{X}'_{out} \in \mathbb{R}^{w_{out} \times h_{out} \times c'_{out}}$ of a single unit can be calculated as

$$\mathbf{X}'_{out} = \text{Unit}(\mathbf{X}'_{in}) = \text{Conv}(\text{Conv}(\mathbf{X}'_{in}, \mathbf{W}_l), \mathbf{W}_r). \tag{2}$$

After designing the basic unit, we can then obtain the whole convolutional layers by stacking a certain number of units together. Suppose m units with various c'_{in} are selected to stack and they are represented as $\{\mathbf{U}^1, \mathbf{U}^2, \ldots, \mathbf{U}^m\}$. For the entire input $\mathbf{X}_{in} \in \mathbb{R}^{w_{in} \times h_{in} \times c_{in}}$ of the whole layer, it first needs to be split into m pieces, where the input channel c'_{in} of each piece is the same as the corresponding unit, so that it can be processed with the unit because the unit only contains c'_{in} input channels. Note that the total input channels of m units should be consistent with c_{in}. If these m units have the same input channel numbers, we can equally split $\mathbf{X}_{in}$ into $m = \frac{c_{in}}{c'_{in}}$ pieces. Let $\{\mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^m\}$ donate m smaller input pieces with the corresponding m outputs $\{\mathbf{X}^1_{out}, \mathbf{X}^2_{out}, \ldots, \mathbf{X}^m_{out}\}$. To ensure that the output of stacked layers can be transformed successfully into the next layer, these m small outputs are also needed to be concatenated to keep the dimensions consistent. Thus, the output of the whole stacked convolutional layers is

$$\begin{aligned}\mathbf{X_{out\_new}} &= \text{Concat}(\mathbf{X}_{in}) \\ &= [\mathbf{X}^1_{out}, \mathbf{X}^2_{out}, \ldots, \mathbf{X}^m_{out}] \\ &= [\text{Unit}(\mathbf{X^1}), \text{Unit}(\mathbf{X^2}), \ldots, \text{Unit}(\mathbf{X^m})],\end{aligned} \tag{3}$$

where the output of each unit is calculated by (2).

By utilizing the simple stacking strategy, we can avoid the sparse weights generated after unstructured pruning. The weight tensors in BUnit-Net will remain *dense* so that it can achieve compression using general-purpose hardware and software, without the support of complex judgment criteria as well. The detailed analysis on memory and computation cost is presented in Section III-E. In addition to CNN, BUnit-Net is also applicable to other network backbones by designing diverse basic units, as shown in Fig. 1(b).

C. Network Construction

In BUnit-Net, there are two kinds of convolutional layers termed as *Stacked Layer* (solid box) and *Normal Layer* (dashed box) as marked in Fig. 1(c). Stacked layer refers to the layers generated by stacking basic units while normal layer is a single standard convolutional layer. Because of the independence among each unit in stacked layers, it is necessary to add normal convolution layers between the stacked layers; otherwise, the entire network will be disconnected resulting in the failure of feature information transmission. Moreover, the normal layer between the stacked layers can also help deal with the dimension matching problem. After obtaining the stacked layers as described in Section III-B, a whole neural network can be constructed using these stacked layers and normal layers alternately. It is notable that the setting of a normal layer is determined by the two contiguous stacked layers. For instance, a normal layer is placed between two stacked layers in Fig. 1(c). Assuming that the two layers contain m and n units respectively, then the input nodes (NL_{in}) of the normal layer are the sum of the output nodes (c'_{out}) of m units on the left, while the output nodes (NL_{out}) is the total number of n unit input nodes on the right.

It is straightforward to implement BUnit-Net with stacked layer as classical network architectures like VGG-style and ResNet-style backbones (style means the similar architecture). As shown in Fig. 2(a), we can reconfigure the continuous standard convolutional layers in VGG-style networks by replacing the first few layers with stacked layers. With respect to ResNet-style in Fig. 2(b), we can also introduce the residual structure in BUnit-Net. Furthermore, the constraint on dimensions between

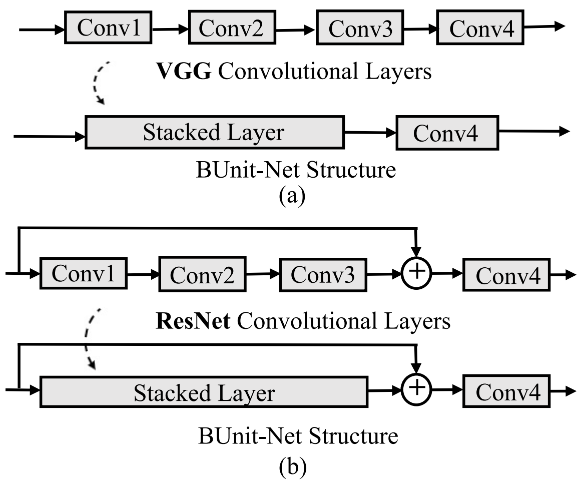

Fig. 2. Network construction. (a) VGG-style Network; (b) ResNet-style Network. In general, the basic units are designed as three-layer architecture. To construct a VGG-style BUnit-Net, the most direct way is to use one stacked layer and one normal layer alternatively. The residual structure can also be added between the layers to build a ResNet-style BUnit-Net.

different layers can be flexibly solved by adjusting the parameters in stacked layer such as convolution stride. In addition to CNN, BUnit-Net is appropriate for fully-connected networks like Multilayer Perceptron (MLP) if the unit is designed as a small network with only fully-connected layers. In terms of the special convolution operations in recent lightweight models such as 3×3 depth-wise and 1×1 point-wise convolution, the basic units can also be designed to equip with these operations, demonstrating the powerful flexibility of the proposed BUnit-Net.

D. Training Process

BUnit-Net involves two parts of weights, one part is the weights of basic units in stacked layers $\{\mathbf{W}_l, \mathbf{W}_r\}$ and the other is the weights of normal layers $\mathbf{W}_n$. Because each unit in BUnit-Net is still a neural network, it follows the standard forward and backward propagation algorithms. Given a set of data pairs $\{(\mathbf{x}, \mathbf{y})\}$, the objective function of training BUnit-Net can be set as

$$\arg\min \mathcal{L}(\hat{\mathbf{Y}}, \mathbf{y}) = \arg\min_{\mathbf{W}, \mathbf{b}} \mathcal{L}(f(C((\mathbf{W}, \mathbf{b}); \mathbf{x}), \mathbf{y})), \quad (4)$$

where $\mathbf{W} = \{\mathbf{W}_l, \mathbf{W}_r, \mathbf{W}_n\}$ is all the weights and b is the bias. The function $C(\cdot)$ contains the Conv operation of normal layers in (1) and Concat operation in stacked layers as described in (3). The function $f(\cdot)$ refers to the calculation in other layers such as fully-connected and pooling layers, and $\mathcal{L}(\cdot)$ is a selected convex loss function.

When designing the basic function unit (generally designed as three layers), the nodes in each of the layers $\{c'_{in}, c_h, c'_{out}\}$, and the number of units to be stacked m need to be preset. Then, the input channels of m input pieces can be adjusted based on each c'_{in}. In forward propagation, the output of stacked layers is computed by (3) while the output of normal layers uses the standard convolution operation in (1). After obtaining the final output of BUnit-Net in forward propagation, we can get the gradient by performing backward propagation and update parameters using the popular optimizer such as SGD or ADAM [72]. The construction and training process of BUnit-Net is summarized in Algorithm 1.

