Parameter-Efficient Masking Networks

Yue Bai[1,*] **Huan Wang**[1,4] **Xu Ma**[1] **Yitian Zhang**[1] **Zhiqiang Tao**[3] **Yun Fu**[1,2,4]
[1]Department of Electrical and Computer Engineering, Northeastern University
[2]Khoury College of Computer Science, Northeastern University
[3]School of Information, Rochester Institute of Technology
[4]AInnovation Labs, Inc.
Project Homepage: https://yueb17.github.io/PEMN

Abstract

A deeper network structure generally handles more complicated non-linearity and performs more competitively. Nowadays, advanced network designs often contain a large number of repetitive structures (e.g., Transformer). They empower the network capacity to a new level but also increase the model size inevitably, which is unfriendly to either model restoring or transferring. In this study, we are the first to investigate the representative potential of fixed random weights with limited unique values by learning diverse masks and introduce the Parameter-Efficient Masking Networks (PEMN). It also naturally leads to a new paradigm for model compression to diminish the model size. Concretely, motivated by the repetitive structures in modern neural networks, we utilize one random initialized layer, accompanied with different masks, to convey different feature mappings and represent repetitive network modules. Therefore, the model can be expressed as *one-layer* with a bunch of masks, which significantly reduce the model storage cost. Furthermore, we enhance our strategy by learning masks for a model filled by padding a given random weights vector. In this way, our method can further lower the space complexity, especially for models without many repetitive architectures. We validate the potential of PEMN learning masks on random weights with limited unique values and test its effectiveness for a new compression paradigm based on different network architectures. Code is available at https://github.com/yueb17/PEMN.

1 Introduction

Deep neural networks have emerged in several application fields and achieved state-of-the-art performances [9, 18, 34]. Along with the data explosion in this era, huge amount of data gathered to build network models with higher capacity [4, 8, 30]. In addition, researchers also pursue a unified network framework to deal with multi-modal and multi-task problems as a powerful intelligent model [30, 42]. All these trending topics inevitably require even larger and deeper network models to tackle diverse data flows, arising new challenges to compress and transmit models, especially for mobile systems.

Despite the success of recent years with promising task performances, advanced neural networks suffer from their growing size, which causes inconvenience for both model storage and transferring. To reduce the model size of a given network architecture, neural network pruning is a typical technique [26, 24, 13]. Pruning approaches remove redundant weights using designed criteria and the pruning operation can be conducted for both pretrained model (conventional pruning: [14, 13]) and randomly initialized model (pruning at initialization: [25, 37]). Another promising direction is to obtain sparse network by dynamic sparse training [10, 29]. They jointly optimize network architectures and weights to find good sparse networks. Basically, these methods commonly demand regular training, and the final weights are updated by optimization algorithms like SGD automatically.

[*]Corresponding author: `bai.yue@northeastern.edu`

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Now that the trained weights have such a great representative capacity, one may wonder what is the potential of random and fixed weights or is it possible to achieve the same performance on random weights? If we consider a whole network, the answer is obviously negative as a random network cannot provide informative and distinguishable outputs. However, picking a subnetwork from a random dense network make it possible as feature mapping varies with changes of subnetwork structures. Then, the question has been updated as *what is the representative potential of random and fixed weights with selecting subnetwork structures?* Pioneer work LTH [11] shows the winning ticket exists in random network with good trainability but cannot be used directly without further training. Supermasks [44] enhances the winning ticket and enable it being usable directly. Recent work Popup [31] significantly improves subnetwork capacity from its dense counterpart by learning the masks using backpropagation. Following this insightful perspective, we further ask a question – *what is the maximum representative potential of a set of random weights?* In our work, we first make a thorough exploration of this scientific question to propose our Parameter-Efficient Masking Networks (PEMN). Then, leveraging on the PEMN, we naturally introduce a new network compression paradigm by combining a set of fixed random weights with a corresponding learned mask to represent the whole network.

We start with network architectures which recent popular design style, *i.e.*, building a small-scale encoding module and stacking it to obtain a deep neural network [9, 33, 32]. Based on this point, we naturally propose the *One-layer* strategy by using one module as a prototype and copy its parameter into other repetitive structures. More generally, we further provide two versions: *max-layer padding (MP)* and *random weight padding (RP)* to handle diverse network structures. Specifically, *MP* chooses the layer with the most number of parameters as the prototype and uses first certain parameters of prototype to fill in other layers. *RP* even breaks the constraint of network architecture. It samples a random vector with certain length as the prototype which is copied several times to fill in all layers based on their different lengths. *RP* is architecture-agnostic and can be seen as a most general strategy in our work. Three strategies are from specific to general manner and reduce the number of unique parameters gradually. We first employ these strategies to randomly initialize network. Then, we learn different masks to explore the random weights potential and positively answer the scientific question above. Leveraging on it, we propose a new network compression paradigm by using a set of random weights with a bunch of masks to represent a network model instead of restoring sparse weights for all layers (see Fig. 1). We conduct comprehensive experiments to explore the random weights representative potential and test the model compression performance to validate our paradigm. We summarize our contributions as below:

- We scientifically explore the representative potential of fixed random weights with limited unique values and introduce our Parameter-Efficient Masking Networks (PEMN). It leverages on learning different masks to represent different feature mappings.
- A novel network compression paradigm is naturally proposed by fully utilizing the representative capacity of random weights. We represent and restore a network based on a given random vector with a bunch of masks instead of retaining all the sparse weights.
- Extensive experimental results explore the random weights potential by using our PEMN and test the compression performance of our new paradigm. We expect our work can inspire more interesting explorations in this direction.

2 Related Works

2.1 Sparse Network Training

Our work is related to sparse network training. Conventional pruning techniques finetune the pruned network from pretrained models [13, 14] with various pruning criteria for different applications [26, 15, 16, 38, 36, 40, 41]. Instead of pruning a pretrained model, pruning at initialization [37] approaches attempt to find winning ticket from the random weight. Gradient information is considered to build pruning criteria in [25, 35]. Different from pruning methods above, sparse network training also can be conducted in a dynamic fashion. To name a few, Rigging the Lottery [10] edits the network connections and jointly updates the learnable weights. Dynamic Sparse Reparameterization [29] modifies the parameter budget among the whole network dynamically. Sparse Networks from Scratch [7] proposes a momentum based approach to adaptively grow weights and empirically

