IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 November 2022, accepted 7 February 2023, date of publication 15 February 2023, date of current version 23 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3245808

RESEARCH ARTICLE

Recurrent Residual Networks Contain Stronger Lottery Tickets

ÁNGEL LÓPEZ GARCÍA-ARIAS[1], (Graduate Student Member, IEEE), YASUYUKI OKOSHI[1], MASANORI HASHIMOTO[2], (Senior Member, IEEE), MASATO MOTOMURA[1], (Fellow, IEEE), AND JAEHOON YU[1], (Member, IEEE)

[1]Tokyo Institute of Technology, Yokohama, Kanagawa 226-8502, Japan
[2]Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

Corresponding author: Ángel López García-Arias (lopez@artic.iir.titech.ac.jp)

This work was supported by Japan Science and Technology Agency (JST) CREST Grant Number JPMJCR18K2, Japan.

ABSTRACT Accurate neural networks can be found just by pruning a randomly initialized overparameterized model, leaving out the need for any weight optimization. The resulting subnetworks are small, sparse, and ternary, making excellent candidates for efficient hardware implementation. However, finding optimal connectivity patterns is an open challenge. Based on the evidence that residual networks may be approximating unrolled shallow recurrent neural networks, we conjecture that they contain better candidate subnetworks at inference time when explicitly transformed into recurrent architectures. This hypothesis is put to the test on image classification tasks, where we find subnetworks within the recurrent models that are more accurate and parameter-efficient than both the ones found within feedforward models and than the full models with learned weights. Furthermore, random recurrent subnetworks are tiny: under a simple compression scheme, ResNet-50 is compressed without a drastic loss in performance to 48.55× less memory size, fitting in under 2 megabytes. Code available at: https://github.com/Lopez-Angel/hidden-fold-networks.

INDEX TERMS Deep neural networks, lottery ticket hypothesis, recurrent neural networks, residual networks, pruning, binary neural networks, ternary neural networks.

I. INTRODUCTION

The Strong Lottery Ticket Hypothesis [1], [2] states that high-performing neural networks can be obtained just by pruning an overparameterized dense model, as they are already available hidden inside of the randomly initialized parent network. This claim supersedes the Lottery Ticket Hypothesis [3], as weight training is entirely unnecessary. These sparse, random, and tiny subnetworks achieve competitive performance in vision tasks and can be exploited for very efficient inference hardware implementation [4].

However, current training methods have trouble finding the optimal connectivity patterns [5]. It has been found that better-performing random subnetworks can be discovered by constraining the weight precision to ternary via the combined effect of random binary weight initialization and a learned binary mask [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Ali.

This paper shows that even higher-performing subnetworks can be obtained by imposing two additional constraints: one in the hypothesis space and one in the search space. Both constraints are imposed simultaneously by transforming a residual network into a recurrent architecture, restraining the hypothesis to iterative functions to be searched within fewer random weights. We make the observation that residual networks are architecturally inclined to learn ensembles of all the possible unrollings of a shallow recurrent neural network, based on which this restriction on the hypothesis space actually increases the number of candidate subnetworks of interest available at initialization time.

The resulting subnetworks are parameter-efficient and have a competitive performance on image classification tasks, supporting the arguments for embracing feedback connections in computer vision. Furthermore, they can be compressed to tiny memory sizes and have a high degree of parameter reusability, making them even better candidates for inference acceleration.

The remainder of the paper is organized as follows. Section II reviews the related work on the Weak and Strong Lottery Ticket Hypotheses, residual networks, and recurrent convolutional neural networks for computer vision. After Section III presents a method for obtaining high performing, sparse, random, recurrent residual subnetworks, Section IV explores it in detail. Finally, after a discussion in Section V, Section VI concludes the paper.

II. BACKGROUND

Fueled by the increasingly more accessible and powerful computational power promised by Moore's Law, artificial neural networks experienced a growth in size that gave the field of deep learning its name [6]. This trend, initially led by image classification models [7], has continued as researchers keep enhancing neural networks at the cost of model growth. Recently natural language processing and generative models have also jumped on board, offering impressive capabilities at the price of an immense size [8], [9].

Although it is a convenient compromise for cutting-edge research, the vast computational cost of these models is impractical for real-world applications, motivating a parallel trend of research that aims for small, efficient models.

Strong lottery tickets are a family of efficient networks obtained by a training approach that merges learning, pruning, and weight quantization in a single process. Section II-A summarizes the evolution of training methods, illustrated in Fig. 1, that lead to their discovery. This paper explores a technique that transforms a residual network, an architecture overviewed in Section II-B, into a recurrent architecture to improve the strong lottery tickets contained within it. Finally, Section II-C reviews previous work on recurrent neural networks for computer vision.

A. THE LOTTERY TICKET HYPOTHESES

Pruning is a common technique for compressing trained networks into much smaller models by removing unnecessary weights [10], [11], [12], [13], [14]. Applied progressively by iterating training and pruning, large portions of trained models can be removed without affecting accuracy, making it evident that the original models are overparameterized. The sparsity of the resulting network can be exploited for additional compression with entropy coding and for arithmetic optimization. Combined with weight quantization, it has resulted in very efficient model compression schemes [15], [16] and specialized hardware neural accelerators [17].

1) THE WEAK LOTTERY TICKET HYPOTHESIS

It was generally found that the connectivity patterns found by pruning were not disentangled from the pre-trained weights, as they could not be reinitialized and trained from scratch. However, a recent breakthrough paper [3] showed that overparameterized neural networks contain a subnetwork that can be trained in isolation to match the original model—the *Lottery Ticket Hypothesis* (LTH). These subnetworks, nicknamed winning tickets, are found by iteratively training, pruning, and resetting the remaining weights to their original value.

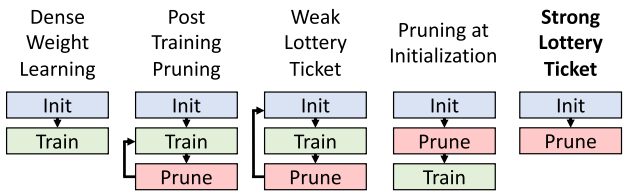


FIGURE 1. Evolution of the training methods leading to the Strong Lottery Ticket Hypothesis.

This paper has inspired a quickly growing body of work that has found that a network contains not one but several tickets [18], which may be connected [19]. Moreover, after it was shown that tickets can be identified early in the training process [20], some methods have succeeded in pruning before training [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], even before looking at the data, in order to reduce training cost.

2) THE STRONG LOTTERY TICKET HYPOTHESIS

In a surprising turn of events, while analyzing the LTH, [1] found that learning weights is unnecessary: an overparameterized neural network contains high-performing subnetworks at its randomly initialized state, which can be found just by pruning. Furthermore, they described an algorithm for finding these subnetworks by training a binary mask. This technique was taken a step further in [2] with a training algorithm and a weight initialization scheme that delivers sparse random subnetworks of competitive performance in image classification tasks.

After a series of papers analyzing the theoretical bounds of necessary overparameterization for the presence of this type of tickets [31], [32], [33], the existence of subnetworks that can be obtained just by pruning has come to be named the *Strong Lottery Ticket Hypothesis* (SLTH), albeit some works refer to it as the Multi-Prize Lottery Ticket Hypothesis [34], or simply as "hidden networks".

Despite being a surprising finding, strong tickets have some closely related precedents. Extreme learning machines [35] are feedforward networks that use fixed random weights in their hidden units, only learning the output layer. Reservoir computers [36] use random recurrent architectures in an analogous manner. Similarly, [37] proposed substituting convolutional layers with fixed additive random noise and a learned linear combination. It was demonstrated that non-trivial accuracy could be achieved just by training the batch normalization parameters of a fixed random network [38]. More directly related, [39] adapted a binary neural network training method to learn binary masks that, when applied to a trained model, extracted subnetworks that performed well on untrained tasks. When using a randomly weighted backbone model, they even found subnetworks with non-trivial accuracy, closely missing the discovery of strong

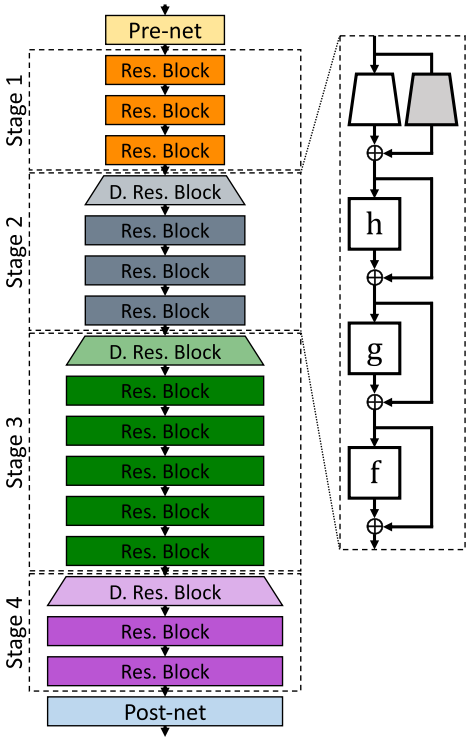


FIGURE 2. The residual network architecture. Specifically, ResNet-50, which has 3, 4, 6, and 3 blocks in each stage, from input to output.

tickets. Furthermore, the bio-plausibility of strong tickets has been considered by linking it to the emergence of innate face-selectivity in the visual cortex [40].

The SLTH does not only merge training and pruning, but also quantization: strong lottery tickets have been found to be robust to binarization [34], [41], sparser when using ternary masks [42], and more accurate with a scalar mask [43]. Furthermore, sparse random weights and binary masks can be exploited for designing energy-efficient inference hardware [4], which can even switch the binary mask for adjusting computational cost at the edge [4] or for reusing the same random weights for a different task [44].

However, current algorithms for finding strong tickets are unable to recover optimal strong tickets planted in the model artificially [5]. Training methods that use multiple random seeds, either concurrently [45] or iteratively [46], can improve performance, but at the cost of increasing model size. This paper analyzes a method, first proposed in [47], that delivers stronger and smaller tickets by constraining the optimization space.