Algorithm 1: BUnit-Net Construction and Training.

```
Construction
1: Design several basic function units $\{U_1, U_2 \ldots U_n\}$.
2: Determine the number of stacked layers $J$ and the
   number of units in each stacked layer, that is
   $M = \{m_1, m_2, \ldots, m_J\}$.
3: for i in M do
4:   Randomly selected i units from $\{U_1, U_2 \ldots U_n\}$ and
     stacked them to build the stacked layer. Skip the
     selection step if there is only single kind of unit.
5: end for
6: Calculate the input and output nodes of $J$ stacked
   layers.
7: Add the normal layer between stacked layers
   according to the obtained node numbers.
8: Add other layers such as pooling and fully-connected
   layers to construct the entire BUnit-Net.
Training
   Input: Training data pairs $\{\mathbf{X}_{train}, \mathbf{y}_{train}\}$.
   Output: Compact neural networks.
1: Initialize weight and bias in stacked layers and normal
   layers.
2: for Iter $= 1$ to maxIter do
3:   Get a minibatch of training data $\{\mathbf{x}, \mathbf{y}\}$.
4:   Compute the network output where the output of
     stacked layers and normal layers is calculated as (3)
     and (1), respectively.
5:   Compute the loss $L(\mathbf{y}, \hat{\mathbf{y}})$.
6:   Perform standard backward propagation.
7:   Update parameters using any popular optimizer.
8: end for
```

E. Analysis of Stacked Layers

In order to analyze the compression effect more intuitively, here we suppose that m units with the same nodes $\{c'_{in}, c_h, c'_{out}\}$ are stacked, a group of three convolutional layers will be constructed with $\{mc'_{in}, mc_h, mc'_{out}\}$ nodes in each layer as in Fig. 1(a). Since the stacked units are independent, which means that there is no connection among them, the memory cost of stacked layers will be

$$M_s = m\mathbf{W}_l + m\mathbf{W}_r = m \times d \times d \times c_h \times (c'_{in} + c'_{out}). \quad (5)$$

If three convolutional layers are fully connected, the two weight matrices will be $\mathbf{W}_{lf} \in \mathbb{R}^{d \times d \times mc_{in} \times mc_h}$ and $\mathbf{W}_{rf} \in \mathbb{R}^{d \times d \times mc_h \times mc_{out}}$ in which memory cost is

$$M_n = \mathbf{W}_{lf} + \mathbf{W}_{rf} = m^2 \times d \times d \times c_h \times (c'_{in} + c'_{out}) = mM_s. \quad (6)$$

Obviously, the number of parameters of stacked convolutional layers is much lower than the layers with fully connection.

By this way, our proposed method can effectively reduce the memory cost to achieve networks compression.

In terms of computation cost, the total FLOPs of m three-layer units are

$$C_s = mF_l + mF_r = m \times d \times d \times c_h \times (c'_{in} \times w_{outl} \times h_{outl} + w_{outr} \times h_{outr} \times c'_{out}), \tag{7}$$

where w_{outl} and w_{outr} are the width of the first and second layer output, respectively, h_{outl} and h_{outr} are the heights. For the fully-connected convolutional layers with the same nodes, the FLOPs are

$$C_n = d \times d \times mc_h \times (mc'_{in} \times w_{outl} \times h_{outl} + w_{outr} \times h_{outr} \times mc'_{out}) = mC_s. \tag{8}$$

Similarly, the memory and computation cost in other architectures can also be calculated in the same way.

With less parameters and FLOPs, the stacked layers in our proposed BUnit-Net is also beneficial to save much storage and computation consumption. Since we also add normal layers between the stacked layers and different kinds of units may be used when building network, the compression and acceleration of the entire BUnit-Net will be smaller than m. The actual compression performance of BUnit-Net will be reported in our experiments for comparison.

IV. Experiments

To evaluate the performance and efficiency of BUnit-Net, we initially design several experiments to explore the effect of basic function units with different structures on both MNIST [31] and CIFAR-10 [32] datasets. In addition, we then conduct empirical experiments for image classification tasks on other three popular datasets, that are CIFAR-100, Tiny ImageNet [33] and ImageNet-2012 [34]. We first construct BUnit-Net as fully-connected MLP on MNIST, and then build the representative CNNs like VGG [35], ResNet [2], and MobileNet [36]. We mainly compare BUnit-Net with baseline and the following pruning methods and lightweight models:

- VP [26]: Channel pruning using variational technique.
- DMCP [73]: A mask-based channel pruning method.
- CHIP [42]: Filter pruning through channel independence.
- FPGM [27]: Utilize geometric median to prune filters.
- DualConv [74]: Efficient group convolution for lightweight models.
- DANL [75]: Neural architecture search method without additional candidates.

A. Performance Metrics

The most commonly used metrics to assess the performance of a compressed model are numbers of parameters and FLOPs. However, when considering the *trade-off* between compression and accuracy, it will be unfeasible to evaluate different models. As we have known, there is no unified indicator that considers these factors simultaneously so far. Thus, we define two performance scores termed TCA and TSA. TCA is the computation performance or called the trade-off between accuracy and FLOPs, while TSA is the storage performance to assess the trade-off between accuracy and parameters. Assume the accuracy, FLOPs and parameters of the baseline are $\{a_b, f_b, p_b\}$ respectively, and $\{a_n, f_n, p_n\}$ are the corresponding metrics of the new compressed model, then the two scores are calculated as

$$TCA = \frac{e^{w_1 \frac{f_b - f_n}{f_b}}}{e^{w_2 \frac{a_b - a_n}{a_b}}}, \quad TSA = \frac{e^{w_1 \frac{p_b - p_n}{p_b}}}{e^{w_2 \frac{a_b - a_n}{a_b}}}, \tag{9}$$

where w_1 and w_2 are two scaling factors used to adjust the importance of different metrics according to specific demands in different applications. For example, a larger w_2 can be chosen if we care more about accuracy. In turn, if we pay more attention to computation and storage consumption in some cases, then we can set w_1 to be greater than w_2. By apply the scaling factors, we can choose the model more flexibly under different constraints. In fact, the essence of these two scores is the ratio between the accuracy drop and FLOPs drop (or parameter drop) after compressing. However, the magnitude gap between accuracy drop and FLOPs drop (or parameter drop) is usually too large. For instance, some compression methods are able to reduce more than 80% FLOPs or parameters while the accuracy loss can be less than 1%. Therefore, the exponential function here is introduced to narrow the gap. In other words, the role of exponentiation is to automatically magnify the subtle differences on accuracy or diminish the remarkable gap on FLOPs (or parameter) compared to the baseline, leading to a fair comparison of different compression methods.

B. Experimental Setting

1) BUnit-Net Construction: We build the diverse kinds of BUnit-Net under different network architectures to verify the generalizability. Following the reconfiguration in Fig. 2, we replace the layers in original VGG and ResNet with our stacked layers and treat the original networks as baseline. To simplify the structure, we replace the last fully-connected layers with an average pooling layer. When constructing the stacked layers in BUnit-Net, we consider both single and hybrid kinds of units. On ResNet-style network, we explore different strategies to deploy stacked layers, such as replacing the intermediate blocks while remaining the layers in the first and last blocks.