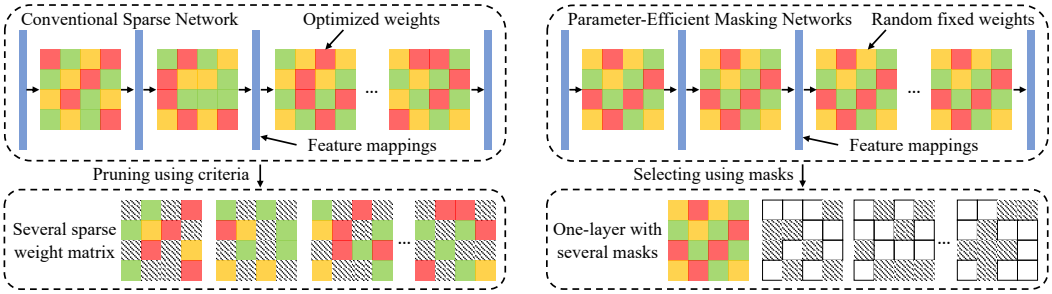


Figure 1: Comparison of different ways to represent a neural network. Different features mappings are shown as blue rectangles. Squares with different color patches inside serve as parameters of different layers. Left is the conventional fashion where weights are optimized and sparse structures are decided by certain criteria. Right is our PEMN to represent a network where the prototype weights are fixed and repetitively used to fill in the whole network and different masks are learned to deliver different feature mappings. Following this line, we explore the representative potential of random weights and propose a novel paradigm to achieve model compression by combining a set of random weights and a bunch of masks.

verifies its effectiveness. Most of the sparse network training achieve the network sparsity by keeping necessary weights and removing others, which reduces the cost of model storage and transferring. In our work, we propose a novel model compression paradigm by leveraging the representative potential of random weights accompanied with subnetwork selection.

2.2 Random Network Selection

Our work inherits the research line of exploring the representative capacity of random network. The potential of randomly initialized network is pioneeringly explored by the Lottery Ticket Hypothesis [11], and further investigated by [28, 2, 39]. It articulates that there exists a winning ticket subnetwork in a random dense network. This subnetwork can be trained in isolation and achieves comparable results with its dense counterpart. Moreover, the potential of the winning ticket is further explored in Supermasks [44]. It surprisingly discovers the subnetwork can be identified from dense network to obtain reasonable performance without training. It extends and proves the potential of subnetwork from good trainability to being used directly. More recently, the representative capacity of subnetworks is enhanced by Popup algorithm proposed by [31]. Based on random dense initialization, the learnable mask is optimized to obtain subnetwork with promising results. Instead of considering network with random weights, the network with the same shared parameters can also delivery representative capacity to some extent, which is investigated by Weight Agnostic Neural Network [12] and also inspires this research direction. We are highly motivated by these researches to validate how is the representative potential of random weights with limited unique values by learning various masks.

2.3 Weight Sharing

Our study is also related to several recent works about weight sharing. This strategy has been explored and analyzed in convolutional neural networks for their efficiency [19, 43]. In addition, several works are also proposed for efficient transformer architecture using weight sharing strategy [22, 5, 1]. There are two main differences between these works and our study: 1) They follow the regular optimization strategy to learn the weight in a recurrent fashion, which is closer to the recurrent neural network. Our work follows a different setting. We use fixed repetitive random weights to fill in the whole network and employs different masks to represent different feature mappings; 2) They mainly conduct cross-layer weight sharing for repetitive transformer structure. In our work, we explore the potential of random weight vector with limited length as much smaller repetitive granularity to fill in the whole network, which is more challenging than cross-layer sharing strategy.

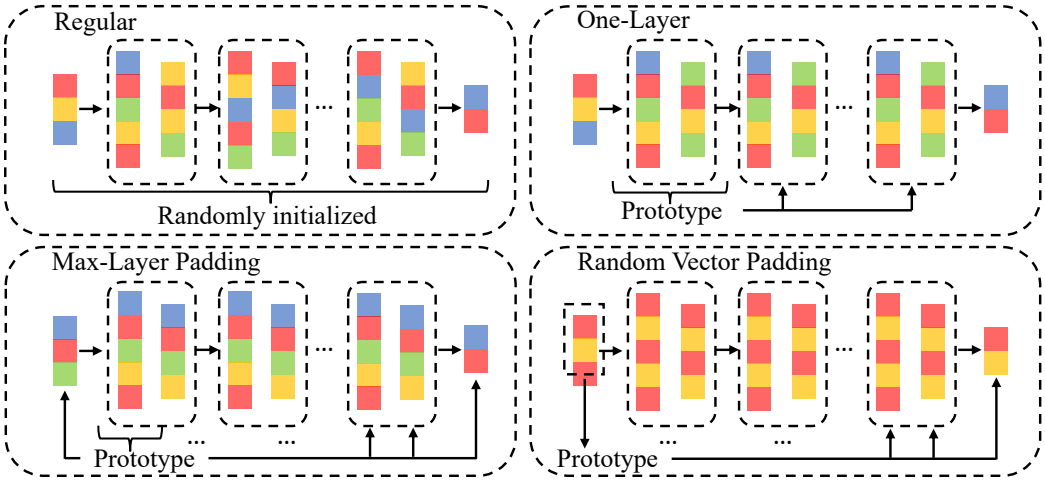


Figure 2: Illustrations of different strategies in PEMN to represent network structures. Compared with regular fashion where all parameters are randomly initialized, we provide three parameter-efficient strategies, *One-layer*, *Max-layer padding (MP)*, and *Random vector padding (RP)*, to fully explore the representative capacity of random weights.

3 Parameter-Efficient Masking Networks

3.1 Instinctive Motivation

Overparameterized randomly initialized neural network benefits network optimization to get higher performance. Inevitably, the trained network contains redundant parameters but can be further compressed, which defines the conventional neural network pruning. On the other side, the network redundancy also ensures a large random network contains a huge number of possible subnetworks, thus, carefully selecting a specific subnetwork should obtain promising performances. This point of view has been proved by [31, 44]. These works demonstrate the representative potential of certain subset combinations of a given random weights. Following this lane, we naturally ask a question: *what is the maximum representative potential of a set of random weights?* or in another word: *can we use random weights with limited unique values to represent a usable network?* We answer this question as positive and introduce our Parameter-Efficient Masking Networks (PEMN). Moreover, leveraging on 1) compared with trained network where the weight values cannot be predicted, we can pre-access the random weights before we select the subnetwork; 2) selected subnetwork can be efficiently represented by a bunch of masks, we can extremely reduce the network storage size and establish a new paradigm for network compression.