B. RESIDUAL NEURAL NETWORKS

Among the ever-growing variety of neural network architectures, the residual neural network (ResNet) [48], depicted in Fig. 2, remains the backbone of many SOTA models.

ResNet is a pyramidal feedforward deep convolutional neural network formed by a convolutional pre-net followed by four stages of residual blocks and finished with a fully-connected post-net classifier. Each of the four stages uses a different representation space size, adjusted by the first residual block by downsampling the feature map and doubling the number of channels. The remaining blocks within a stage, of identical shape and size, keep the same representation space size.

Residual blocks are formed by a concatenation of batch-norm, ReLU, and convolutional layers. Specifically, this paper uses the bottleneck block [48], composed of three convolutional layers with kernel sizes 1×1, 3×3, and 1×1, in order. Each residual block has a skip-connection: an identity function in parallel to the block that adds its input to its output. In order to adjust the different representation space sizes, downsampling blocks have a learnable layer in the skip-connection—a projection shortcut.

The original intuition behind this architecture was that skip-connections provide a clear path for backpropagation for reaching a layer directly, thus solving the vanishing gradient problem. Based on the *representation view*, which directly associates layer depth with level of representation, a deeper network is able to form a deeper feature hierarchy and thus can solve more complex problems. Indeed, skip connections allowed to successfully train ResNets of thousands of layers. Furthermore, a continuation of the original work [49] updated the original architecture with a full pre-activation structure, relocating all activation layers into the residual blocks to leave a clear path of identity shortcuts. In the forward pass, this path could be seen as an implementation of all the feedforward lateral connections observed in the visual cortex [50].

However, this is not the only way to interpret ResNet. Lesion studies showed that removing or shuffling the order of residual blocks does not have an acute effect on its performance [51]. Instead, there is a proportional relationship between the introduced corruption and the performance loss. This phenomenon is not observed in other feedforward models, where similar lesions are critical. From these results stem two alternative views of ResNet: some authors have defended that ResNet is an ensemble of shallow networks, while others have argued that it may be an approximation of an unrolled shallow recurrent neural network. This paper tries to reconcile these two views into a single one.

1) THE ENSEMBLE VIEW

Each residual block can be viewed as a path bifurcation. Then, ResNet can be interpreted as an ensemble of all the possible paths within it [51], i.e., 2^n paths for a model with n residual blocks.

If considering a chain of three full pre-activation residual blocks approximating functions h, g, and f, their composition can be written as

$$\begin{aligned}(f+I)\circ(g+I)\circ(h+I)&\\ =(f+I)\circ(g\circ(h+I)+(h+I))&\\ =f\circ(g\circ(h+I)+(h+I))&\\ +(g\circ(h+I)+(h+I))&\\ +(h+I),&\end{aligned} \tag{1}$$

where $\circ$ denotes function composition, and I is the identity function with the same domain and codomain as h, g, and f. Residual blocks perform a transformation on the previous output, but this transformation is not compositional, as they contribute their output additively—as a new element of an ensemble. Indeed, if approximating h, g, and f as linear functions, this chain of residual blocks can be directly interpreted as the ensemble of all possible paths:

$$\begin{aligned}&(f+I)\circ(g+I)\circ(h+I)\\&\approx f\circ g\circ h+f\circ g+f\circ h+g\circ h+f+g+h+I.\end{aligned}\tag{2}$$

A downsampling block also contributes additively to the ensemble, but its projection shortcut transforms all the previously ensembled outputs:

$$\begin{aligned}&(f+s)\circ(g+I)\circ(h+I)\\&=(f+s)\circ(g\circ(h+I)+(h+I))\\&=f\circ(g\circ(h+I)+(h+I))\\&\quad+s\circ(g\circ(h+I)+(h+I)),\end{aligned}\tag{3}$$

where s is the shortcut function of downsampling block f.

A parallel analysis could be done for any other feedforward neural network, but in the case of ResNet, paths are not the same length and do not go through the same layers, i.e., ResNet is an ensemble of networks of various depths. Then, it makes sense that removing a block only has a mild effect on accuracy, as it only removes a subset of the ensemble. The effect of shuffling can also be understood through the linear approximation.

Of course, residual blocks are non-linear. However, residual blocks' nonlinearity can be considered weak, as their outputs have a small magnitude centered around zero that contributes little compared to the identity function [48], [52], [53].

Several ResNet improvements have been proposed based on this view. Removing random subsets of blocks during training makes the ensembled networks shallower and acts as regularization [54]. Using multiple skip-connections per block increases the amount of ensembled paths [55], boosting performance. Network depth can be reduced by increasing width, allowing to train bigger models to higher performance in lower time [56].

2) THE UNROLLED ITERATIVE ESTIMATION VIEW

Another possible explanation for the lesion and shuffling results is that all the blocks within a stage—which have the same shape and receive partially the same inputs and gradients through the identity shortcuts—are approximating the same function. That is, that ResNet naturally converges to the approximation of an unrolled shallow recurrent neural network.

Proponents of this view have argued that each of ResNet's stages corresponds to a different hierarchical level of representation, composed by the downsampling block, whereas the rest of the blocks perform iterative refinement of features [52].

ResNet's iterative behavior was observed by visualizing its activations [57], and it has also been analytically and empirically demonstrated that residual blocks perform iterative refinement of features by approximating a gradient descent step during inference [53]. Furthermore, ResNet can be transformed explicitly into a recurrent architecture without a critical loss in performance [58], a transformation exploited in this paper.

Although they have been presented as opposing views, we argue that they are compatible: ResNet can be interpreted as both an ensemble and an unrolled shallow recurrent neural network at the same time.

If $h\approx g\approx f$, (1) is rewritten as

$$\begin{aligned}&(f+I)\circ(g+I)\circ(h+I)\\&\approx(f+I)\circ(f+I)\circ(f+I)\\&=(f+I)^3\\&=f\circ(f\circ(f+I)+(f+I))\\&\quad+(f\circ(f+I)+(f+I))\\&\quad+(f+I),\end{aligned}\tag{4}$$

which can be seen as an ensemble of three iterations, where each iteration performs an additional recursion of f over the previous ensemble. Using the same linear approximation as above,

$$(f+I)^3\approx f^3+3f^2+3f+I.\tag{5}$$

More generally, a chain of n identical linear residual blocks can be seen as *a weighted ensemble of all the possible unrollings of a linear recurrent residual block* up to n iterations:

$$(f+I)^n\approx f^n+\sum_{i=1}^{n-1}nf^i+I.\tag{6}$$

Strictly, the repetitive application of an iterative function would compound to

$$f^n,\tag{7}$$

so the ensemble of unrollings view is a closer approximation than the strict unrolled iteration view.

This ensemble of unrollings provides a rich search space for finding strong tickets within a model with few parameters and suggests that stronger tickets will be present in a ResNet if it contains more recurrent approximations at its randomly initialized state.

C. RECURRENT NEURAL NETWORKS FOR VISION

Although deep neural networks for computer vision are predominantly feedforward, there have been some recurrent approaches. Recurrent convolutional neural networks (RCNN) have been used for scene parsing [59], object recognition [60], and image classification [61]. RCNN with long short-term memory (LSTM) [62] have also been used for

improving weather forecasting [63]. The authors of [64] exploited recurrence for early image prediction, and found that an RCNN naturally learns taxonomic representations that are iteratively refined to finer-grain classes, improving explainability. Similar to this paper, some works have explicitly transformed ResNet into RCNN for constructing efficient models [65], [66], which can be taken a step further by using recurrent depthwise separable convolutions [67], [68].

If feedforward deep neural networks have been successful in computer vision, why use recurrent architectures, usually associated with time-series data? Here we summarize some of the compelling arguments for this paradigm shift, which are thoroughly covered in [69].

The main arguments come from the field of neuroscience. While the primate visual cortex has only a few layers deep, with a wide variety of feedback and lateral connections [50], [70], computer vision models are hundreds of layers deep and strictly feedforward. Despite this, the visual cortex has high accuracy in multiple tasks, even though it has fewer units and much lower energy consumption. Iterative computation can be exploited to implement complex functions with limited hardware, and for adjusting computational time dynamically, a behavior observed in biological brains [50].

Vision neural network models that take direct inspiration from neuroscience are accurate, small, and better predictors of the brain [70], and can effectively adjust the number of iterations dynamically for improved efficiency [71].

Furthermore, some vision capabilities can only be implemented with recurrence. Vision is an active process, therefore requiring feedback connections to control the associated organs and to integrate priors and attention into the vision tasks. Although computer vision has given overwhelming attention to static images, vision processes time-series data—''video''. Feedback connectivity is necessary for processing temporal dependency, integrating stimulus history, and making predictions of the dynamic world.

Experimentally, neural network performance on vision tasks improves when models have recurrent and lateral connections [72], and they naturally learn a rich variety of feedback loops when given the freedom of choosing to share weights during training [73].

Intriguingly, ResNet, which is conjectured to approximate an RCNN, is both one of the best performing computer vision models and one of the most brain-like [74].

III. HIDDEN-FOLD NETWORKS

This section describes a method, first proposed in [47], that first folds a residual network to then find a strong lottery ticket within it. These folded subnetworks, which overperform the strong tickets hidden in feedforward ResNets, are referred to as *Hidden-Fold Networks (HFN)*.

Section III-A defines the recurrent ResNet architecture, and the training method employed to discover the strong ticket is described in Section III-B. The importance of the weight initialization choice for constraining the search space is detailed in Section III-C, after which Section III-D explains the normalization strategies considered. Finally, Section III-E clarifies the memory size compression scheme considered in this paper, which makes HFN some of the tiniest ResNets.

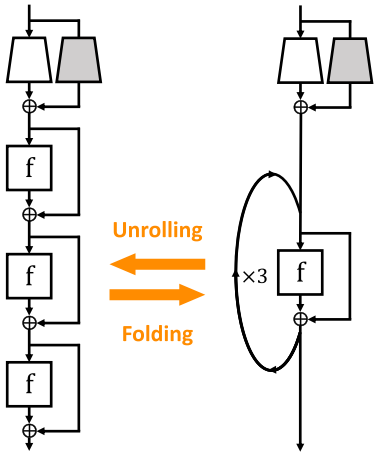


FIGURE 3. A ResNet stage folded into a recurrent residual stage through the opposite of time unrolling. The downsampling block is left untouched, whereas the rest of the stage is folded into a single recurrent block. That is, a stage of n blocks is folded into 2 blocks, the second of which is applied $n-1$ iterations.