2) Training Strategy: For CIFAR datasets, we train BUnit-Net for 500 epochs with the initial learning rate of 0.1. BUnit-Net on MNIST and Tiny-ImageNet is trained 200 epochs. Stochastic gradient decent (SGD) with momentum 0.9 is used as optimizer on both CIFAR and Tiny-ImageNet. For ImageNet, we also train the models for 200 epochs, where the learning rate is 0.01. Adam with 10^{-4} weight decay and 0.01 epsilon is applied as optimizer. For all the dataset, the batch size is 128 and the learning rate is adjusted with cosine annealing. The cross-entropy loss is adopted as a criterion for all datasets. In addition, all the training procedures are warmed up with 10 epochs and applied weight initialization. As for the network backbones, VGG-style BUnit-Net is trained on CIFAR and Tiny-ImageNet, while ResNet-style BUnit-Net is applied on CIFAR and ImageNet. Besides, we also evaluate the performance of lightweight model MobileNetV2 on

TABLE I
RESULTS ON MNIST USING SINGLE AND HYBRID KINDS OF UNITS

Single				Hybrid			
Model	FLOPs(M)	Param(k)	Acc(%)	Model	FLOPs(M)	Param(k)	Acc(%)
$\mathbf{U}^1$	0.72	1.41	98.53 ± 0.05	Hybrid[1]	0.92	1.78	98.52 ± 0.10
$\mathbf{U}^2$	0.95	1.83	98.38 ± 0.07	Hybrid[2]	0.95	1.83	98.66 ± 0.06
$\mathbf{U}^3$	1.19	2.26	98.70 ± 0.06	Hybrid[3]	1.04	1.99	98.61 ± 0.05

TABLE II
RESULTS OF TEN INITIAL EXPERIMENTS WITH DIFFERENT COMBINATIONS OF BASIC FUNCTION UNITS

	Stacked-layer 1						Stacked-layer 2						Acc (%)	Flops (M)	Param (k)
	A	B	C	D	E	F	A	B	C	D	E	F			
No.1	3	3	2	3	5	4	4	7	2	0	4	3	71.42	9.24	90.77
No.2	5	5	2	2	3	3	3	4	1	4	4	4	72.00	9.60	89.98
No.3	5	3	1	3	4	4	3	4	2	3	5	3	71.63	9.33	91.11
No.4	3	4	5	2	2	4	3	1	6	3	3	4	71.42	10.92	106.64
No.5	5	4	1	2	3	5	1	6	2	6	2	3	71.47	9.53	104.05
No.6	4	6	3	3	2	2	3	3	3	3	6	2	71.91	10.08	95.39
No.7	0	4	3	3	3	7	3	3	2	4	5	3	72.21	11.00	93.29
No.8	2	2	2	3	6	5	3	5	4	1	5	2	71.11	9.30	98.26
No.9	3	2	1	3	4	7	6	4	5	1	1	3	71.42	9.06	100.00
No.10	2	6	5	2	3	2	1	2	5	5	3	6	72.08	11.70	105.99

CIFAR. The experiments on CIFAR and ImageNet dataset are conducted on a NVIDIA Tesla P40 GPU and 2 NVIDIA Tesla V100S GPUs, respectively.

C. Exploration on Basic Units

1) Hybrid Units versus Single Unit: We first apply hybrid kinds of units in stacked layers to verify the flexibility of BUnit-Net as illustrated in Fig. 1(c). Specifically, we design three different units $\{\mathbf{U}^1, \mathbf{U}^2, \mathbf{U}^3\}$ with increment on FLOPs and parameters, where the neurons in $\{\mathbf{U}^1, \mathbf{U}^2, \mathbf{U}^3\}$ are {'1-2-1', '1-3-1', '1-4-1'}, respectively. Then, the models are constructed as the similar architecture of LeNet-5 [76]. The performance of BUnit-Net using single and hybrid units is reported in Table I. To avoid randomness, we provide the average accuracy after running the experiments for five times. We first build the network separately with three kinds of units. For hybrid units, one of the three units is randomly selected to stack for each unit position in stacked layers. To exclude fortuity in random selection, we perform the evaluation using hybrid units three times. Compared with single unit, hybrid units exhibit the potential to help improve the accuracy, when maintaining the FLOPs and parameters at the same compression level. For example, we adjust the numbers of three units stacked in Hybrid[2] to keep the FLOPs and parameters the same as using single unit $\mathbf{U}^2$, it is observed that model with hybrid units can achieve higher accuracy. The results on $\mathbf{U}^1$ and $\mathbf{U}^2$ also indicate the possibility that units with fewer FLOPs and parameters can still bring better accuracy. Therefore, these observations motivate us to explore more effective units and hybrid combinations.

2) Exploration of Different Units: For a BUnit-Net with high performance, one of the most important components is the designed basic unit. In this section, we try to explore the effect and ability of various units on extracting features through a series of initial experiments. We first design six kinds of units (i.e., Unit A-F) with different numbers of input, hidden and output node as shown in Fig. 3, and build a toy network by randomly selecting and stacking them. Specifically, the toy models are composed of two normal layers and two stacked layers, followed by a fully-connected layer as the classifier. Each stacked layer contains 20 units (numbered from Unit_1 to Unit_20) randomly selected from Unit A-F. In the exploration on basic units, the models are trained on CIFAR-10 dataset for 200 epochs with initial learning rate of 0.001. Other setting is the same as the extensive experiments on CIFAR-10. We conduct ten random experiments and the amount of different units are recorded in Table II. We also report the evaluation of toy model including accuracy, number of FLOPs and parameters for illustration. For instance, Model No.7 obtain the highest accuracy of 72.21% with 93.29 k parameters. However, Model No.10 occupies more FLOPs and parameters while the accuracy is still lower than No.7. The comparison of Model No.3 and No.4 also reflects this phenomenon. It implies that more FLOPs and parameters are helpless to improve the accuracy of the model, or even hurt the performance, which is also verified in previous experiments. The results also indicate that the number of different units has a great impact on the performance of BUnit-Net, which inspires us to

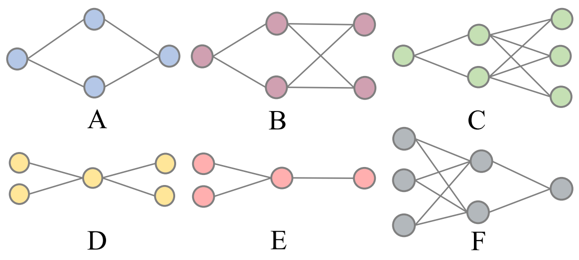

Fig. 3. Six kinds of units with different input, hidden and output nodes. The units in toy models are randomly selected from Unit A-F.