3.2 Sparse Selection

We follow [31] to conduct the sparse network selection. We start from a randomly initialized neural network consisting of L layers. For each $l \in \{1, 2, ..., L\}$, it has

$$\mathcal{I}_{l+1} = \sigma(\mathcal{F}[\mathcal{I}_l; w_l]), \tag{1}$$

where $\mathcal{I}_l$ and $\mathcal{I}_{l+1}$ are the input and output of layer l. σ is the activation. $\mathcal{F}$ represents the encoding layer such as convolutional or linear layer with parameter $w_l = \{w_l^1, w_l^2, ..., w_l^{d_l}\}$, where d_l is the parameter dimension of layer l. To perform the sparse selection, all the weights $w = \{w_1, w_2, ..., w_L\}$ are fixed and denoted as $\widetilde{w}$. To pick the fixed weights for subnetwork, each weight w_l^j is assigned a learnable element-wise score s_l^j to indicate its importance in the network. The Eq. 1 is rewrited as

$$\mathcal{I}_{l+1} = \sigma(\mathcal{F}[\mathcal{I}_l; w_l \odot h(s_l)]), \tag{2}$$

where $s_l = \{s_l^1, s_l^2, ..., s_l^{d_l}\}$ is the score vector and $h(\cdot)$ is the indicator function to create the mask. It outputs 1 when the value of s_l^j belongs to the top K% highest scores and outputs 0 for others, where K is predefined sparse selection ratio. Through optimizing s with fixed w, a subset of original dense

weights is finally selected. Since $h(\cdot)$ is a non-derivable function, the gradient of each s_l^j cannot be obtained directly. The straight-through gradient estimator [3] is applied to treat $h(\cdot)$ as identity function during gradient backwards pass. Formally, the gradient of s is approximately computed as

$$\widetilde{g}(s_l^j) = \frac{\partial \mathcal{L}}{\partial \widetilde{\mathcal{I}}_{l+1}} \frac{\partial \widetilde{\mathcal{I}}_{l+1}}{\partial s_l^j} \approx \frac{\partial \mathcal{L}}{\partial \mathcal{I}_{l+1}} \frac{\partial \mathcal{I}_{l+1}}{\partial s_l^j}, \tag{3}$$

where $\widetilde{\mathcal{I}}_{l+1} = \sigma(\mathcal{F}[\mathcal{I}_l; w_l \odot s_l])$, which is applied estimation. $\widetilde{g}(s_l^j)$ is approximately estimated gradient of weight score s_l^j. In this way, the dense network is randomly initialized but fixed, but one of its subnetwork can be selected using Backpropagation. In our work, we name this optimization process as *sparse selection*.

3.3 Parameter-Efficient Strategy

Following the logic of parameter-efficient exploration from specific to general scenario, PEMN utilizes three strategies to construct the whole network based on given random weights: 1) *One-layer* (Sec. 3.3.1), 2) *Max-layer padding (MP)* (Sec. 3.4), and 3) *Random vector padding (RP)* (Sec. 3.4), which are detailedly introduced below.

3.3.1 One-Layer

Sparse selection initializes a dense network as a pool to pick certain weights. It provides a novel direction to find admirable subnetwork without pre-training and pruning. However, with increasing of network scales, the cost of restoring and transfering a neural network grows rapidly. Noticed that more popular network structures follow a similar design style: *proposing a well-designed modeling block and stacking it several times to boost network capacity*, we are inspired to explore the feasibility of finding subnetworks by iteratively selecting different masks in a series of repetitive modules. Formally, a L-layer randomly initialized network $\mathcal{N}_L$ can be represented as a series of parameters:

$$\mathcal{N}_L : w = [w_1, w_2, ..., w_L]; w_l \in \mathbb{R}_l, \tag{4}$$

where $l \in \{1, 2, ..., L\}$ and w_l is used for various layers. $\mathbb{R}_l$ denotes different parameter spaces (e.g., $\mathbb{R}_l^{I \times O}$ for linear, $\mathbb{R}_l^{N \times H \times W}$ for CNN layer, where I/O are input/output dimensions and $N/H/W$ are CNN kernel dimensions). From shallow to deep layer, we first sample the *prototype* layers for the whole network with unique parameter spaces, represented as $w_{pro} = [w_{pro^1}, w_{pro^2}, ..., w_{pro^P}]$. For each w_{pro^p}, we use its parameters to replace its *target* layers w_{tar^p} which share the same parameter space:

Algorithm 1 One-Layer

Input: A random network with L-layer: $w = \{w_1, w_2, ..., w_L\}$
Output: A L-layer network filled by P prototype layers with parameters: $w^* = \{w_{pro^1}, w_{pro^2}, ..., w_{pro^P}\}$
Randomly initialize layers from 1 to L
Record parameter space dimensions: $\{\mathbb{R}_1, \mathbb{R}_2, ..., \mathbb{R}_L\}$
Initialize a prototype layers list: $List_{pro} = []$
for l in $1, 2, ..., L$ **do**
  if $\forall \mathbb{R}_{pro^p} \in List_{pro} \neq \mathbb{R}_l$ **then**
    Append w_l into $List_{pro}$
  else
    Find w_{pro^p}, where $\mathbb{R}_{pro^p} = \mathbb{R}_l$
    Replace w_l with w_{pro^p}
  end if
end for
Return updated w as w^*

$$w_{tar_t^p} \leftarrow w_{pro^p}; \mathbb{R}_{tar_t^p} = \mathbb{R}_{pro^p}, t \in \{1, 2, ..., T^p\}, \tag{5}$$

where T^p is the number of target layers of prototype p. The whole network filled by several layers with unique weight size and Eq. 4 can be rewrited as

$$\mathcal{N}_L : w^* = [\overbrace{w_{pro^1}, ..., w_{pro^2}, ..., w_{pro^p}, ..., w_{pro^P}}^{T^1+T^2+...+T^P}]; \sum T^p = L. \tag{6}$$

We take a simple 5-layer MLP to clarify this operation: its dimensions are $[512, 100, 100, 100, 10]$ with 4 weight matrices $w_1 \in \mathbb{R}^{512 \times 100}$, $w_2 \in \mathbb{R}^{100 \times 100}$, $w_3 \in \mathbb{R}^{100 \times 100}$, and $w_4 \in \mathbb{R}^{100 \times 10}$. In this case, w_1, w_2 and w_4 are three prototype layers. w_2 has two target layers w_3 and itself. w_1/w_4 only has itself as target layer. The general algorithm is summarized in Alg. 1.

In this way, any repetitive modules in a given network structure can be represented by one bunch of random weights. Using the *sparse selection* strategy, we iteratively pick subnetworks in the same set of random weights to obtain diverse feature mappings. Therefore, the cost to represent the network significantly reduces, especially for deep network with many repetitive blocks. In other words, one random layer with different masks can represent the majority of a complete network structure, which is named as *one-layer*.