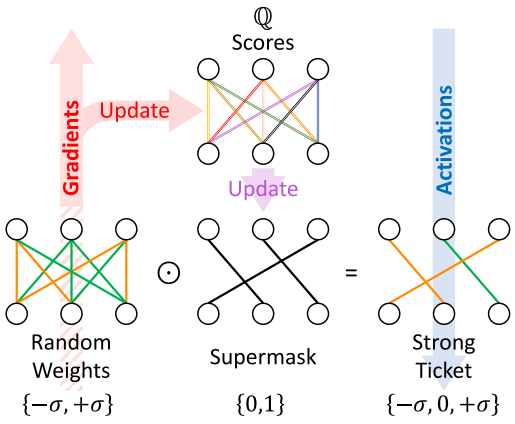
A. FOLDED RESNET ARCHITECTURE

According to the unrolled iteration view (see Section II-B2), the chains of residual blocks of identical shape within each stage are approximating an iterative function. Folding [58][1] is a method that transforms these chains into recurrent blocks, via weight sharing. That is, $h \approx g \approx f$ is transformed explicitly to $h = g = f$, and since applying identical functions in succession is equivalent to applying one of them repeatedly, the feedforward chain can be transformed into a single recurrent block in a process opposite to time unrolling.

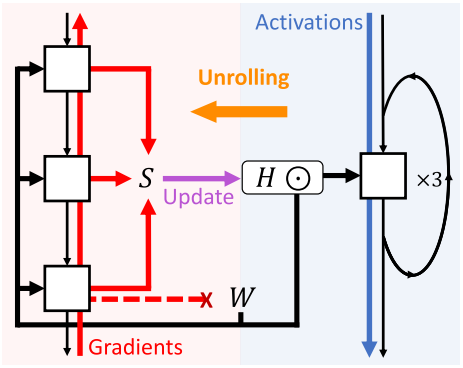
Since the downsampling block has a different shape than the rest of the blocks, it cannot be folded with the rest. This block could be removed by transforming ResNet into an isotropic architecture or by substituting it with a simpler block, strategies that were explored in [53], [58]. Nevertheless, based on the view that different stages correspond to different hierarchical levels of features composed by downsampling blocks [52], this paper leaves them untouched. The rest of the stage is folded into a single recurrent residual block that is iterated the same number of times as the number of original blocks, as illustrated in Fig. 3. Section IV-C2 experiments with the number of folded stages.

Folding has a double effect on the search space: it limits the set of functions that the model can learn—the hypothesis space—to iterative functions and reduces the number of parameters. We argue that, despite this trim in parameters means an exponential reduction of the available sub-

[1] Not to be confused with the method that compresses trained networks by removing unnecessary activation layers [75], or with the method that adds longer skip-connections [76], both also named folding.



(a) Supermask training with `edge-popup` [2]



(b) Folded models are unrolled, so scores receive a gradient from each iteration.

FIGURE 4. Training an HFN with a supermask. The supermask (*H*) includes the random weights (*W*) with the top-*k*% scores (*S*), updated via backpropagation. ⊙ is the Hadamard product.

network candidates, folded residual networks contain better performing strong tickets than their feedforward counterparts. If training ResNet's weights naturally converges to approximations of unrolled iterative functions, likely, the strong tickets contained within it are also approximations of recurrent networks. This limits the number of subnetworks of interest to only those that include similar random weights in consecutive blocks, a very small subset. Folding ResNet forces *all* candidate subnetworks to have a recurrent form. Therefore, the number of candidate subnetworks of interest is larger and likely to contain stronger tickets.

Furthermore, the parameter reduction happens not only at inference time but also at train time. At train time, the search space for strong tickets is largely reduced, making these folded tickets also easier to find. At inference time, the found subnetworks are even smaller and exploit parameter reusability, making them excellent candidates for efficient hardware implementation.

B. SUPERMASK TRAINING

Instead of optimizing the model's weights, the model is pruned to find a high-performing subnetwork hidden within the randomly initialized folded model—an HFN.

This connectivity pattern is learned by training a *supermask* [1], a pruning mask that contains a binary element for each weight. The ticket is unearthed at inference by applying an element-wise product of the random weight tensor and the trained supermask.

This paper uses the `edge-popup` [2] algorithm for training the supermasks, illustrated in Fig. 4a. `Edge-popup` defines a score for each weight, which is updated in the backward pass via backpropagation using straight-through estimation for the supermask (i.e., the supermask is not applied in the backward pass). The scores are then sorted by absolute value, and the supermask is updated to include the weights with the top-$k\%$ highest ranking scores and prune the rest. Top-$k\%$, thus, is a hyperparameter that sets the supermask's density a priori. Although top-$k\%$ is set globally, the sparsity is enforced at the layer level.

Folding does not affect this process: supermasks and scores are shared in the same way as their corresponding weights, and backpropagation flows through the unrolled model in the same way it would travel a feedforward model. Therefore, folded parameters receive a separate gradient from each iteration, as depicted in Fig. 4b.

This training method takes more computation time than standard dense weight learning (about 1.75× in our experiments), as it uses three parameters for each node, and scores must be sorted every training step. However, the resulting models are smaller, sparse, and formed by random weights, allowing for efficient inference implementations.

C. RANDOM SIGN CONSTANT WEIGHT INITIALIZATION

Following [1], [2], this paper considers two weight initialization strategies: Kaiming normal initialization (KN) [77], and signed Kaiming constant initialization (SC).

KN initializes weights of later l by sampling from

$$\mathcal{N}_l(0, \sigma_l^2), \tag{8}$$

where $\mathcal{N}$ denotes the normal distribution, and the standard deviation is scaled by the density k [1], [2]:

$$\sigma_l = \sqrt{\frac{2}{n_l k}}, \tag{9}$$

where n_l is the number of input dimensions of layer l.

On the other hand, SC initializes weights to the same modulus and a random sign by sampling from

$$\mathcal{D}_l\{-\sigma_l, +\sigma_l\}, \tag{10}$$

where $\mathcal{D}$ denotes the discrete uniform distribution.

Since the same modulus σ_l is shared by all the random weights in layer l, it can be absorbed by the normalization layer's scaling factor γ. Then, all weights in the model belong to $\{-1, +1\}$, and after applying the binary supermask in inference, the effective weights belong to $\{-1, 0, +1\}$. In other words, in combination with the binary supermask, SC initialization constrains the search space to *balanced ternary neural networks*. We argue that this strict constraint is the reason for its positive effect on accuracy.

Scores are always initialized with Kaiming uniform initialization [77] by sampling from

$$\mathcal{U}_l(-\sqrt{\frac{6}{n_l}}, \sqrt{\frac{6}{n_l}}), \tag{11}$$

where $\mathcal{U}$ denotes the uniform distribution.

D. UNSHARED BATCH NORMALIZATION

It is well known that the choice of normalization strategy is crucial for folded architectures. Specifically, batch normalization (batchnorm) [78] with independent affine parameters for each iteration—which has been rediscovered numerous times under different names [53], [58], [65], [66], [67]—has been reported to be especially effective. This paper also employs this strategy and refers to it as unshared batch normalization (UBN), following [53].

In addition to UBN, this work experiments with folding using shared affine batchnorm parameters (SBN), and folding using non-affine batchnorm (NA-BN). If the affine parameters are unshared, technically each iteration of a folded block applies a different function. Although this may have a significant impact on a recursive block's expressive power, in practice, it requires a very small number of additional parameters, and may even be bio-plausible [58].

Other normalization layers that do not normalize along the mini-batch dimension, which have been reported to be well suited for recurrent architectures, are also considered for folded blocks. Namely: layer normalization [79], which computes statistics along the channel dimension instead of the batch dimension; instance normalization [80], which uses a single channel; and group normalization [81], which generalizes the previous two by considering an arbitrary number of channels. Furthermore, we also consider local response normalization [7], a non-affine normalization scheme inspired by the lateral inhibition of biological neurons that has been successful on recurrent convolutional neural networks [60].

Previous work on supermask training of feedforward networks only considered either non-affine batchnorm [2] or affine batchnorm [43]. This paper explores both options also for the feedforward sections of the HFN model, as well as for the baselines.

E. MODEL COMPRESSION SCHEME

This paper reports model memory sizes considering a specific compression scheme oriented to specialized hardware.

All weights and biases are counted as occupying 32 bit. However, it is not necessary to store weights in the case of supermask training, since they can be generated on the fly from the original seed with a random number generator [1], [4]. Furthermore, this seed can be substituted with a hash of other model parameters [4], so it is not necessary to store it either. Therefore, the model size of models trained with supermask training is only the size of the supermask—one bit per weight—in addition to the memory size of the affine normalization parameters, if any. When used, affine parameters constitute a very small part of the total size: UBN models only occupy 5% to 15% more memory than their SBN or BN counterparts.

Although supermasks can be compressed further by exploiting their sparsity with entropy coding [4], [43], these techniques are not considered in this paper for simplicity. Counterintuitively, this means that sparsity has no impact on the reported model sizes.

HFNs can fit in under 2 MB under this scheme, as will be demonstrated later in Section IV-D, small enough for specialized hardware's on-chip SRAM memory resources. This provides a significant advantage, as off-chip DRAM memory access consumes orders of magnitude more energy and time than on-chip DRAM access or arithmetic operations [82]. Moreover, the weight's balanced ternary precision and sparsity can be leveraged to reduce the arithmetic cost.

It shall be noted that in conventional hardware, weights must be uncompressed to full precision and processed densely. In that case, the proposed models provide no computational cost advantage compared to their feedforward dense learned weight counterparts. For this reason, and because the arithmetic cost on specialized hardware is heavily implementation-dependent, this paper does not report FLOP counts and instead focuses on model memory size gains under the described compression scheme.

Table 1. Summary of the four methods compared on ResNet in this paper.