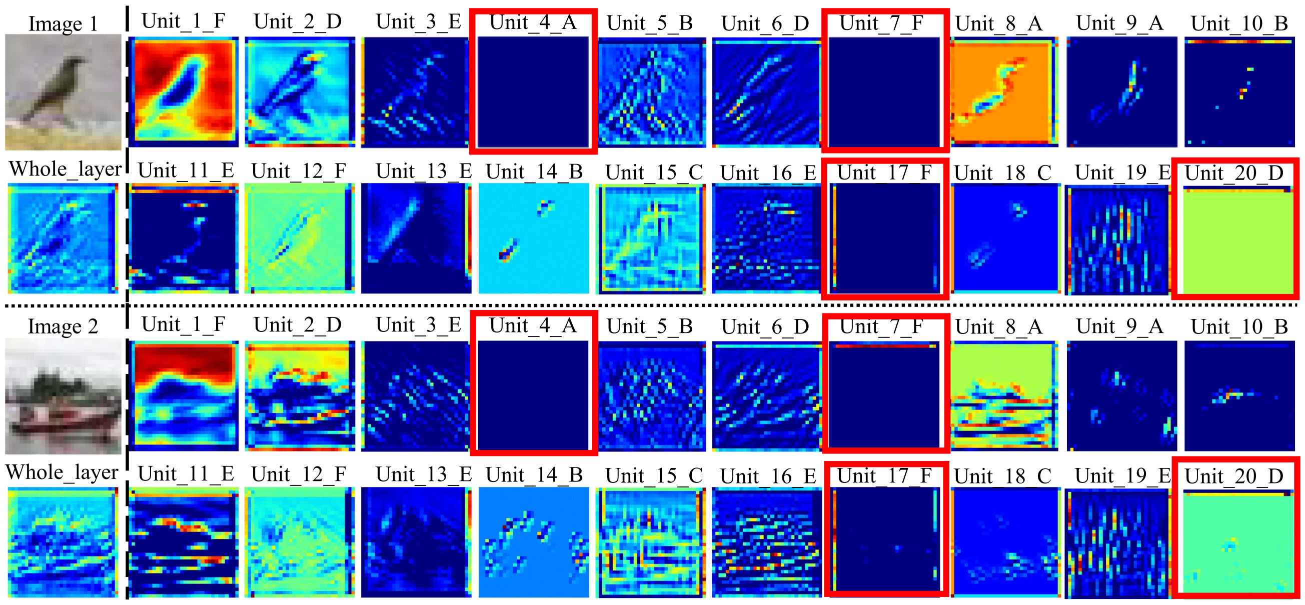

Fig. 4. Visualization results of each unit in the first stacked layer. Image 1 and Image 2 are the original input images. "Whole_layer" refers to the output feature map of the entire stacked layer, that is, the concatenation of each unit output. The structures of each unit are also recorded, for example, "Unit_1_F" means that F is selected as the first unit in this stacked layer.

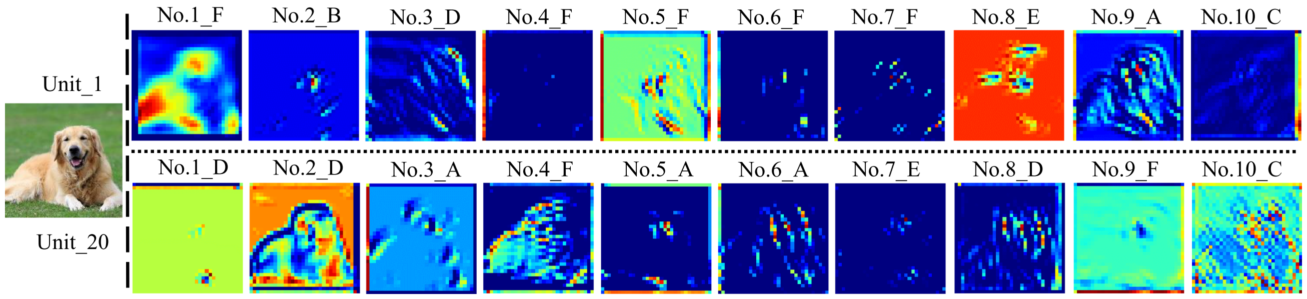

Fig. 5. Visualization results of the first and last unit output in the first stacked layer. The top line is the feature maps of the first unit (Unit_1) while the bottom line is the last unit (Unit_20). The structures of each unit in the ten experiments are recorded, for example, "No.1_F" means that the first unit in the first stacked layer of No.1 Model is F.

find the optimal combinations and adjust the model structured during training.

For further exploration of the learning ability of different units, we also visualize the feature maps of each unit in "Stacked-layer 1" of the compact network. We first report the visualization results of Model No.1 using 10 different images that belong to 10 classes of CIFAR-10. According to the results, it is worth highlighting that some of the units fail to learn useful information no matter what image is input. Take the class of bird and ship as examples in Fig. 4, the feature maps of some units such as Unit_4_A, Unit_7_F, Unit_17_F and Unit_20_D are almost a single color, meaning that the units miss valuable information even after training. This promotes us to replace these useless units with more powerful ones to improve the accuracy.

Although some units fail to extract the important features in Fig. 4, it is hard to conclude that these units have poor learning ability because the feature map is also closely related to the input. For example, Unit_6 has the same structure (i.e., D) as Unit_20 while it manages to learn more texture orientation and marginal features. Therefore, the position of unit also plays an essential role in improving the performance of BUnit-Net. To evaluate the effect of different units, we visualize the feature maps of the first and last unit in Stacked-layer 1 whose inputs are consistent with each other, the results of 10 experiments (Model No.1-No.10) are shown in Fig. 5. For the single-input units (i.e., A-C), it can be noted that the features of an object obtained by A with single output are more obvious than B and C, such as No.9_A of Unit_1 and No.6_A of Unit_20. However, for the two-input units (i.e., D and E), D with two outputs extracts more useful information than E with only one output in most cases, like No.3_D of Unit_1 and No.2_D of Unit_20. These findings illustrate that units with the same number of input and output nodes are prone to extract valuable features more effectively, providing a path for designing powerful units in the future.

D. MNIST

On MNIST, we choose the MLP with two hidden layers as baseline, where the number of nodes in each layer is "784-500-300-10" as applied in [77]. Note that the MLP only

TABLE III
COMPARISON RESULTS ON MNIST USING MLP

Method	Acc(%)	FLOPs (M)	Param (M)	TCA	TSA
Baseline	92.65	1.09	0.55	1	1
SSL [78]	92.53	0.17	0.09	6.40	6.10
NS [79]	92.49	0.17	0.09	6.40	6.10
CAC [77]	92.72	0.17	0.09	6.42	6.12
Our	91.95	0.03	0.02	***36.06***	***27.29***

contains fully-connected layers, so the basic units also need to be consistent. Based on the observations in the initial exploration of basic units, we design the basic units with the same input and output nodes, that is, $c_{in} = c_{out} = 2$. The number of hidden nodes c_h is flexible to adjust the compression ratio. In Table III, we report the performance of BUnit-Net on MNIST using MLP. In addition to the widely used metrics like accuracy, number of FLOPs and parameters, we also report the two proposed scores TSA and TCA ($w_1 = w_2 = 1$) for comparison with other methods. It is showed that we can achieve 97% FLOPS and 96% parameters reduction compared to baseline model, with only 0.7% accuracy loss. Besides, we can also obtain much higher TCA and TSA scores compared with the other three pruning methods. Thus, the results indicate that our proposed BUnit-Net can work well on MLP.

E. CIFAR-10 and CIFAR-100

On CIFAR dataset, BUnit-Net is built as VGG-16, ResNet-20 and MobileNetV2 backbones. For the two proposed scores, we set three pairs of scaling factors $\{w_1 = w_2 = 1\}$, $\{w_1 = 1, w_2 = 4\}$ and $\{w_1 = 4, w_2 = 1\}$ for different focuses. The theoretical speedup computed by FLOPs ratio is also reported.