3.4 Random Weights Padding

One-layer strategy efficiently handles networks with many repetitive modules. The majority of the whole network can be compressed into one random layer with a set of masks. However, in real-world applications, various network architectures may not follow a tidy pattern resulting in different shapes for different layers. Hence, the *one-layer* is not flexible enough to efficiently represent such networks. To handle it, we naturally propose a enhanced strategy, *Random Weights Padding*. It consists of two versions, *Max-layer padding* and *Random vector padding*. We first formally rewrite Eq. 4 as

$$\mathcal{N}_L : w = [w_1, w_2, ..., w_L]; w_l \in \mathbb{R}^{d_l}, l \in \{1, 2, ..., L\}, \tag{7}$$

where w_l is flatten into a vector with dimension d_l (e.g., $d_l = I \times O$ for linear layer and $d_l = N \times H \times W$ for CNN layer).

Max-Layer Padding (MP). It chooses the layer w_m as the prototype where $d_m = max([d_1, d_2, ..., d_L])$ is the highest dimension. All other layers in $\mathcal{N}_L$ have fewer parameters than w_m. We keep the prototype as it is and simply pick the first d_l parameters from w_m to replace the parameters in w_l, which is described by

$$w_l \leftarrow w_m[: d_l]; l \in \{1, 2, ..., L\}. \tag{8}$$

Random Vector Padding (RP). Instead of picking a complete layer as prototype, RP further reduces the granularity of random prototype from a layer to a random weights vector with a relatively short length. We let $v_{pro} \in \mathbb{R}^{d_v}$ as the random weights vector with length d_v. For each layer l, we repeat v_{pro} several times to reach the length of w_l. It can be formally described as

$$w_l \leftarrow [\overbrace{v_{pro}, ...}^{d_l}]; l \in \{1, 2, ..., L\}. \tag{9}$$

After the padding operation, weights in Eq. 7 are reshaped back into the format of Eq. 4 to perform as a network. These two padding strategies are summarized in Alg. 2 and Alg. 3.

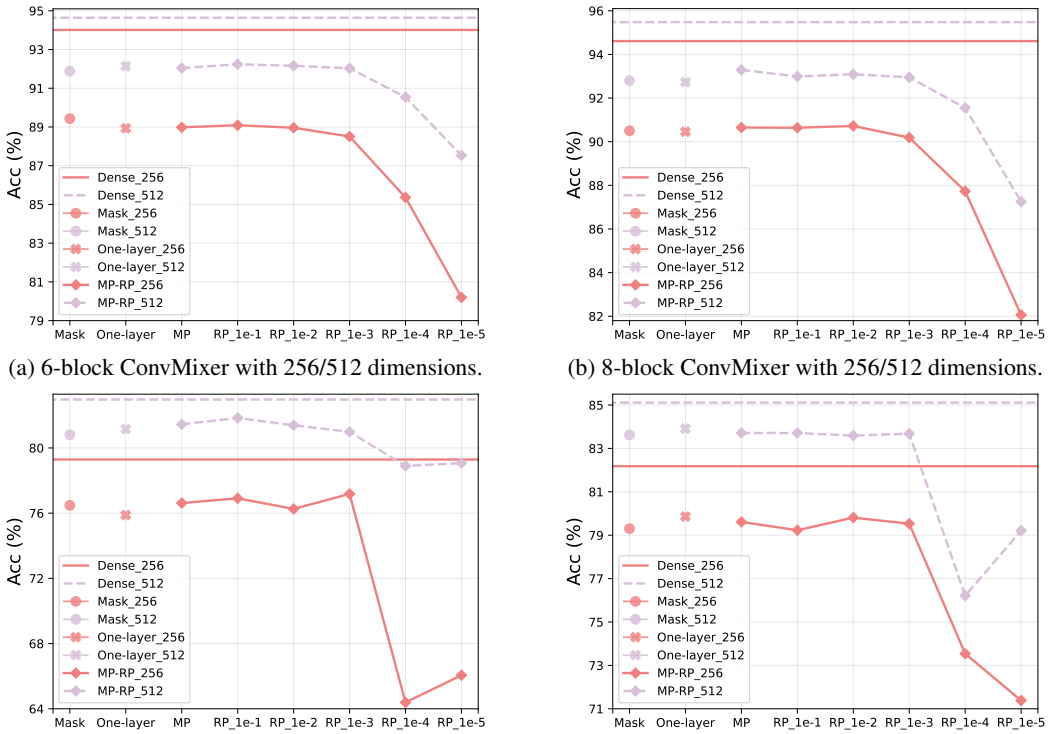
In the series of *one-layer*, *MP*, and *RP*, based on *sparse selection*, we explore using fewer unique weights to represent the whole network. Leveraging on the property of the fixed weight values, the cost of delegating a network keeps decreasing by using a random vector with a bunch of masks. By this mean, we fully explore the representative capacity of random weights with limited unique values. Furthermore, a novel model compression paradigm can be correspondingly established by restoring a set of random weights with different masks. Our three strategies compared to regular network setting are shown in Fig. 2.

Algorithm 2 Max-Layer Padding

Input: A L-layer random network with parameters: $w = \{w_1, w_2, ..., w_L\}$
Output: A L-layer network with MP: $w^* = \{w_m[: d_1], w_m[: d_2], ..., w_m[: d_L]\}$
Randomly initialize layers from 1 to L
Find the layer w_m with the maximum weight dimension d_m among all layers $w_l, l = \{1, 2, ..., L\}$
for l in $1, 2, ..., L$ **do**
    Replace w_l with the first d_l values in w_m given by $w_m[: d_l]$
end for
Return updated w as w^*

Algorithm 3 Random Vector Padding

Input: A L-layer random network with parameters: $w = \{w_1, w_2, ..., w_L\}$
Output: A L-layer network with RP: $w^* = \{[\overbrace{v_{pro}, ...}^{d_1}], [\overbrace{v_{pro}, ...}^{d_2}], ..., [\overbrace{v_{pro}, ...}^{d_L}]\}$
Randomly initialize layers from 1 to L
Randomly initialize a weights vector $v_{pro} \in \mathbb{R}^{d_v}$ with d_v dimension
for l in $1, 2, ..., L$ **do**
    Repeat v_{pro} until reaching the length d_l
    Replace w_l with the repeated vector v_{pro}
end for
Return updated w as w^*



(a) 6-block ConvMixer with 256/512 dimensions.

(b) 8-block ConvMixer with 256/512 dimensions.

(c) 6-block ViT with 256/512 dimensions.

(d) 8-block ViT with 256/512 dimensions.