Method	Architecture	Training
Standard ("Vanilla") [48]	Feedforward	Dense weight learning
Folding [58]	**Recurrent**	Dense weight learning
Hidden-Networks (HNN) [2]	Feedforward	**Supermasks**
Hidden-Fold Networks (HFN)	**Recurrent**	**Supermasks**

IV. EXPERIMENTS AND RESULTS

This section analyzes how to better combine supermask training with a recurrent residual network and compares the results with the baseline methods, summarized in Table 1.

After describing the methodology in Section IV-A, Section IV-B examines the compatibility of both methods under diverse strategies of weight initialization, normalization, and bias learning. Section IV-C explores different model size options, including supermask sparsity, the number of stages folded, and model depth and width. Now with the optimal settings, the four methods are compared on CIFAR100 and ImageNet on Section IV-D, revealing HFN's strengths and weaknesses. Section IV-E investigates the iterative generality of the learned recurrent blocks by modifying the number of iterations after training, and Section IV-F concludes this section.

A. EXPERIMENTAL SETTINGS

All experiments described were implemented on PyTorch [83] based on the original code of [2], which is publicly available [84]. The baseline model for all experiments is ResNet-50, as described in [48], and variations of it. Since folding only applies to deep networks, basic experiments

were performed on the rather complex CIFAR100 [85] dataset, whereas ImageNet [86] was used for large-scale dataset evaluation.

Except when explicitly stated otherwise, experiments were carried out using the following methodology. The 60 000 images of CIFAR100 were split into 45 000 for training, 5000 for validation, and 10 000 for the test set. Image pre-processing and augmentation were done in the same way as [2]. Training on CIFAR100 is performed using stochastic gradient descent (SGD) with weight decay 0.0005, momentum 0.9, and batch size 128 during 200 epochs, with an additional 100 epochs for models deeper than 100 layers or double width. The learning rate is reduced using cosine annealing starting from 0.1, with no warmup. Experiments using ImageNet were performed for 100 epochs, using the hyperparameters recommended in [87]. Reported accuracy on CIFAR100 is the average of three runs of top-1 test accuracy scores, measured at the highest scoring validation epoch, whereas on ImageNet they correspond to single-run top-1 validation accuracy. Standard deviation is indicated with lines on bar graphs and with shaded areas on plots.

Training a vanilla ResNet-50 on CIFAR100 on an NVIDIA GeForce RTX 3090 takes 2.4 h, whereas its folded, HNN, and HFN versions take 2.2 h, 4.4 h, and 4.2 h, respectively.

B. COMPATIBILITY OF SUPERMASKS AND FOLDING

This section takes a preliminary look at how well supermask training and folding mix under different weight initialization and normalization strategies. Fig. 5 compares feedforward ResNet-50 to models with increasingly more stages folded, trained with dense weight learning in Fig. 5a and top-50% supermask training in Fig. 5b.

The impact on accuracy of transforming ResNet into a recurrent model is slightly more negative the more stages are folded, but very small in any case. In the case of supermask training, folding even results in a slight improvement in accuracy, as predicted. Both weight initialization choices produce similar results for weight learning, but for supermask training, signed Kaiming constant (SC) initialized models reach notably higher accuracy, as reported in [2]. SC is used for all methods hereafter, except for dense feedforward models, for which Kaiming normal (KN) is used.

1) FOLDING THE NORMALIZATION LAYER

Fig. 5 compares three batchnorm strategies for folded blocks: non-affine (NA-BN), affine with shared learned parameters (SBN), and affine with independent learned parameters for each iteration (UBN). Feedforward blocks use non-affine batchnorm, except when learning dense weights, when affine batchnorm is used. Accuracy tends to be higher the more affine parameters are learned, although the difference between SBN and UBN is smaller than previously reported.

This analysis is expanded in Fig. 6 to also compare layer norm (LN) [79], instance norm (IN) [80], group norm (GN) [81] (32 groups), and local response normalization

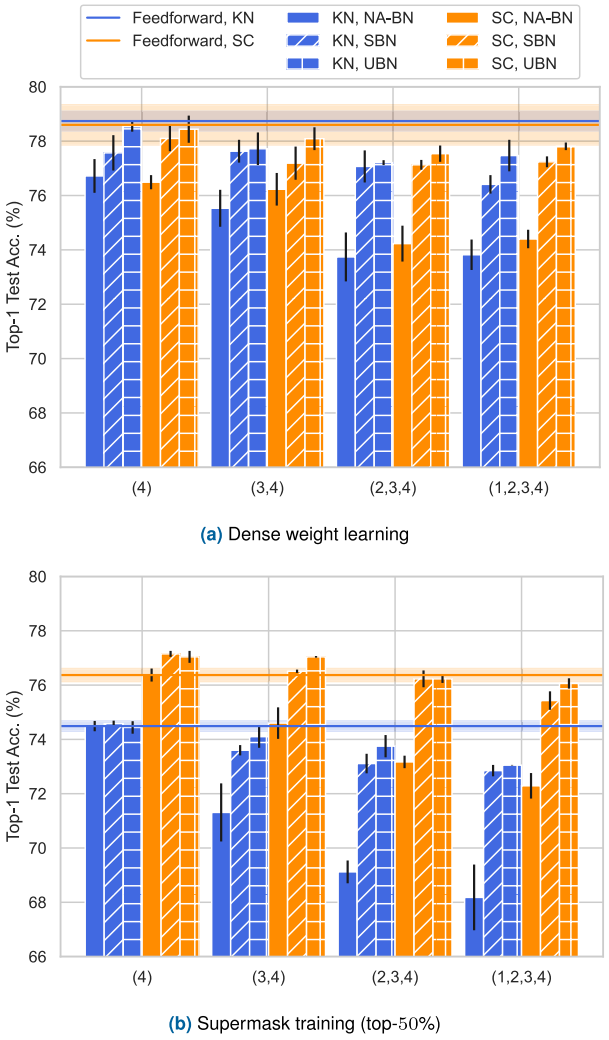


FIGURE 5. Impact of folding ResNet-50 when training (a) dense weights (b) a supermask for different combinations of weight initialization and batchnorm strategies. Numbers in parentheses indicate folded stages.

(LRN) [7]. Their affine parameters are shared in all cases except for LRN, which is intrinsically non-affine. GN uses 32 groups, and LRN uses size 5, additive factor 2, multiplicative factor 0.0001, and exponent 0.75.

As reported in previous work, additional affine parameters have a significant impact when folding with learned weights. However, normalizing along the channel dimension provides an even more significant advantage—bigger the more channels are considered. Unlike weight learning, HFN just works better when more affine parameters are learned. LRN overperforms NA-BN, but is surpassed by all the affine methods. Although closely followed by GN, UBN provides the highest accuracy for HFN. All experiments hereafter use UBN for recurrent blocks.

2) LEARNING ADDITIONAL PARAMETERS

Since learning more affine parameters results in better performance, it could seem obvious that they should be learned

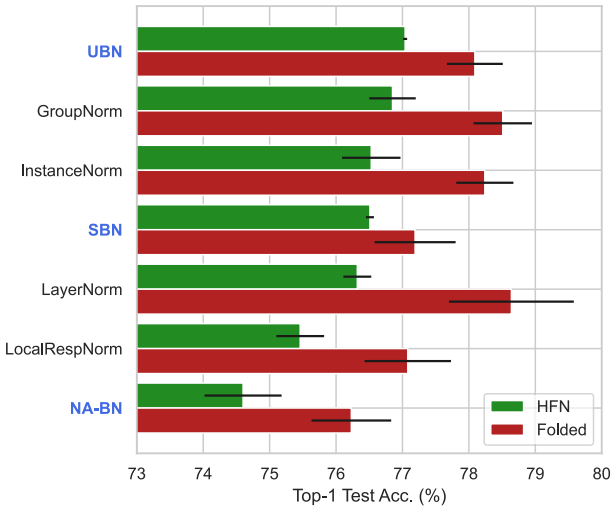


FIGURE 6. Impact of normalization layer choice for folded blocks. Only folding stages 3 and 4.

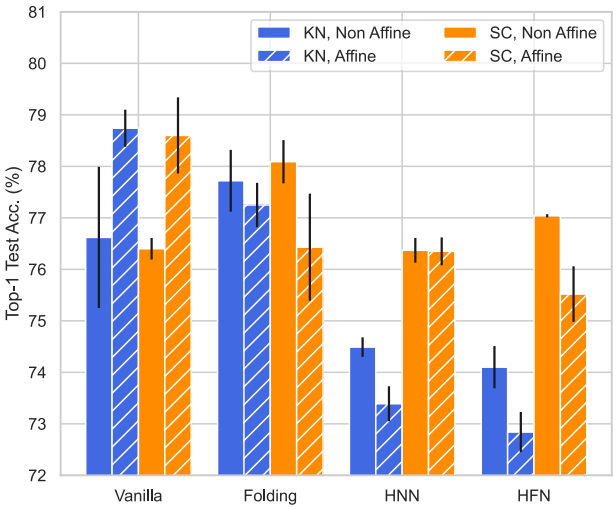


FIGURE 7. Affine batchnorm on unfolded blocks is only beneficial for Vanilla. Folded stages use UBN, and supermasks use top-50%.

in all layers. However, Fig. 7 shows that independently of the weight initialization used, using affine batchnorm on non-folded blocks is harmful if some part of the architecture is folded or using supermask training. Therefore, affine batchnorm is only used for dense feedforward models and folded blocks (as UBN).

Previous work [2], [47] reported results without learning biases of convolutional and fully connected layers. This is standard practice when using batchnorm, as the role of biases is absorbed by the shift parameter β in batchnorm's affine transformation, but these works used non-affine batchnorm. The impact of learning these additional parameters is explored in Fig. 8, which shows that it has an insignificant influence. Therefore, biases are not learned hereafter.