1) CIFAR-10: The comparison results on CIFAR-10 are reported in Table IV. For VGG-16, we manage to save 79.87% FLOPs and 72.90% parameters with only 0.43% accuracy loss. Besides, BUnit-Net gets the highest TCA scores regardless of focusing on accuracy or flops, showing the efficiency of computational consumption. On ResNet-20, our proposed BUnit-Net can reduce around half of the FLOPs and parameters, where the accuracy loss of 0.58% is still acceptable. It is encouraging that the two scores are both slightly higher than other competitors except for when focusing more on accuracy, indicting better utilization of computation and storage resources. For ResNet-style, we also try to replace the layers in ResNet-20 with our stacked layers in BUnit-Net apart from the first and last blocks (represented as the symbol 'r'). Under this replacing strategy, we can prune 40% FLOPs and 32% parameters of baseline, where the accuracy gap can be narrowed to 0.23%.

2) CIFAR-100: In Table V, our BUnit-Net obtain an accuracy of 72.51% with 73.05 FLOPs and 70.26 parameters reduction when applying VGG-16. And the TSA scores are also larger than other state-of-the-art methods under different settings of scaling factors. Although the accuracy drop is more than 1%, it can be reduced by adjusting the hidden nodes in basic units or utilizing the same replacing strategy as ResNet. On ResNet-20, the advantage of BUnit-Net lies on saving computation resources because it leads to higher TCA scores.

In terms of latency on CPU and GPU, we conduct a comparison study between the proposed method with the baseline models. The results are presented in Table VII. Although the actual speedup ratio is lower than the theoretical value due to the computing and concatenation of each unit output in Algorithm 1, the proposed method still outperforms the baseline models with lower latency. This finding implies that the proposed method can help improve efficiency in real-life applications.

On the whole, the results on CIFAR dataset verify the flexibility and effectiveness of the proposed BUnit-Net on network compression, demonstrating that BUnit-Net can produce a more compressed model with comparable accuracy.

Lightweight Models: In addition to classical CNNs like VGG and ResNet, we also evaluate BUnit-Net on lightweight models, such as MobileNetV2, ShuffleNetV2 and EfficientNet, that are also popular in recent years. In MobileNet, there exist special 1×1 point-wise and 3×3 depth-wise convolution operations, where the group number in depth-wise convolution equals to the number of input channels. Our BUnit-Net can easily deal with these special operations by adjusting the parameters in basic units such as the kernel size and group number. The results are summarized in Tables IV and V. It can be seen that our BUnit-Net can also be implemented successfully as lightweight models. For instance, we get an accuracy of 91.58% on CIFAR-10, although there is a 0.62% loss compared to baseline, it is encouraging that the FLOPs and parameters can be reduced 38% and 36% respectively. Moreover, the TCA score is somewhat higher than other competitors. On CIFAR-100, the accuracy gap between BUnit-Net and baseline is relatively large. However, it manages to save 62% FLOPs and 60% parameters, and the performance scores of TCA and TSA are still larger when setting $w_1 = 1$ and $w_2 = 4$. With respect to other efficient architectures, we also apply our stack strategy on ShuffleNetV2 and EfficientNet-B0. Specifically, we replace parts of the convolutional layers in the basic block of ShuffleNetV2 and SE modules of EfficientNet-B0. The results are provided in Table VIII, showing that the proposed stack strategy can help achieve further compression with comparable accuracy. Overall, these results demonstrate that BUnit-Net can work efficiently on lightweight model.

Convergence: On CIFAR dataset, we investigated the convergence of BUnit-Net as illustrated in Fig. 6. Fig. 6(a) and (b) display the convergence curve on CIFAR-100 using ResNet-18. It is worth noting that BUnit-Net can almost reach an optimal status after certain training epochs. Besides, the training process of BUnit-Net is more stable though the maximum accuracy is a bit lower than baseline. On CIAFR-10 using VGG-16 in Fig. 6(c) and (d), our BUnit-Net is trained to be converged in fewer epochs compared with original VGG network, providing the potential possibility of acceleration.

F. Tiny-ImageNet and ImageNet

1) Tiny-ImageNet: We apply VGG-19 on Tiny-ImageNet and the results are recorded in Table VI(a). Compared with

TABLE IV
COMPARISON RESULTS ON CIFAR-10 UNDER DIFFERENT NETWORK BACKBONES

Method	Acc(%)			FLOPs↓(%)	Param↓(%)	Speedup	$w_1 = w_2 = 1$		$w_1 = 1, w_2 = 4$		$w_1 = 4, w_2 = 1$	
	Baseline	Compressed	Acc↓				TCA	TSA	TCA	TSA	TCA	TSA
VGG-16 (FLOPS: 77.95M Params: 14.91M)												
VP [26]	93.25	93.18	0.07	39.10	73.34	1.64×	1.48	2.08	1.47	2.08	4.77	18.78
GAL [80]	93.96	92.03	1.93	39.60	77.57	1.66×	1.46	2.13	1.37	2.0	4.78	21.81
HRank [81]	93.96	93.43	0.53	53.59	82.90	2.15×	1.70	***2.28***	1.67	2.24	8.48	***27.39***
DMCP [73]	92.38	92.21	0.17	25.05	69.75	1.33×	1.28	2.01	1.28	1.99	2.72	16.25
PFEC [29]	92.38	91.85	0.56	13.89	40.34	1.16×	1.14	1.49	1.12	1.46	1.73	4.99
DCP [82]	93.98	94.29	-0.31	50.08	48.29	2.0×	1.66	1.63	1.67	1.64	7.44	6.92
CHIP [42]	93.96	93.86	0.10	58.10	81.60	2.39×	1.79	2.26	1.78	***2.25***	10.21	26.13
FPGM [27]	93.58	93.23	0.35	35.90	-	1.56×	1.43	-	1.41	-	4.19	-
DANL [75]	93.77	93.53	0.22	37.75	75.10	1.61×	1.45	2.11	1.44	2.10	4.52	20.11
DualConv [74]	93.95	93.61	0.34	75.95	74.27	4.16×	2.13	2.09	2.11	2.07	20.79	19.44
Ours	93.25	92.82	0.43	79.87	72.90	4.97×	***2.21***	2.06	***2.18***	2.04	***24.29***	18.38
ResNet-20 (FLOPS: 94.18M Params: 8.52M)												
FPGM [27]	92.20	91.99	0.21	54.00	-	2.17×	1.71	-	***1.70***	-	8.65	-
SRR-GR [83]	92.27	92.48	-0.21	45.80	-	1.85×	1.58	-	1.60	-	6.26	-
SFP [84]	92.20	91.20	1.00	29.30	-	1.41×	1.33	-	1.28	-	3.19	-
VP [26]	92.01	91.66	0.35	16.47	20.61	1.20×	1.17	1.22	1.16	1.21	1.93	2.27
Ours('r')	95.10	94.87	0.23	40.34	32.67	1.68×	1.49	1.38	1.48	1.37	5.01	3.69
Ours	95.10	94.52	0.58	55.02	49.35	2.22×	***1.72***	***1.63***	1.69	***1.60***	***8.98***	***7.16***
MobileNetV2 (FLOPS: 134.95M Params: 2.25M)												
FPGM [27]	92.20	90.56	1.64	30.56	-	1.44×	1.33	-	1.26	-	3.34	-
DCP [82]	94.47	94.04	0.43	27.07	23.59	1.37×	1.30	1.26	1.29	1.24	2.94	2.56
DANL [75]	94.31	94.17	0.14	33.65	37.83	1.51×	1.40	1.46	1.39	1.45	3.84	4.53
FLGC [85]	94.31	94.11	0.20	13.35	48.70	1.16×	1.14	***1.62***	1.13	***1.61***	1.70	***7.0***
DualConv [74]	91.99	90.71	0.28	35.59	38.82	1.55×	1.41	1.45	1.35	1.39	4.09	4.66
Ours	92.20	91.58	0.62	38.56	36.81	1.63×	***1.46***	1.44	***1.43***	1.41	***4.64***	4.33