Figure 3: Performances of ConvMixer and ViT backbones on CIFAR10 dataset with different model hyperparameters. Y-axis represent the test accuracy and X-axis denotes different network parameter settings. *Dense* means the model is trained in regular fashion. *Mask* is the sparse selection strategy. *One-layer*, MP, and *RP* are our strategies. The decimal after *RP* means the number of unique parameters compared with *MP*. From *Mask* to *RP 1e-5*, the unique values of network decrease. Different experimental settings illustrate the representative potential of random weights.

4 Experiments

Our experiments conduct empirically validations on two aspects of our interests. Firstly, we validate how large is the representative potential of random weights with limited unique values to test the effectiveness of our proposed Parameter-Efficient Masking Networks (PEMN). Secondly, leveraging on the pre-accessibility of random weights and lightweight storage cost of binary mask, it is promising to establish a new model compression paradigm.

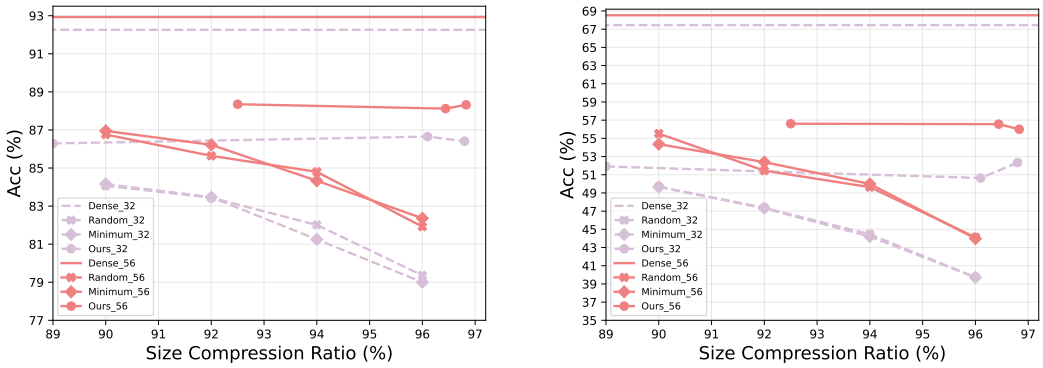
4.1 Preparation

We comprehensively use several classic or recently popular backbones for image classification task to conduct general validations. Backbones include ResNet32, ResNet56 [18], ConvMixer [33], and ViT [9]. We use CIFAR10 and CIFAR100 datasets [21] for our experiments.

4.2 Representative Random Weights in PEMN

We first explore the representative potential of random weights in PEMN based on our proposed strategies, *One-Layer*, *Max-Layer Padding (MP)*, and *Random Vector Padding (RP)*. We use a CNN based architecture Convmixer [33] and a MLP based model ViT [9] to conduct our experiments on CIFAR10 dataset [21].

In Fig. 3, we show 8 pairs of experiments based on 2 backbones (ConvMixer, ViT) using 2 depth numbers (6, 8) and 2 hidden dimensions (256, 512). Each pair includes a dense network performance and a series of results obtained by *sparse selection* with different random weighting strategies. Specifically, *Mask* learns the mask on the randomly initialized network. *One-layer*, *MP*, and *RP* represent our proposed strategies. To simplify the comparison, we use a rate number after *RP* to show how many unique parameters used in *RP* compared with *MP*. From the left *Mask* to right



(a) ResNet32/ResNet56 on CIFAR10 dataset. (b) Resnet32/ResNet56 on CIFAR100 dataset.

Figure 4: Compression performance validation on CIFAR10/CIFAR100 datasets on ResNet32/ResNet56 backbones. Y-axis denotes the test accuracy. X-axis means the network size compression ratio. Different colors represent different network architectures. The straight lines on the top are performance of dense model with regular training. Lines with different symbol shapes denote different settings. For ResNet, our three points are based on *MP*, *RP 1e-1*, and *RP 1e-2*, respectively. This pair of figures show that our proposed paradigm achieves admirable compression performance compared with baselines. In very high compression ratios, we can still maintain the test accuracy.

RP 1e-5, the number of unique parameters gradually decreases. Different settings show the similar patterns concluded as: 1) Compared with regular trained dense model, sparse selection approach generally obtains promising results, even if with a performance drop caused by the constraint of fixing all the parameters; 2) From left to right on X-axis, the performance gradually drops. This is caused by the decreasing number of unique values in network, which makes network has less representative capacity; 3) However, performance drop arises when the number of unique parameters is extremely low (e.g., *RP 1e-4*, *RP 1e-5*). The results remain stable for the most of random weights strategies; 4) Larger depth and hidden dimension boost the model capacity for different configurations. The performance drop of random weights strategies from their dense counterpart is also decreased. In addition, ViT shows some unstable fluctuation when fewer unique parameter, compared with ConvMixer with relatively stable patterns. This may caused by the difficulty of training MLP based network itself and will not affect our main conclusions.

The performance stability shown above illustrates the network representative capacity can be realized not only by overparameterizing the model, but also carefully picking different combinations of random parameters with limited unique values. In this way, the effectiveness of our PEMN is validated and we can represent a network using a random parameters prototype with different learned masks, instead of typically restoring all the different parameters. This property inspires us to introduce a new model compression paradigm proposed in following section.

4.3 A New Model Compression Paradigm

Practically, our work proposes a new network compression paradigm based on a group of random weights with different masks. We first elaborate the network compression and storage processes to clarify our advantages then report the empirical results.

4.3.1 Sparse Network Storage

Previous works aim to remove redundant weights (e.g., unstructured pruning) among different layers. The trivial weights are set to zero based on different criteria. The ratio of zero-weight in the whole network is regarded as sparsity ratio. Different approaches are compared based on their final test accuracy with a given sparsity ratio. Different from this conventional fashion which restoring sparse trained weights, we instead use fixed random weights with different masks to represent a network. To compare these two paradigm, we calculate the required storage size as an integrated measurement.

Assuming we have a trained network with p as parameter numbers and r as sparsity ratio. Due to its sparsity, we only need to restore the non-zero weight values accompanied with their position [13] denoted as a binary mask. The storage cost can be separated into two parts, C_w for weight values and

C_m for mask given by

$$C = C_w + C_m. \tag{10}$$

For conventional setting, C_w restores the values of kept sparse weights which is $p \cdot (1 - r)$ for the whole network. It needs to be restore in float format. C_m restores sparse positions of these weights, which can be restored into compressed sparse column (CSC) or compressed sparse row (CSR) formats with cost around $2p \cdot (1 - r)$ [13]. In our new paradigm, C_w records the values of the given random weights. For example, the *one-layer* requires to record all weights of non-repetitive layers and one prototype weights of all repetitive layers. *MP* requires to keep the values of layer with the largest number of parameters. *RP* only requires to restore the values of a random vector with given length. C_m is also for the sparse positions to record the selected subnetwork.