C. TUNING HFN's SIZE

This subsection analyzes HFN's tradeoff between size and accuracy by exploring its different size hyperparameters: the

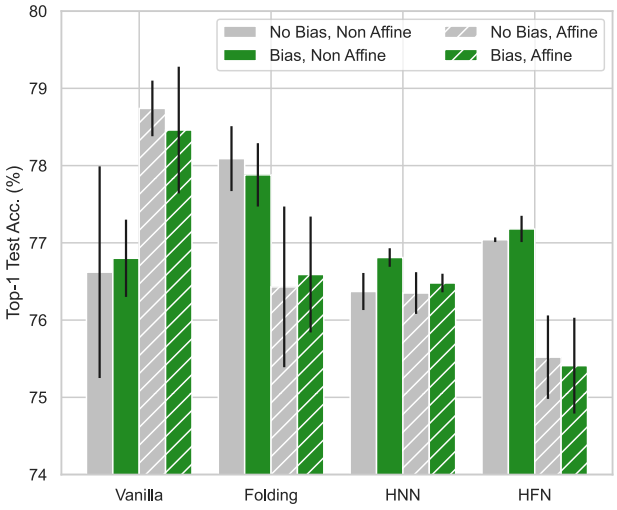


FIGURE 8. Learned biases have a negligible effect.

sparsity of the supermask, the number of stages folded into recurrent blocks, and the architectures' depth and width.

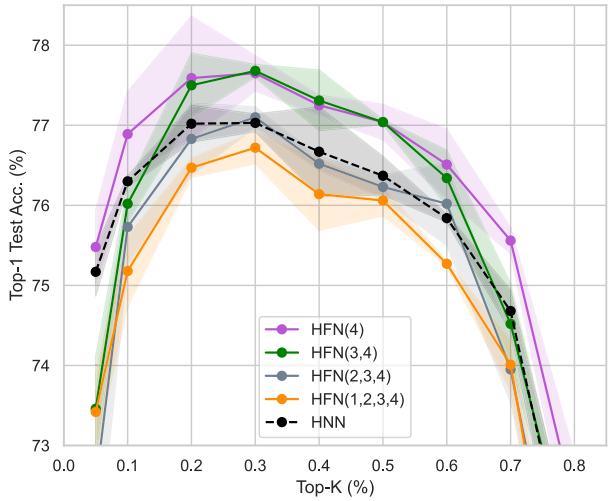
1) SUPERMASK DENSITY

Fig. 9a explores a range of top-k% values for different numbers of folded stages. The number of folded stages is increased from the deeper to the shallower stages, based on the evidence that the later stages of the visual ventral stream are more likely to exploit iterative computation [50].

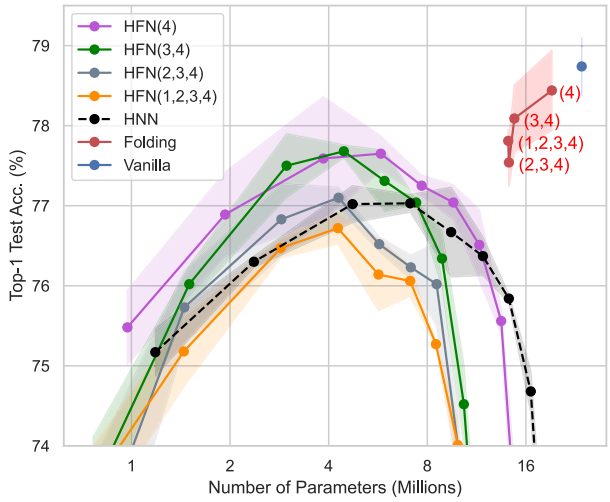
The optimal density range is between 20% and 40% in all cases, with a peak around top-30%. Interestingly, it is the same for the feedforward and folded models, suggesting that it may be optimal for the learning algorithm used (`edge-popup`) but not necessarily the density of the optimal strong ticket. Fig. 9b shows that HFNs are more parameter-efficient than feedforward strong tickets for the whole optimal range. Since density has no impact on model size in the proposed compression scheme, all experiments hereafter adopt the optimal density value of top-30%. This may not be the optimal density for the deeper and wider models introduced later. However, due to `edge-popup`'s drawback of sparsity having to be set a priori, tuning it for each configuration is costly.

2) NUMBER OF FOLDED STAGES

Fig. 9 shows that, despite the smaller size, folding even a single stage of ResNet results in higher accuracy and only suffers a drop when folding all stages. That means that the recurrent version of ResNet either contains strong tickets that are better performing or at least are easier to find for `edge-popup`. The best results are achieved when only folding stages 3 and 4, which contain most of the parameters of ResNet. Folding additional stages has a very small effect on model size and negatively impacts accuracy. For weight learning, accuracy drops when more stages are folded, but folding only stages 3 and 4 provides the best size-accuracy



(a) Impact of supermask density (top-k%) on feedforward and folded ResNet-50.



(b) Parameter count of the compared methods.

FIGURE 9. Accuracy to parameter count tradeoff for different supermask densities and number of folded stages. Points in (b) correspond to the densities in (a). Numbers in parentheses indicate folded stages.

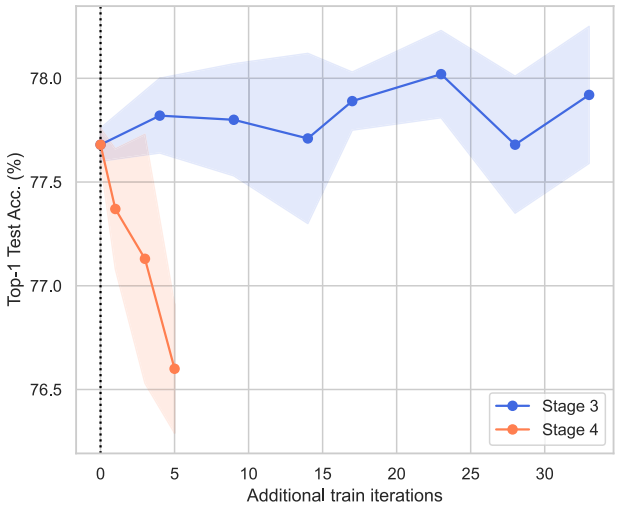


FIGURE 10. Adding extra iterations to individual folded stages of ResNet-50. ResNet-101 is equivalent to ResNet-50 with 17 additional blocks in stage 3.

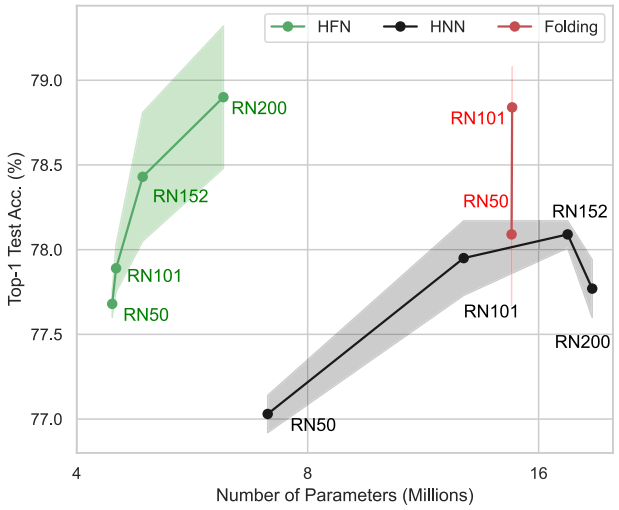


FIGURE 11. Depth scalability of folding and supermask training. ResNet-101 is only deeper than ResNet-50 in stage 3, but ResNet-152 and ResNet-200 are also deeper in stage 2.

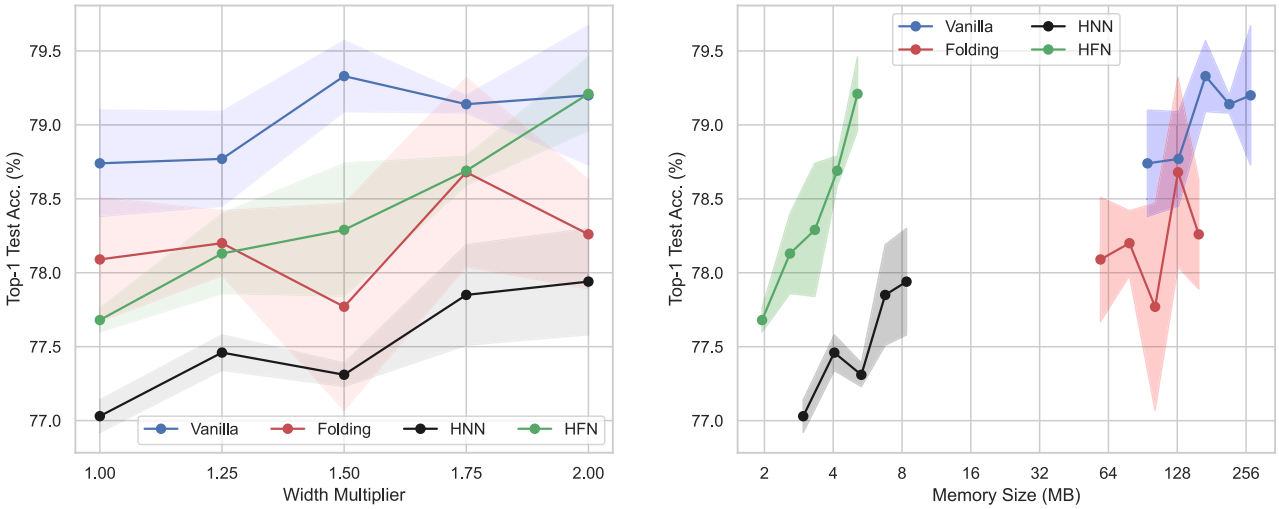
tradeoff. Therefore, this is the folding strategy used for both methods in all experiments hereafter.

3) ARCHITECTURE DEPTH

Deep learning's success in general, and ResNet's in specific, has been commonly attributed to network depth. However, folding collapses a stage's depth into a single block, transforming it into number of iterations. This subsection looks at the effect of this transformation and examines whether there is a benefit to extra depth when it does not imply additional parameters.

Fig. 10 depicts the effect of increasing the depth of a single stage of an HFN version of ResNet-50. Although increasing depth in stage 3 has a monotonic positive impact, it is milder than what would be expected for deep neural networks. Additional iterations in stage 4 have a destructive effect, presumably because of its much larger size (ResNets rarely have more than 3 blocks in this stage).