TABLE V
COMPARISON RESULTS ON CIFAR-100 UNDER DIFFERENT NETWORK BACKBONES

Method	Acc(%)			FLOPs↓(%)	Param↓(%)	Speedup	$w_1 = w_2 = 1$		$w_1 = 1, w_2 = 4$		$w_1 = 4, w_2 = 1$	
	Baseline	Compressed	Acc↓				TCA	TSA	TCA	TSA	TCA	TSA
VGG-16 (FLOPS: 78.01M Params: 14.96M)												
VP [26]	73.26	73.33	-0.07	18.05	37.87	1.22×	1.20	1.46	1.20	1.47	2.06	4.55
NCP [41]	72.21	72.01	0.20	37.38	64.84	1.60×	1.45	1.91	1.44	1.89	4.45	13.34
DMCP [73]	70.11	69.32	0.79	5.46	29.96	1.06×	1.04	1.33	1.01	1.29	1.23	3.28
PFEC [29]	70.11	68.82	1.29	18.15	4.60	1.22×	1.18	1.03	1.11	0.97	2.03	1.18
DualConv [74]	72.41	71.52	0.89	80.33	37.08	5.08×	***2.21***	1.43	***2.13***	1.38	***24.55***	4.35
Ours	73.55	72.51	1.04	73.05	70.26	3.71×	2.05	***1.99***	1.96	***1.91***	18.32	***16.38***
ResNet-20 (FLOPS: 94.19M Params: 8.54M)												
TAS [86]	68.69	68.90	-0.21	44.0	-	1.79×	1.56	-	1.57	-	5.83	-
FPGM [27]	67.62	66.86	0.76	42.20	-	1.73×	1.51	-	1.46	-	5.35	-
SFP [84]	68.82	67.91	0.91	28.15	-	1.39×	1.31	-	1.26	-	3.04	-
DI-unif [87]	68.50	69.30	-0.80	54.30	63.0	2.19×	1.74	***1.90***	1.80	***1.97***	8.88	***12.57***
Ours	77.32	76.48	0.83	57.46	62.56	2.35×	***1.76***	1.85	***1.70***	1.79	***9.85***	12.08
MobileNetV2 (FLOPS: 135.19M Params: 2.37M)												
FPGM [27]	72.58	68.44	4.14	29.73	-	1.42×	1.27	-	1.07	-	3.1	-
DualConv [74]	68.54	67.19	1.35	63.82	65.98	2.76×	***1.86***	***1.90***	1.75	1.79	***12.59***	***13.73***
Ours	72.58	71.54	1.04	62.61	60.48	2.67×	1.84	1.88	***1.77***	***1.80***	12.06	13.0

baseline, the accuracy drop of BUnit-Net is relatively large but it can save more than 63% FLOPs and 54% parameters. As analysed in the previous section, we can improve the accuracy by adjusting the hyperparameters to add nodes in stacked layers or applying a different replacing strategy. Compared with other methods, it is reasonable to select DMCP [73] with higher TSA and lower accuracy drop if we pay more attention to accuracy (i.e., $w_1 = 1, w_2 = 4$). However, BUnit-Net can still obtain a relatively larger score both on TCA and TSA when focusing on resource consumption (i.e., $w_1 = 4, w_2 = 1$).

TABLE VI
COMPARISON RESULTS ON TINY-IMAGENET AND IMAGENET UNDER VGG-19 AND RESNET-34/50 BACKBONES

Method	Acc(%) top-1 (top-5)			FLOPs↓(%)	Param↓(%)	Speedup	$w_1 = w_2 = 1$		$w_1 = 1, w_2 = 4$		$w_1 = 4, w_2 = 1$	
	Baseline	Compressed	Acc↓				TCA	TSA	TCA	TSA	TCA	TSA
(a) Tiny-ImageNet VGG-19 (FLOPS: 9.50G Params: 20.32M)												
PFEC [29]	59.05	58.17	0.88	5.91	27.85	1.06×	1.05	1.30	1.0	1.24	1.25	3.0
DMCP [73]	59.05	58.00	1.05	18.51	51.57	1.23×	1.18	1.65	1.12	***1.56***	2.06	***7.73***
Ours	55.65	53.54	2.11	63.42	54.63	2.80×	***1.82***	***1.66***	***1.62***	1.48	***12.17***	***8.56***
(b) ImageNet ResNet-34 (FLOPS: 7.34G Params: 21.80M)												
PFEC [29]	73.23 (-)	72.17 (-)	1.06 (-)	24.20	-	1.32×	1.26	-	1.20	-	2.59	-
NISP [88]	73.31 (-)	73.02 (-)	0.28 (-)	27.32	27.14	1.38×	1.31	1.31	1.29	1.29	2.97	2.98
Taylor [89]	73.31 (-)	72.83 (-)	0.48 (-)	22.43	21.10	1.28×	1.24	1.23	1.22	1.20	2.44	2.31
FPGM [27]	73.92 (91.62)	72.54 (91.13)	1.38 (0.49)	41.10	-	1.70×	1.48	-	***1.40***	-	5.08	-
Ours	73.31 (91.45)	71.51 (90.54)	1.80 (0.91)	42.05	35.31	1.73×	***1.49***	***1.39***	1.38	1.29	***5.25***	***4.01***
(c) ImageNet ResNet-50 (FLOPS: 8.22G Params: 25.56M)												
ThiNet-70 [44]	75.30 (92.20)	74.03 (92.11)	1.27 (0.09)	36.79	33.72	1.58×	1.42	1.38	1.35	1.31	4.28	3.79
Taylor [89]	76.15 (-)	75.48 (-)	0.7 (-)	34.96	30.08	1.54×	1.41	1.34	1.37	1.30	4.01	3.30
FPGM [27]	76.15 (92.87)	75.59 (92.27)	0.56 (0.60)	42.2	37.5	1.73×	1.51	1.44	1.48	1.41	5.37	4.45
GAL [80]	76.15 (92.87)	71.95 (90.94)	4.20 (1.93)	43.03	16.86	1.76×	1.46	1.12	1.23	0.95	5.29	1.86
HRank [81]	76.15 (92.87)	74.98 (92.33)	1.17 (0.54)	43.77	36.47	1.78×	1.53	1.42	1.46	1.35	5.67	4.24
AutoPruner [48]	76.15 (92.87)	74.76 (92.15)	1.39 (0.72)	51.10	-	2.04×	***1.64***	-	1.55	-	***7.58***	-
CHIP [42]	76.15 (92.87)	76.30 (93.02)	-0.15 (-0.15)	44.80	40.80	1.81×	1.57	***1.51***	***1.58***	***1.52***	6.01	5.12
DANL [75]	75.19 (92.56)	74.07 (92.02)	1.12 (0.54)	49.39	39.96	1.98×	1.61	1.47	1.54	1.40	7.10	4.87
DualConv [74]	74.27 (92.09)	74.09 (91.80)	0.18 (0.29)	17.60	17.21	1.21×	1.19	1.18	1.18	1.18	2.02	1.99
Ours('r')	77.09 (93.38)	77.0 (93.45)	0.09 (-0.07)	25.0	15.54	1.33×	1.28	1.17	1.28	1.16	2.72	1.86
Ours	77.09 (93.38)	75.33 (92.60)	1.76 (0.78)	43.80	43.67	1.78×	1.51	***1.51***	1.41	1.41	5.64	***5.61***