4.3.2 Compression Performance Validation

We test the compression performance on CIFAR10 and CIFAR100 datasets using ConvMixer with 6/8 depths, ResNet32, and ResNet56 backbones. The compression ratio is based on the storage size as we discussed instead of the conventional pruning ratio. Since we propose a new strategy to compress network, we involve two sparse network training baselines in our experiments. Specifically, we train a sparse network from scratch by removing random weight and minimum magnitude weights. For compression ratio, we set four settings for baselines: 90%, 92%, 94%, and 96%. We directly refer to settings, *MP*, *RP* from Sec. 4.2 for our paradigm. Their compression ratio is calculated using the same measurement as baselines.

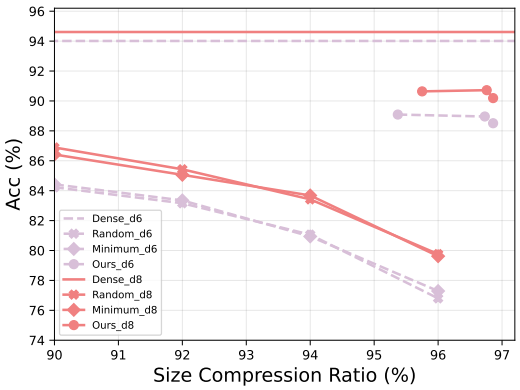


Figure 5: Compression performance validation on CIFAR10 dataset on ConvMixer backbone. Y-axis is the test accuracy. X-axis means compression ratio. Two pairs of comparisons are for different depths shown in different colors. Straight line on the top is the dense model performance. Curves in different symbols represent baselines and our method. For ConvMixer, our three points are based on *RP 1e-1*, *RP 1e-2*, and *RP 1e-3*, respectively.

In Fig. 4, we show 4 pairs of comparison based on 2 backbones (ResNet32/ResNet56) and 2 datasets (CIFAR10/CIFAR100). Each pair includes dense model, 2 baselines with 4 compression ratios, and our results in 3 ratios. Two baselines are sparse network training by pruning random weights and minimum magnitude weights, respectively. For convenience, we do not follow exactly the same compression ratios as baseline but directly use settings from Sec.4.2. For ResNet, we use *MP*, *RP 1e-1*, and *RP 1e-2*. Their corresponding compression ratios are computed as shown in figures. Experiments based on different networks and datasets show the similar conclusions summarized as: 1) our method outperforms the baselines by a significant margin, with even higher compression ratio; 2); Compared with conventional sparse network training where compressed model performance decreases obviously along with increasing compression ratio, our method is relatively robust to the compressed model size; 3) If we compare cross different models, we find compressed small model by our method even performs better than baselines using larger model; 4) Network scale affects the compression performance, compared with ResNet32, ResNet56 basically contains more parameters and performance drop between compressed network with its dense counterpart is relatively small. In Fig. 5, we show compression performance on CIFAR10 dataset using ConvMixer with depth 6 and 8. The settings are basically similar to Fig. 4. We can also draw the similar conclusions: 1) Our method outperforms the baselines on ConvMixer with different depths; 2) Our method compresses network into lower size but maintains higher performance.

As a summary, in Sec. 4, our experiments can be separated into two parts. Firstly, to validate our PEMN, we investigate the representative potential of random weights which are used to fill in the complete network structure using different proposed strategies. Secondly, the promising conclusion (Sec. 4.2) for this investigation naturally leads to the newly proposed network compression paradigm. Different from conventional fashion restoring sparse weights, we instead restore the fixed random weights and different masks. Empirically, we validate the effectiveness of our new paradigm for

network compression. Our experiments involve diverse network architectures to demonstrate the proposed paradigm can be generalized into different network designs.

5 Discussion and Conclusion

Discussion We summarize our intuitive logic and potential research direction in the future. Our fundamental insight is motivated by Supermasks [44] and Popup [31] showing random network encodes informative pattern by selecting subnetworks. They inspire us to understand neural network in a decoupled perspective: the informative output is delivered by certain weight-structure combination. Even if weights are fixed, the flexibility of learnable masks still provides promising capacity to represent diverse semantic information. We are the first to fully explore the representative potential of random weights, and practically, a new network compression paradigm is naturally established. We further discuss some research directions in the future following this study. Firstly, compared with conventional approaches need to record learned weights, our paradigm records random weights which can be pre-accessed, can this property be used for improve the model security? In addition, leveraging on the property that repetitive random weights existing in networks for our strategies, is it possible to specifically design hardware deployment configurations to achieve further compression or acceleration? Moreover, our PEMN is based on random initialized weights but cannot be directly deployed on pretrained models, which are more powerful these days. How to further improve our strategy on large-scale pretrained models is another interesting point to explore. We leave these topics in our future work.

Conclusion We first explore the maximum representative potential of a set of fixed random weights, which leverages different learned masks to obtain different feature mappings. Correspondingly, we introduce our proposed Parameter-Efficient Masking Networks (PEMN). Specifically, we naturally propose three strategies, *one-layer*, *max-layer padding (MP)*, and *random vector padding (RP)*, to fill in a complete network with given random weights. We find that a neural network with even limited unique parameters can achieve promising performance. It shows that parameters with fewer unique values have great representative potential achieved by learning different masks. Therefore, we can represent a complete network by combining a set of random weights with different masks. Inspired by this observation, we propose a novel network compression paradigm. Compared with traditional approaches, our paradigm can restore and transfer a network by only keeping a random vector with masks, instead of recording sparse weights for the whole network. Since the cost of restoring a mask is significantly lower than weight, we can achieve admirable compression performance. We conduct comprehensive experiments based on several popular network architectures to explore the random weights potential for PEMN and test the compression performance of our new paradigm. We expect our work can inspire further researches for both exploring network representative potential and network compression.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *NeurIPS*, 32, 2019.

[2] Y. Bai, H. Wang, Z. Tao, K. Li, and Y. Fu. Dual lottery ticket hypothesis. *arXiv preprint arXiv:2203.04248*, 2022.

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

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

[5] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[7] T. Dettmers and L. Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[10] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, pages 2943–2952. PMLR, 2020.

[11] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[12] A. Gaier and D. Ha. Weight agnostic neural networks. *NeurIPS*, 32, 2019.

[13] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *NeurIPS*, 28, 2015.

[15] B. Hassibi and D. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *NeurIPS*, 5, 1992.

[16] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299, 1993.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[19] S. Jastrzębski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio. Residual connections encourage iterative inference. *arXiv preprint arXiv:1710.04773*, 2017.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[22] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[23] Y. Le and X. Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[24] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *NeurIPS*, 2, 1989.