Additional iterations should not have a different impact on gradients than additional depth: folding preserves the identity mappings of the skip-connections, and after unrolling, gradients flow through the same number of layers as in the corresponding feedforward model. The reduced amount of parameters could explain the impaired effect of extra depth, but Fig. 11 shows that increasing iterations in folded models with learned weights has a much stronger effect. Furthermore, extra depth also has a more substantial effect on supermask training on feedforward models. However, when also increasing depth in stages that are not folded, HFN scales better than when not folded at all, although with increasingly bigger



(a) Testing a variety of widths on ResNet-50.

(b) Impact of width on model size.

FIGURE 12. Width scalability of the different methods on ResNet-50. Points in (b) correspond to the width multipliers in (a). All models are trained for 200 epochs.

variance. Deeper recurrent ResNets also contain stronger tickets, supporting our conjecture on hypothesis space: extra depth makes subnetworks randomly initialized close to a recurrent approximation even scarcer in feedforward ResNets of finite width.

4) ARCHITECTURE WIDTH

Although incrementing depth in a folded block only increases the number of iterations, deeper unfolded stages are not the only way of upscaling HFN. The search space for HFNs can be broadened by expanding ResNet's width [56].

Fig. 12 compares HFN's width scalability to that of the other methods by multiplying the number of channels of all layers by a width multiplier. Although all methods benefit from extra channels, HFN profits the most, overperforming both feedforward supermasks and dense folded models, with smaller variance. More remarkably, HFN even matches the wider dense feedforward architecture, despite being 52.47x smaller.

D. METHOD COMPARISON ON IMAGE CLASSIFICATION

1) CIFAR100

Fig. 13a (left) compares the accuracy and parameter count of different ResNet sizes trained with the compared methods on CIFAR100. HFN models are the most parameter-efficient: fewer parameters than equally performing models and more accurate than models with a similar parameter count. Furthermore, deeper HFNs and wider HFNs reach the highest accuracies overall.

Moreover, HFN's advantage is more evident when comparing in Fig. 13a (right) model memory sizes under the compression scheme described in Section III-E. HFNs are about half the size of their feedforward counterparts, with ResNet-50 fitting in under 2 MB. Wide-ResNet-50 overperforms dense models more than 30x larger, only matched by the wider dense (Fig. 12b).

2) IMAGENET

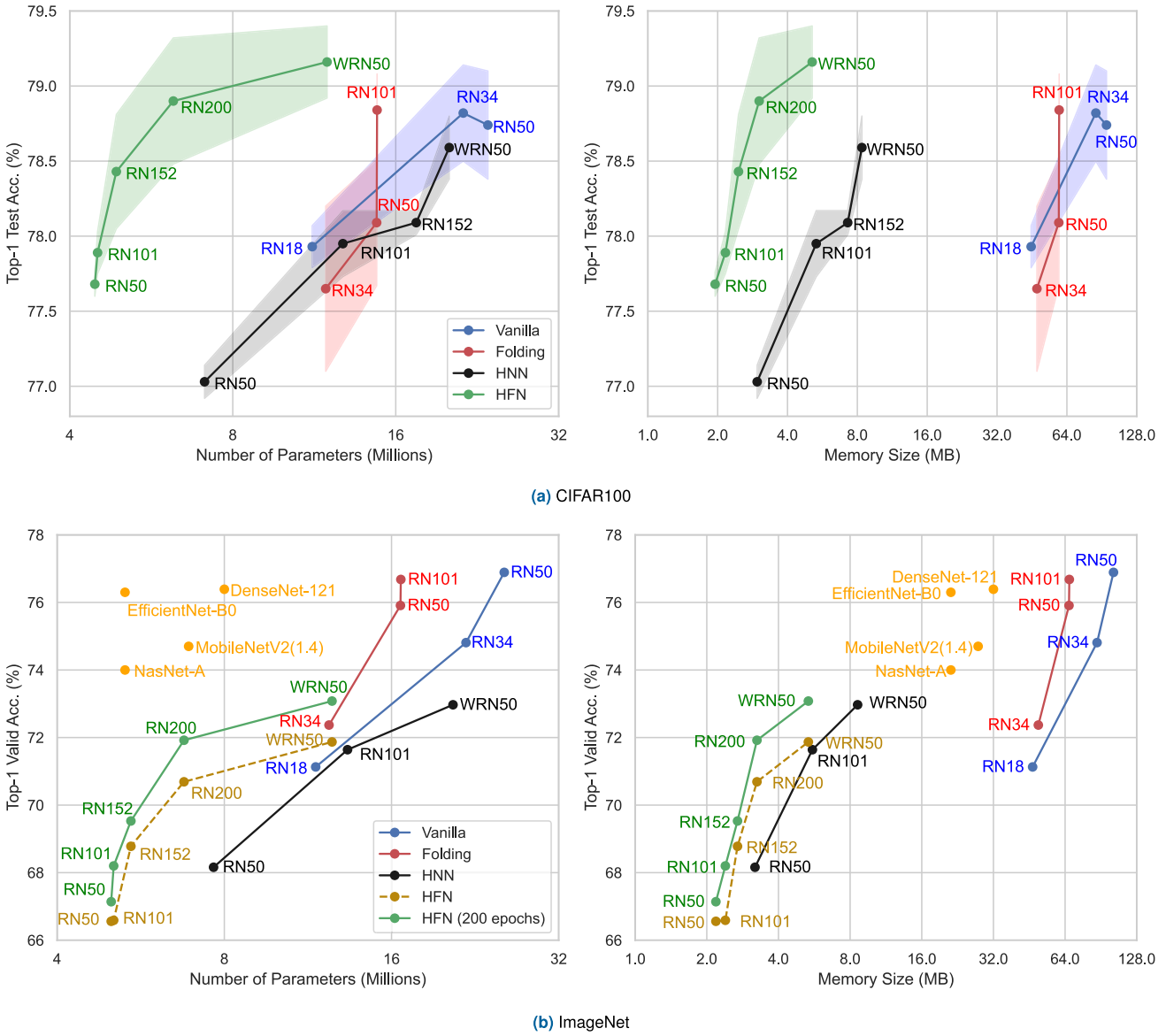
When the same models are compared on ImageNet, Fig. 13b shows that HFN's accuracy advantage scales poorly to a larger dataset. It is no longer the best-performing method, only matching the shallowest weight-learning architectures considered.

A suboptimal train schedule can explain this decay. As shown in Fig. 14, HFN converges to CIFAR100 in 200 epochs, only needing additional epochs for larger models. However, HFN underfits on ImageNet and fails to reach convergence, even when doubling the number of epochs, using a different learning rate scheduler, or using RAdam [92]. This phenomenon is observed in all HFN sizes but in none of the other methods, making it evident that `edge-popup` is unable to exploit the full potential of HFN. Although doubling the number of epochs is enough to match the accuracy of feedforward strong tickets, it contradicts the hypothesis that folded tickets are easier to find.

Nonetheless, HFN is still the most parameter-efficient and tiny: more accurate than models of similar parameter count and one order of magnitude smaller memory size than similarly performing dense models. Popular small models are even more parameter-efficient, but offer a much smaller memory size compression ratio. Although not the focus of this paper, it is also noticeable that folded ResNets with learned weights achieve higher accuracy than previously reported in both datasets, especially with deeper configurations.

E. STRETCHING HFN AFTER TRAINING

This subsection explores the effect of changing the number of iterations after training. Since the goal is to investigate



(a) CIFAR100

(b) ImageNet

FIGURE 13. Comparison of the four methods using different model sizes on two common image classification datasets. HFN is both the most parameter-efficient and the tiniest. (b) also compares with popular efficient models: DenseNet [88], MobileNetV2 [89], NasNet [90], EfficientNet [91].

iterative generality, this section does not use UBN, so all iterations apply strictly the same function.

As discussed above, folding transforms architecture depth into a new dimension: iterations. Where the parameters of a conventional block would receive a gradient per training update, a recurrent block propagates to its parameters a number of gradients equal to its number of iterations, as was covered in Section III-B. It is then tempting to draw a parallelism between train iteration and recurrence iteration.

Fig. 15 investigates if additional gradients from increased iterations at train time have the same effect as additional gradients from increased train epochs. Training ResNet-50 with added iterations at individual folded stages has no positive impact, regardless of whether normalization is affine or not. The best performance is at the original number of iterations, but stretching in training has a very mild impact, suggesting that the learned functions have some degree of iterative generality.

Fig. 16 stretches ResNet-50 at inference time to further investigate this phenomenon, testing for more or fewer iterations than the models were trained for. Similar experiments with weight learning were presented in [53], [58]. If ResNet is learning an ensemble, its accuracy would suffer little from removing blocks but unpredictably from adding additional blocks. However, if these blocks are learning generally iterative functions, a small change in the number of iterations, should have a small effect on performance.

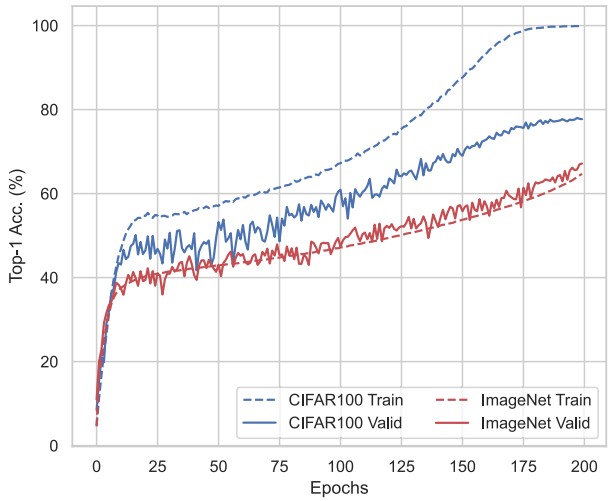


FIGURE 14. Comparison of train graphs for the two datasets: HFN underfits on ImageNet and does not reach convergence.