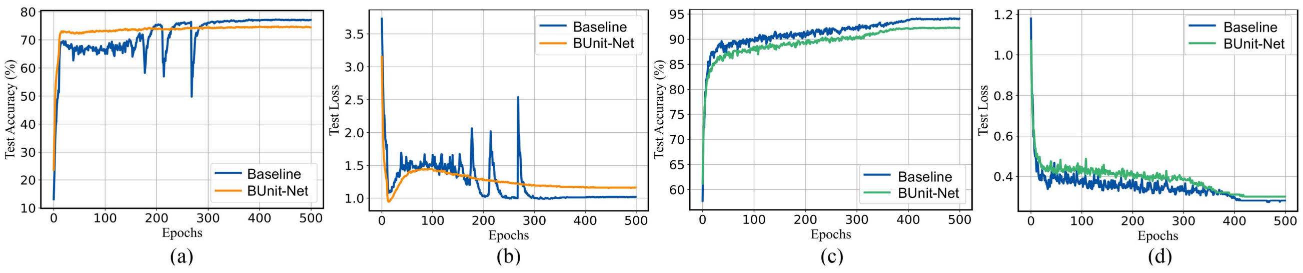

Fig. 6. Convergence Curve. (a)/(b): Test Accuracy/Loss on CIFAR-100 using ResNet-18; (c)/(d): Test Accuracy/Loss on CIFAR-10 using VGG-16.

TABLE VII
COMPARISON OF INFERENCE TIME FOR 32×32 IMAGES

Model	Inference Time (ms/image)	
	Intel Core i7-8700	NVIDIA Tesla P40
VGG-16	871	235
Ours	643	196
ResNet-18	792	437
Ours	649	356

TABLE VIII
COMPARISON WITH SHUFFLENET AND EFFICIENTNET

Dataset	Model	FLOPs (M)	Param (M)	Acc (%)
CIFAR-100	ShuffleNetV2	1.36	91.30	71.83
	Ours	1.25	85.27	71.02
	Ours*	1.07	78.40	70.94
ImageNet	Efficient-B0	0.78	5.32	71.24
	Ours	0.64	4.85	70.17

$*$ We replace the layers in different numbers of blocks to obtain varying compression ratio.

2) ImageNet: On large-scale ImageNet, we select ResNet-34 and ResNet-50 as the baseline model and compare the performance of our BUnit-Net with state-of-the-art pruning methods in Table VI(b). On ResNet-34, it is inspiring that our method achieves 71.51% top-1 accuracy and 35% parameters degradation, with higher TCA and TSA scores than other methods under most of the settings. Also, we manage to reduce the amount of FLOPs by around 42% although the TCA score is slightly smaller than FPGM [27] when considering accuracy performance. On ResNet-50, our model reduces more parameters and obtain higher TSA score compared with the other methods while maintaining the accuracy, resulting in 43.67% reduction in terms of number of parameters. These results demonstrate that the proposed BUnit-Net is efficient to reduce computation and memory consumption on complex tasks, meanwhile maintaining a comparable accuracy.

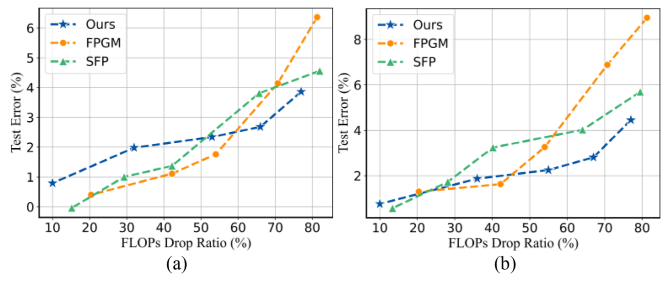

Fig. 7. Trade-off curve between test error and FLOPs drop ratio of BUnit-Net on (a) CIFAR-10 and (b) CIFAR-100. It can be noted that our method performs better under larger compression ratio.

G. Ablation Study

Varying FLOPs and Parameters: To study the performance under different compression ratios, we apply different units to adjust FLOPs and parameters for compact BUnit-Net under ResNet-18 backbone on CIFAR-10 and CIFAR-100. The results in Fig. 7 help comprehensively understand the trade-off of the proposed BUnit-Net. The four curves reveal the same trend that BUnit-Net will perform worse as the numbers of FLOPs and parameters decrease. This trend is also consistent with our experience that a more compact model will sacrifice more accuracy. On CIFAR-100, when we tune the remaining FLOPs from 21% to 30% and corresponding parameters from 10% to 30%, the compression ratio drops a little but the accuracy increases drastically from 72.87% to 74.51%. The same observation is also verified on CIFAR-10. It is reasonable that continuing to compress will result in a sharp drop in accuracy after the model is compressed to a certain extent. Thus, to achieve a better trade-off between compression and accuracy, we can empirically control the remaining FLOPs and parameters as more than 20% except for the simple task such as MNIST when applying BUnit-Net for network compression.

V. Conclusion

Limitations: Although our method has been proved to be effective on compressing networks, there are still some limitations. First, the basic units are currently designed and stacked randomly. It is therefore desirable to find the optimal structure and combination which can further enhance the accuracy. Second, the independence of each unit in the stacked layer may hinder the communication of information that will also influence the performance. In the future, we will make more exploration and combine our method with other techniques such as NAS and channel shuffle to obtain better performance.

Conclusions: This paper is the first attempt to construct human-designed compact networks called BUnit-Net, which combines a number of designed basic function units to generate stacked layers. Compared with standard dense layers, the stacked layers contain much fewer parameters and FLOPs due to the independence of each unit so that BUnit-Net can achieve compression. We have provided the analysis of the memory and computation cost of BUnit-Net in detail. Also, several initial experiments have explored the powerful basic units. For comparison with different compression methods, we have introduced two unified indicators to measure the trade-off between accuracy and compression ratio. The comparative studies have shown that the proposed BUnit-Net can lead to better compression with comparable accuracy compared with the state-of-the-art pruned and lightweight models. This implies the flexibility and efficacy of BUnit-Net, providing a new promising way for network compression.

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] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.
[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.
[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.
[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.
[7] J. Cheng, P.-S. Wang, G. Li, Q.-H. Hu, and H.-Q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Front. Inf. Technol. Electron. Eng.*, vol. 19, no. 1, pp. 64–77, 2018.
[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, *arXiv:1510.00149*.
[9] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," 2017, *arXiv:1702.04008*.
[10] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
[11] 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. 2016.
[12] S. Li, E. Hanson, H. Li, and Y. Chen, "PENNI: Pruned kernel sharing for efficient CNN inference," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 5863–5873.
[13] X. Jin et al., "Knowledge distillation via route constrained optimization," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1345–1354.
[14] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," 2014, *arXiv:1412.6550*.
[15] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[16] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[17] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.
[18] D. Lepikhin et al., "GShard: Scaling giant models with conditional computation and automatic sharding," 2020, *arXiv:2006.16668*.
[19] 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.
[20] M. A. Carreira-Perpinán and Y. Idelbayev, ""Learning-compression" algorithms for neural net pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8532–8541.
[21] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, 2016.
[22] X. Zhou et al., "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2018, pp. 15–28.
[23] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs," in *Proc. IEEE Annu. Int. Symp. Field- Program. Custom Comput. Machines*, 2019, pp. 17–25.