[25] N. Lee, T. Ajanthan, and P. H. Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[26] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[27] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. 2018.

[28] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *ICML*, pages 6682–6691. PMLR, 2020.

[29] H. Mostafa and X. Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML*, pages 4646–4655. PMLR, 2019.

[30] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763. PMLR, 2021.

[31] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? In *CVPR*, pages 11893–11902, 2020.

[32] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *NeurIPS*, 34, 2021.

[33] A. Trockman and J. Z. Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.

[35] C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[36] H. Wang and Y. Fu. Trainability preserving neural structured pruning. *arXiv preprint arXiv:2207.12534*, 2022.

[37] H. Wang, C. Qin, Y. Bai, Y. Zhang, and Y. Fu. Recent advances on neural network pruning at initialization. *arXiv e-prints*, pages arXiv–2103, 2021.

[38] H. Wang, C. Qin, Y. Zhang, and Y. Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020.

[39] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

[40] Y. Zhang, H. Wang, C. Qin, and Y. Fu. Aligned structured sparsity learning for efficient image super-resolution. *NeurIPS*, 34:2695–2706, 2021.

[41] Y. Zhang, H. Wang, C. Qin, and Y. Fu. Learning efficient image super-resolution networks via structure-regularized pruning. In *ICLR*, 2021.

[42] Y. Zhang and Q. Yang. A survey on multi-task learning. *TKDE*, 2021.

[43] Z. Zhang and C. Jung. Recurrent convolutions: A model compression point of view. 2018.

[44] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *NeurIPS*, 32, 2019.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes]
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes] See supplementary material.
(c) Did you discuss any potential negative societal impacts of your work? [Yes] See supplementary material.
(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See supplementary material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.
   (b) Did you mention the license of the assets? [Yes] See Section 4.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A. Implementation Details

For all the backbones used in our experiments, we follow their default training settings. For ConvMixer [33], we use AdamW [27] optimizer with triangular learning rate scheduler. We set the maximum learning rate as 0.05 with weight decay as 0.005. We set batch size as 512 and the number of total epochs is 100. We use different configurations for hidden dimension (256/512) and depth (6/8) in our experiments section. For ViT [9], we use Adam [20] optimizer with cosine learning rate scheduler. We set the maximum learning rate as 0.0001. We set batch size as 256 and the number of total epochs as 200. We use different configurations for hidden dimension (256/512) and depth (6/8) in our experiments section. For ResNet [18], we follow the implementation configurations in Popup [31]. Specifically, we use SGD optimizer with maximum learning rate 0.1 and cosine scheduler. The weight decay and momentum are set as 0.0005 and 0.9. We train 100 epochs with 256 batch size. For the sparse selection to pick the subnetworks, we follow the optimal choice in Popup and set it as 0.5 for our experiments. And we use kaiming uniform and kaiming normal [17] to initialize the scores and random weights, respectively.

B. Limitations and Potential Negative Societal Impacts

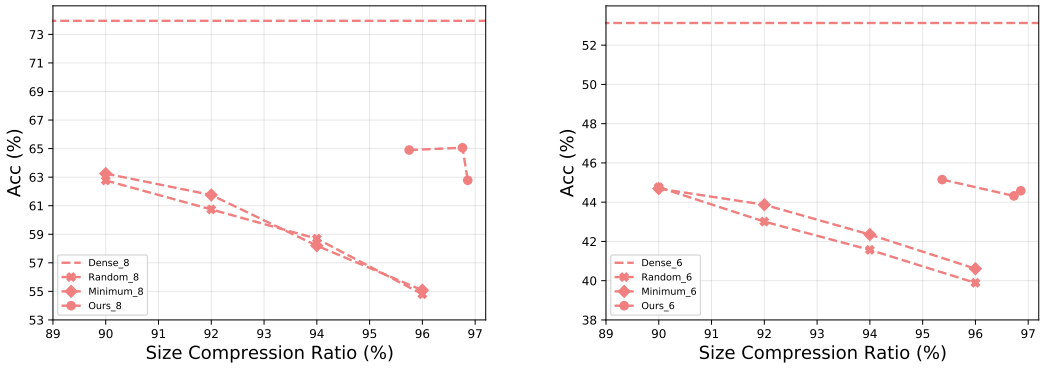
Firstly, our study focuses on exploring the random weight potential and representing a neural network with small storage cost. However, it cannot further help for accelerating the training and inference. This is a good point for us to further explore. Secondly, our proposed network compression strategy leverages on the representative potential of random weights with different masks, which benefits to reducing storage cost. However, following this strategy, it is hard to take advantages of powerful pretrained model these days. How to tailor our insight to large-scale pretrained model is another good point to explore. To the best of our knowledge, our study has no potential negative societal impacts.

C. More Experimental Evaluations

We further add experiments for CIFAR100 [21] and Tiny-ImageNet [23] datasets using ConvMixer as backbone to test our new network compression paradigm. The results are shown in Fig. 6. We use 8 and 6 as depth number for CIFAR100 and Tiny-ImageNet datasets, respectively and other settings follow the same mentioned above. The results in figure show that our compression strategy outperforms the baseline methods and validate the effectiveness of our propose network compression paradigm. We also conduct experiments on large-scale ImageNet [6] dataset using ResNet50 [18] as backbone. For simplicity, we customize two ImageNet subset for convenient evaluation. We sample 100/200 classes from original ImageNet to construct our subsets. We compare our compression strategy with magnitude pruning baseline and results are shown in Fig. 7. The results of our method are promising and demonstrate its effectiveness on challenging ImageNet dataset.