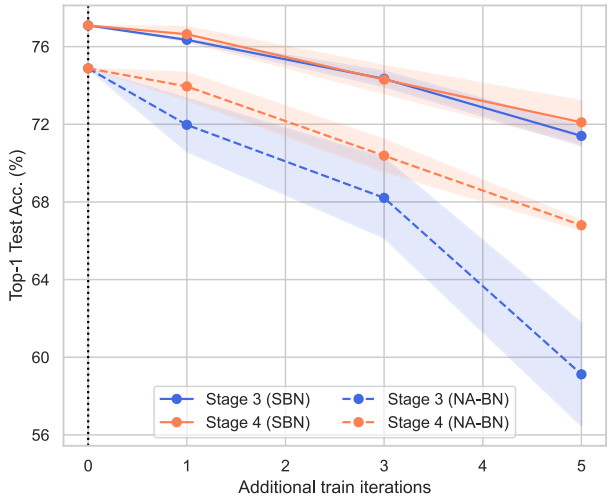
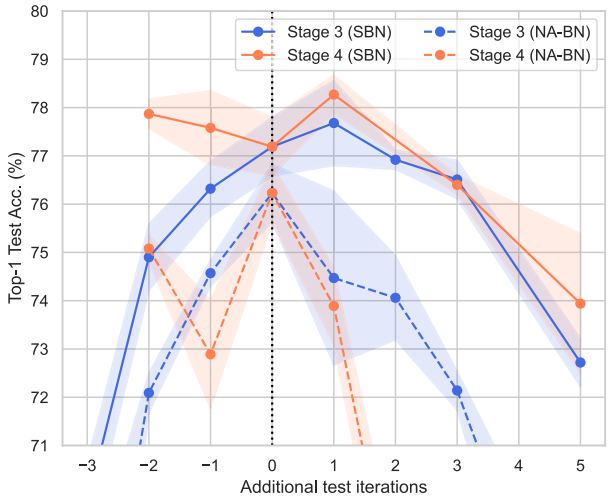
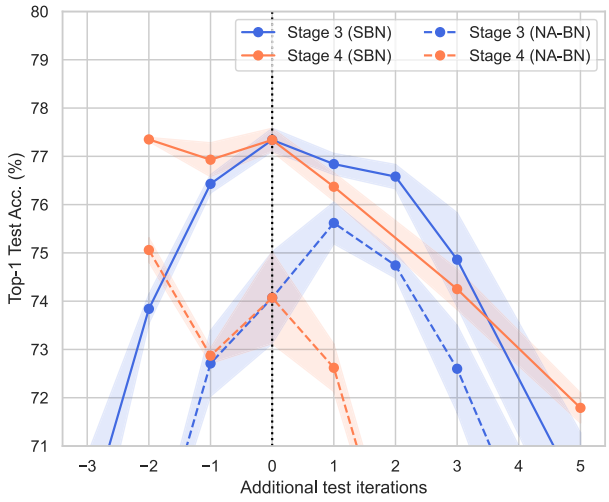


FIGURE 15. Adding iterations to individual stages of ResNet-50 only at train time. SBN:shared affine batchnorm; NA-BN: non-affine batchnorm.



(a) Stretching with learned dense weights.



(b) Stretching with supermask training.

FIGURE 16. Testing ResNet-50 for more iterations it was trained for. Additional iterations are added to an individual folded stage, on independent runs. SBN:shared affine batchnorm; NA-BN: non-affine batchnorm.

The results show precisely that kind of behavior when the modification is small, sometimes even improving performance. Affine batchnorm makes recurrent blocks more resilient to these alterations, and HFN shows similar behavior to its dense counterpart, other than a narrower sweet spot.

Strikingly, in both cases, higher accuracy is observed consistently when completely removing stage 4's folded block at test time, leaving only the downsampling block before the classifier. Although it confirms that these blocks do not perform a compositional transformation in the feature hierarchy, the reason for this phenomenon is not clear yet.

Since complex samples that benefit from additional iterations are a minority [53], increasing the number of iterations for all samples generally has a globally negative effect. A possible way of exploiting this iterative generality is to adjust the number of iterations dynamically on a per-sample basis as in [71] or [66], a mechanism that has also been observed in primate's visual cortex [50].

F. CONCLUSIONS

Table 2 compares the four methods when applied to the same ResNet-50. In the case of CIFAR100, the folded recurrent network hides a more accurate and smaller ticket than the feedforward model, even close in performance to the models with dense learned weights. Although HF-ResNet-50 falls slightly short compared to its feedforward version, HF-ResNet101, with the same architecture but more iterations, matches it in performance.

The model compression potential of HFN is better appreciated when comparing models of similar accuracy in Table 3: even on ImageNet, an HFN with similar performance to a

Table 2. Comparison of the four methods on the same model, ResNet50. F: folded; H: HNN; HF: HFN; RN:ResNet; WRN:ResNet of 2x width.

Dataset	Model	Top-1 Acc. [%]	Parameters (Millions)	Size [MB]	Size Reduction
CIFAR100	RN50	78.74	23.68	94.82	-
	F-RN50	78.09	14.75	59.07	1.61x
	H-RN50	77.03	7.10	2.96	32.07x
	HF-RN50	77.68	4.45	**1.95**	**48.55x**
ImageNet	RN50	76.89	25.55	102.22	-
	F-RN50	75.91	16.60	60.47	1.54x
	H-RN50	68.16	7.65	3.19	32.07x
	HF-RN50	67.14	5.00	**2.18**	**46.80x**

Table 3. Size comparison of models of similar accuracy, using the proposed compression.

Dataset	Model	Top-1 Acc. [%]	Parameters (Millions)	Size [MB]	Size Reduction
CIFAR100	RN34	78.82	21.32	85.31	-
	F-RN101	78.84	14.78	59.28	x1.44
	H-WRN50	78.59	20.10	8.37	x10.19
	HF-RN200	78.90	6.21	**3.02**	**x28.27**
ImageNet	RN18	71.13	11.68	46.75	-
	F-RN34	72.37	12.35	49.41	x0.95
	H-RN101	71.64	13.33	5.56	x8.42
	HF-RN200	71.92	6.77	**3.25**	**x14.39**

dense feedforward model occupies one order less memory size. As a technique for model compression, combining folding and supermask training results in a superior accuracy-size tradeoff that produces residual networks capable of fitting in just 2 MB, small enough for on-chip memory. From the standpoint of specialized hardware design, the sparsity, ternary nature, and, most importantly, the reduction in data transfers from external memory provides computation time and energy advantages that far outweigh the additional depth necessary to match in accuracy the baseline methods. In the case of 32 bit operations on 45 mm CMOS, a DRAM read consumes 640 pJ, two orders of magnitude more than the 3.7 pJ consumed by a floating-point multiplication [82]. On a specialized architecture such as [4], HFN's reduction of memory reads guarantees at least a proportional two order of magnitude reduction of energy consumption.

Although folding ResNet is a detriment to its performance when learning dense weights, for supermask training it is beneficial. Independently of sparsity, depth, or width, folded residual networks contain smaller and stronger tickets than their feedforward counterparts. Furthermore, these models outperform densely learned models with the same number of parameters, and occupy one order of magnitude less memory than those of similar performance.

V. DISCUSSION

Multiple improvements on the `edge-popup` algorithm have been proposed, which future work may apply to the method presented in this paper: learning signs instead of using random binary weights [42]; learning weight scales instead of the fixed modulus of SC initialization [34]; annealing sparsity [5], [93] or enforcing it at a global level instead of per-layer [94]; an improved straight-through estimator [95]; a better strategy for using biases [93]. Strategies for pruning trained models with masks could be adapted to supermask training, such as learning the pruning thresholds [11] instead of calculating them from top-k% at every update step.

HFN's effective weights can be equivalently regarded as ternary or binary, depending on whether the unselected weights are considered zero-weighted or simply not part of the model. Independently of this detail, we believe this trend should be reconnected with the literature on binary and ternary neural networks, where supermasks originated at first [39]. Essentially, supermasks are just a binarization—ternarization in [42]—of the scores, which act as latent weights. Recent advances in binarized neural network optimization have shown that learning can be done more efficiently without latent weights [96].

Pruning and quantization are two central techniques for implementing energy-efficient hardware inference accelerators. Supermask training concurrently learns and prunes a quantized network. Although many advances have been made by using ternary effective weights, it has been shown that supermask training can be expanded to learning weights quantized at a larger precision [43]. This points to a future trend of training methods that blend learning, pruning, and quantization into a concurrent process.

Previous work has shown that introducing recurrent connections in computer vision model is beneficial for weight learning. This paper expands this argument to supermask training. ResNets with skip-connections and recurrent loops include most of the lateral connections observed in the visual cortex, but still miss one type: lateral feedback connections. Additionally, this paper has only considered standard pyramidal ResNets. A study exploring isotropic ResNets with various complex connections was recently published in [76]. Their results could be leveraged for expanding HFN to modern isotropic architectures [97], [98].

Lastly, we have only evaluated HFNs on image classification tasks. Future work should evaluate them in time-series data tasks, considering standard RNN practices such as including LSTM units. Some previous work on RCNN for vision considered recurrent architectures that produced an output at every iteration [64], [72], whereas other methods, including the one studied in this paper, consider a single final output. Future work should also investigate the difference between these two readout strategies.

VI. CONCLUSION

This paper proposes and tests the conjecture that recurrent residual networks contain stronger tickets than their feedforward counterparts. The results show that these strong tickets can be exploited to vastly reduce the memory footprint, and upscaled to match or surpass the accuracy of models with dense learned weights: ResNet-50 is compressed 48.55× to 2 megabytes without a drastic loss in performance, and a ResNet-200 with superior accuracy is compressed 28.27× to just 3 megabytes. However, more than the additional restrictions on the search space imposed by recurrence is needed:

the current supermask training algorithm is unable to find the best possible tickets, especially in the case of a larger dataset. Nonetheless, we find that these recurrent subnetworks with random weights have similar iterative properties to their learned weight counterparts and, furthermore, can be fitted in on-chip memory with a straightforward compression scheme, encouraging efficient edge implementations of computer vision based on recurrent neural networks.

REFERENCES

[1] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 3597–3607.

[2] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's hidden in a randomly weighted neural network?" in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11893–11902.

[3] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Repr.*, 2019, pp. 1–42.

[4] K. Hirose, J. Yu, K. Ando, Y. Okoshi, A. L. Garcia-Arias, J. Suzuki, T. V. Chu, K. Kawamura, and M. Motomura, "Hiddenite: 4K-PE hidden network inference 4D-tensor engine exploiting on-chip model construction achieving 34.8-to-16.0 TOPS/W for CIFAR-100 and ImageNet," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2022, pp. 1–3.