[24] W. You and C. Wu, "RSNN: A software/hardware co-optimized framework for sparse convolutional neural networks on FPGAs," *IEEE Access*, vol. 9, pp. 949–960, 2020.
[25] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-Z. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–8.
[26] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2780–2789.
[27] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4340–4349.
[28] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 1389–1397.
[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.
[30] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," in *Proc. Mach. Learn. Syst.*, vol. 2, pp. 129–146, 2020.
[31] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits, 1998," 1998. [Online]. Available: http://yann. lecun. com/ exdb/mnist
[32] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Toronto Univ., Tech. Rep., 2009, pp. 1–60.
[33] L. Hansen, "Tiny ImageNet challenge submission," *Course Project CS 231N*, Stanford Univ., 2015. [Online]. Available: http://tiny-imagenet. herokuapp.com
[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[37] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 598–605.
[38] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Conf. Neural Netw.*, 1993, pp. 293–299.
[39] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1387–1395.
[40] T. Zhang et al., "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 184–199.
[41] Y. Hu, S. Sun, J. Li, X. Wang, and Q. Gu, "A novel channel pruning method for deep neural network compression," 2018, *arXiv:1805.11394*.
[42] Y. Sui, M. Yin, Y. Xie, H. Phan, S. Aliari Zonouz, and B. Yuan, "CHIP: Channel independence-based pruning for compact neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 24604–24616.
[43] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 5058–5066.
[44] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, Oct. 2019.
[45] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal SGD for pruning very deep convolutional networks with complicated structure," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4943–4953.
[46] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li, "Toward compact ConvNets via structure-sparsity regularized filter pruning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 574–588, Feb. 2020.
[47] Z. Chen, Y. Li, S. Bengio, and S. Si, "You look twice: GaterNet for dynamic filter selection in CNNs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9172–9180.
[48] J.-H. Luo and J. Wu, "AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognit.*, vol. 107, 2020, Art. no. 107461.
[49] A. Alqahtani, X. Xie, M. W. Jones, and E. Essa, "Pruning CNN filters via quantifying the importance of deep visual representations," *Comput. Vis. Image Understanding*, vol. 208, 2021, Art. no. 103220.
[50] S.-K. Yeom et al., "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognit.*, vol. 115, pp. 1–14, 2021.
[51] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS One*, vol. 10, no. 7, 2015, Art. no. e0130140.
[52] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 784–800.
[53] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 4876–4883.
[54] Z. Yang et al., "CARS: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1829–1838.
[55] Z. Liu et al., "MetaPruning: Meta learning for automatic neural network channel pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3296–3305.
[56] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.
[57] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[58] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "EspNetV2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9190–9200.
[59] Y. Jeon and J. Kim, "Constructing fast network through deconstruction of convolution," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5955–5965.
[60] W. Chen, D. Xie, Y. Zhang, and S. Pu, "All you need is a few shifts: Designing efficient convolutional neural networks for image classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7241–7250.
[61] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 6105–6114.
[62] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.
[63] B. Wu et al., "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10734–10742.
[64] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to 1 or -1," 2016, *arXiv:1602.02830*.
[65] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2016, pp. 525–542.
[66] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016, *arXiv:1605.04711*.
[67] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*.
[68] 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*.
[69] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[70] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4320–4328.
[71] Y. Guan et al., "Differentiable feature aggregation search for knowledge distillation," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 469–484.
[72] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[73] Z. Xu, J. Sun, Y. Liu, and G. Sun, "An efficient channel-level pruning for CNNs without fine-tuning," in *Proc. Int. Joint Conf. Neural Netw.*, 2021, pp. 1–8.
[74] J. Zhong, J. Chen, and A. Mian, "DualConv: Dual convolutional kernels for lightweight deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 01, 2022, doi: 10.1109/TNNLS.2022.3151138.
[75] Q. Guo, X.-J. Wu, J. Kittler, and Z. Feng, "Differentiable neural architecture learning for efficient neural networks," *Pattern Recognit.*, vol. 126, 2022, Art. no. 108448.
[76] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[77] Z. Chen, T.-B. Xu, C. Du, C.-L. Liu, and H. He, "Dynamical channel pruning by conditional accuracy change for deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 799–813, Feb. 2021.
[78] 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. 2082–2090.
[79] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2736–2744.
[80] S. Lin et al., "Towards optimal structured CNN pruning via generative adversarial learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2790–2799.
[81] M. Lin et al., "HRank: Filter pruning using high-rank feature map," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1529–1538.
[82] J. Liu et al., "Discrimination-aware network pruning for deep model compression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 4035–4051, Aug. 2022.
[83] Z. Wang, C. Li, and X. Wang, "Convolutional neural network pruning with structural redundancy reduction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14913–14922.
[84] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," 2018, *arXiv:1808.06866*.
[85] X. Wang, M. Kan, S. Shan, and X. Chen, "Fully learnable group convolution for acceleration of deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9049–9058.
[86] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 69.
[87] Z. Hou and S.-Y. Kung, "A discriminant information approach to deep neural network pruning," in *Proc. Int. Conf. Pattern Recognit.*, 2021, pp. 9553–9560.
[88] R. Yu et al., "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9194–9203.
[89] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11264–11272.

Weichao Lan received the BS degree in electronics and information engineering from Sichuan University, Chengdu, China, in 2019. She is currently working toward the PhD degree with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, under the supervision of Prof. Yiu-ming Cheung. Her present research interests include network compression and acceleration, lightweight models.

Yiu-Ming Cheung (Fellow, IEEE) received the PhD degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. He is currently a chair professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. His research interests include machine learning, data science, visual computing, and optimization. He is a fellow of American Association for the Advancement of Science (AAAS), the Institution of Engineering and Technology (IET), and British Computer Society (BCS). Also, he is an awardee of RGC senior research fellow. He is the editor-in-chief of IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, and is an associate editor for several prestigious journals, including IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS, PATTERN RECOGNITION, PATTERN RECOGNITION LETTERS, NEUROCOMPUTING, to name a few. For more details, please refer to: https://www.comp.hkbu.edu.hk/∼ymc.

Juyong Jiang received the BS degree in computer science and technology with Hohai University, Nanjing, China, in 2020. Currently, he is working toward the PhD degree in data science and analytics thrust with the Hong Kong University of Science and Technology (Guangzhou), under the supervision of Prof. Sunghun Kim. He was a research assistant with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, from 2021 to 2022. His current research interests include data mining, large language models, and code generation.

Zhikai Hu received the BS degree in computer science from China Jiliang University, Hangzhou, China, in 2015, and the MS degree in computer science from Huaqiao University, Xiamen, China, in 2019. He is currently working toward the PhD degree with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, under the supervision of Prof. Yiu-ming Cheung. His present research interests include information retrieval, pattern recognition and data mining.

Mengke Li received the BS degree in communication engineering from Southwest University, Chongqing, China, in 2015, the MS degree in electronic engineering from Xidian University, Xi'an, China, in 2018, and the PhD degree from the Department of Computer Science, Hong Kong Baptist University, Hong Kong SAR, China, under the supervision of Prof. Yiu-ming Cheung, in 2022. She is currently an associate researcher with the Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Guangdong, China. Her current research interests include image restoration and imbalanced data learning.