D. The Results of Repeated Experiments

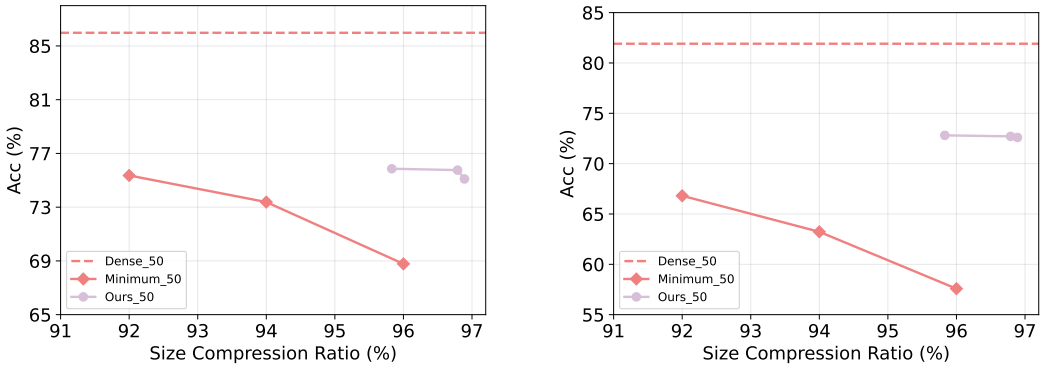
We further supplement the repeated experimental results. We supplement the main results from Fig. 3 and Fig. 4. We repeated each experiment three times and report mean and std. In Tab. 1, Tab. 2, Tab. 3, Tab. 4, we provide the repeated results for Fig. 3. According to these results, we make several conclusions: 1) Our experimental performance are generally consistent across different datasets and different settings. The supplemented results follow the accuracy patterns and support the conclusions provided in our draft; 2) ConvMixer is more stable than ViT backbone across different settings; 3) Overall, along with the decreasing number of unique values in the network (from the left column to the right column of tables), the performance variations increase correspondingly. The limited unique weight values decreases the stability of the network. In Tab. 5, Tab. 6, we provide the repeated results for Fig. 4. The first four columns show the compression baselines (the first / second items represent random and magnitude pruning). Based on the results shown above, we find our strategies generally outperform the model compression baselines and these results support our conclusion in the draft.



(a) ConvMixer on CIFAR100 dataset.

(b) ConvMixer on Tiny-ImageNet dataset.

Figure 6: Compression performance validation on CIFAR100/Tiny-ImageNet datasets on ConvMixer backbone. Y-axis denotes the test accuracy. X-axis means the network size compression ratio. The straight lines on the top are performance of dense model with regular training. Lines with different symbol shapes denote different settings. Our three points are based on *RP 1e-1*, *RP 1e-2*, and *RP 1e-3*, respectively. This figure shows that our proposed paradigm achieves admirable compression performance compared with baselines. We can still maintain the test accuracy in very high compression ratios.



(a) ResNet50 on ImageNet 100 subset.

(b) ResNet50 on ImageNet 200 subset.

Figure 7: Compression performance validation on ImageNet 100/200 subsets on ResNet50 backbone. Y-axis denotes the test accuracy. X-axis means the network size compression ratio. The straight lines on the top are performance of dense model with regular training. Lines with different symbol shapes denote different settings. Our three points are based on *RP 1e-1*, *RP 1e-2*, and *RP 1e-3*, respectively. This figure shows that our proposed compression strategy achieves promising performance on challenging ImageNet dataset compared with baselines.

Table 1: Repeated experimental results for ConvMixer 6-block in subfigure (a) of Fig. 3

Dim	Mask	One-layer	MP	RP_1e-1	RP_1e-2	RP_1e-3	RP_1e-4	RP_1e-5
256	89.31 (0.16)	88.80 (0.11)	88.97 (0.08)	88.70 (0.21)	88.89 (0.13)	88.52 (0.15)	85.76 (0.34)	81.01 (0.38)
512	91.90 (0.03)	91.87 (0.14)	92.02 (0.02)	92.07 (0.12)	92.13 (0.16)	92.05 (0.03)	90.55 (0.14)	87.40 (0.20)

Table 2: Repeated experimental results for ConvMixer 8-block in subfigure (b) of Fig. 3.

Dim	Mask	One-layer	MP	RP_1e-1	RP_1e-2	RP_1e-3	RP_1e-4	RP_1e-5
256	90.42 (0.09)	90.47 (0.04)	90.65 (0.11)	90.63 (0.14)	90.59 (0.06)	90.06 (0.22)	87.64 (0.19)	82.34 (0.26)
512	92.69 (0.11)	92.71 (0.06)	93.21 (0.05)	92.90 (0.07)	92.88 (0.15)	92.90 (0.07)	91.71 (0.20)	87.40 (0.22)

Table 3: Repeated experimental results for ViT 6-block in subfigure (c) of Fig. 3.

Dim	Mask	One-layer	MP	RP_1e-1	RP_1e-2	RP_1e-3	RP_1e-4	RP_1e-5
256	76.35 (0.15)	76.21 (0.14)	76.70 (0.10)	77.01 (0.17)	76.76 (0.21)	76.80 (0.20)	64.84 (0.21)	65.76 (0.23)
512	80.73 (0.21)	81.56 (0.25)	81.50 (0.11)	81.87 (0.06)	81.25 (0.13)	80.98 (0.16)	79.17 (0.25)	79.00 (0.14)

Table 4: Repeated experimental results for ViT 8-block in subfigure (d) of Fig. 3.

Dim	Mask	One-layer	MP	RP_1e-1	RP_1e-2	RP_1e-3	RP_1e-4	RP_1e-5
256	79.21 (0.09)	79.54 (0.16)	79.23 (0.22)	79.30 (0.08)	79.62 (0.15)	79.28 (0.14)	73.79 (0.26)	71.71 (0.28)
512	83.44 (0.17)	83.50 (0.26)	83.66 (0.12)	83.67 (0.13)	83.25 (0.20)	83.34 (0.19)	76.73 (0.31)	78.84 (0.29)

Table 5: Repeated experimental results for ResNet56/32 on CIFAR10 in subfigure (a) of Fig. 4

Network	pr0.9	pr0.92	pr0.94	pr0.96	Ours-MP	Ours-RP_1e-1	Ours-RP_1e-2
ResNet56	86.35 (0.22) / 86.74 (0.28)	85.55 (0.14) / 86.01 (0.17)	85.01 (0.19) / 84.62 (0.27)	81.93 (0.16) / 82.04 (0.18)	88.13 (0.19)	88.36 (0.24)	87.97 (0.21)
ResNet32	84.21 (0.13) / 84.22 (0.05)	83.29 (0.11) / 83.60 (0.20)	82.08 (0.16) / 81.56 (0.28)	79.36 (0.13) / 79.12 (0.15)	86.33 (0.14)	86.58 (0.09)	86.39 (0.18)

Table 6: Repeated experimental results for ResNet56/32 on CIFAR100 in subfigure (b) of Fig. 4.

Network	pr0.9	pr0.92	pr0.94	pr0.96	Ours-MP	Ours-RP_1e-1	Ours-RP_1e-2
ResNet56	55.21 (0.36) / 54.01 (0.38)	51.96 (0.33) / 52.54 (0.16)	49.72 (0.18) / 49.70 (0.25)	44.00 (0.13) / 44.18 (0.26)	56.39 (0.29)	56.58 (0.21)	55.84 (0.27)
ResNet32	49.21 (0.29) / 49.35 (0.14)	47.36 (0.11) / 47.38 (0.07)	43.70 (0.41) / 44.60 (0.29)	39.78 (0.23) / 39.54 (0.37)	51.76 (0.25)	50.78 (0.30)	51.94 (0.19)