[5] J. Fischer and R. Burkholz, "Plant 'n' seek: Can you find the winning ticket?" 2021, *arXiv:2111.11153*.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 2, pp. 84–90, Jun. 2017.

[8] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 8821–8831.

[9] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," 2019, *arXiv:1906.02243*.

[10] 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.

[11] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. H. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," 2020, *arXiv:2005.06870*.

[12] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[13] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," 2019, *arXiv:1902.09574*.

[14] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" 2020, *arXiv:2003.03033*.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, 2016, pp. 1–14.

[16] A. Dubey, M. Chatterjee, and N. Ahuja, "Coreset-based neural network compression," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 454–470.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[18] K. Grosse and M. Backes, "How many winning tickets are there in one DNN?" 2020, *arXiv:2006.07014*.

[19] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, "Linear mode connectivity and the lottery ticket hypothesis," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3259–3269.

[20] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin, "Drawing early-bird tickets: Toward more efficient training of deep networks," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–13.

[21] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–13.

[22] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 2943–2952.

[23] N. Lee, T. Ajanthan, S. Gould, and P. H. Torr, "A signal propagation perspective for pruning neural networks at initialization," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–16.

[24] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 7, Apr. 2020, pp. 12273–12280.

[25] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–11.

[26] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 6377–6389.

[27] S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh, "Robust pruning at initialization," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–37.

[28] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Pruning neural networks at initialization: Why are we missing the mark?" in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–41.

[29] G. Yuan, X. Ma, W. Niu, Z. Li, Z. Kong, N. Liu, Y. Gong, Z. Zhan, C. He, Q. Jin, S. Wang, M. Qin, B. Ren, Y. Wang, S. Liu, and X. Lin, "MEST: Accurate and fast memory-economic sparse training framework on the edge," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 20838–20850.

[30] K. Sreenivasan, J.-Y. Sohn, L. Yang, M. Grinde, A. Nagle, H. Wang, E. Xing, K. Lee, and D. Papailiopoulos, "Rare gems: Finding lottery tickets at initialization," 2022, *arXiv:2202.12002*.

[31] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 6682–6691.

[32] A. Pensia, S. Rajput, A. Nagle, H. Vishwakarma, and D. S. Papailiopoulos, "Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 2599–2610.

[33] L. Orseau, M. Hutter, and O. Rivasplata, "Logarithmic pruning is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 2925–2934.

[34] J. Diffenderfer and B. Kailkhura, "*Multi-Prize Lottery Ticket Hypothesis*: Finding accurate binary neural networks by pruning a randomly weighted network," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–23.

[35] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, Jul. 2006.

[36] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Netw.*, vol. 115, pp. 100–123, Jul. 2019.

[37] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, "Perturbative neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 3310–3318.

[38] J. Frankle, D. J. Schwab, and A. S. Morcos, "Training batchnorm and only batchnorm: On the expressive power of random features in CNNs," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–28.

[39] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 67–82.

[40] S. Baek, M. Song, J. Jang, G. Kim, and S.-B. Paik, "Face detection in untrained deep neural networks," *Nature Commun.*, vol. 12, no. 1, pp. 1–15, Dec. 2021.

[41] K. Sreenivasan, S. Rajput, J.-Y. Sohn, and D. Papailiopoulos, "Finding nearly everything within random binary networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2022, pp. 3531–3541.

[42] N. Koster, O. Grothe, and A. Rettinger, "Signing the supermask: Keep, hide, invert," in *Proc. Int. Conf. Learn. Represent.*, 2022, 1–37.

[43] Y. Okoshi, A. López García-Arias, K. Hirose, K. Ando, K. Kawamura, T. Van Chu, M. Motomura, and J. Yu, "Multicoated supermasks enhance hidden networks," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 17045–17055.

[44] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi, "Supermasks in superposition," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 15173–15184.

[45] M. M. Aladago and L. Torresani, "Slot machines: Discovering winning combinations of random weights in neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 163–174.

[46] D. Chijiwa, S. Yamaguchi, Y. Ida, K. Umakoshi, and T. Inoue, "Pruning randomly initialized neural networks with iterative randomization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 4503–4513.

[47] A. López García-Arias, M. Hashimoto, M. Motomura, and J. Yu, "Hidden-fold networks: Random recurrent residuals using sparse supermasks," in *Proc. Brit. Mach. Vis. Conf.*, 2021, pp. 1–13.

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

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[50] K. Kar, J. Kubilius, K. Schmidt, E. B. Issa, and J. J. D. Carlo, "Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior," *Nature Neurosci.*, vol. 22, no. 6, pp. 974–983, 2019.

[51] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–9.

[52] K. Greff, R. K. Srivastava, and J. Schmidhuber, "Highway and residual networks learn unrolled iterative estimation," 2016, *arXiv:1612.07771*.

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

[54] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 646–661.

[55] M. Abdi and S. Nahavandi, "Multi-residual networks: Improving the speed and accuracy of residual networks," 2016, *arXiv:1609.05672*.

[56] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, 2016, pp. 1–15.

[57] B. Chu, D. Yang, and R. Tadinada, "Visualizing residual networks," 2017, *arXiv:1701.02362*.

[58] Q. Liao and T. Poggio, "Bridging the gaps between residual learning, recurrent neural networks and visual cortex," 2016, *arXiv:1604.03640*.

[59] P. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 82–90.

[60] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3367–3375.

[61] I. Caswell, C. Shen, and L. Wang, "Loopy neural nets: Imitating feedback loops in the human brain," Stanford Univ., Stanford, CA, USA, Tech. Rep. 110 cs231n, 2016.

[62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[63] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.

[64] A. R. Zamir, T.-L. Wu, L. Sun, W. B. Shen, B. E. Shi, J. Malik, and S. Savarese, "Feedback networks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, Jul. 2017, pp. 1308–1317.

[65] Z. Zhang and C. Jung, "Recurrent convolution for compact and cost-adjustable neural networks: An empirical study," 2019, *arXiv:1902.09809*.

[66] Q. Guo, Z. Yu, Y. Wu, D. Liang, H. Qin, and J. Yan, "Dynamic recursive neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5147–5156.

[67] O. Köpüklü, M. Babaee, S. Hörmann, and G. Rigoll, "Convolutional neural networks with layer reuse," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 345–349.

[68] W. Kang and D. Kim, "Deeply shared filter bases for parameter-efficient convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 7397–7408.

[69] R. S. van Bergen and N. Kriegeskorte, "Going in circles is the way forward: The role of recurrence in visual inference," *Current Opinion Neurobiol.*, vol. 65, pp. 176–193, Dec. 2020.

[70] J. Kubilius, M. Schrimpf, A. Nayebi, D. Bear, D. L. K. Yamins, and J. J. DiCarlo, "CORnet: Modeling the neural mechanisms of core object recognition," *bioRxiv*, 2018, Art. no. 408385.

[71] S. Leroux, P. Molchanov, P. Simoens, B. Dhoedt, T. Breuel, and J. Kautz, "IamNN: Iterative and adaptive mobile neural network for efficient image classification," 2018, *arXiv:1804.10123*.

[72] C. J. Spoerer, P. McClure, and N. Kriegeskorte, "Recurrent convolutional neural networks: A better model of biological object recognition," *Frontiers Psychol.*, vol. 8, p. 1551, Sep. 2017.

[73] P. Savarese and M. Maire, "Learning implicitly recurrent CNNs through parameter sharing," 2019, *arXiv:1902.09701*.

[74] M. Schrimpf, J. Kubilius, H. Hong, N. J. Majaj, R. Rajalingham, E. B. Issa, K. Kar, P. Bashivan, J. Prescott-Roy, K. Schmidt, D. L. K. Yamins, and J. J. DiCarlo, "Brain-score: Which artificial neural network for object recognition is most brain-like?" *bioRxiv*, 2020, Art. no. 407007.

[75] A. Ben Dror, N. Zehngut, A. Raviv, E. Artyomov, R. Vitek, and R. Jevnisek, "Layer folding: Neural network depth reduction using activation linearization," 2021, *arXiv:2106.09309*.

[76] W. Feng, X. Zhang, Q. Song, and G. Sun, "The incoherence of deep isotropic neural networks increases their performance in image classification," *Electronics*, vol. 11, no. 21, p. 3603, Nov. 2022.

[77] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[78] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[79] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[80] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," 2016, *arXiv:1607.08022*.

[81] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–19.

[82] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2014, pp. 10–14.

[83] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[84] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. (2020). *What's Hidden in a Randomly Weighted Neural Network?* [Online]. Available: https://github.com/allenai/hidden-networks

[85] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[86] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[87] NVIDIA. (2021). *Deeplearningexamples/Pytorch/*. [Online]. Available: https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets/resnet50v1.5

[88] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[89] 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.*, Jun. 2018, pp. 4510–4520.

[90] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[91] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[92] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *Proc. ICLR*, Apr. 2020, pp. 1–14.

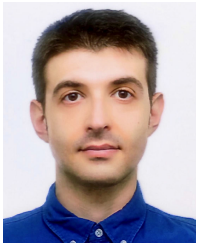
[93] J. Fischer, A. Gadhikar, and R. Burkholz, "Lottery tickets with nonzero biases," 2021, *arXiv:2110.11150*.

[94] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 3599–3608.

[95] R. Dupont, M. A. Alaoui, H. Sahbi, and A. Lebois, "Extracting effective subnetworks with Gumbel-Softmax," 2022, *arXiv:2202.12986*.

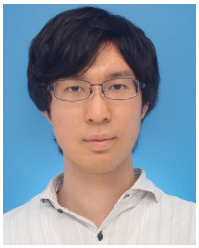
[96] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, "Latent weights do not exist: Rethinking binarized neural network optimization," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[97] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[98] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, and M. Lucic, "MLP-mixer: An all-MLP architecture for vision," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 24261–24272.



ÁNGEL LÓPEZ GARCÍA-ARIAS (Graduate Student Member, IEEE) received the B.E. degree in telecommunications engineering from the Autonomous University of Madrid, Madrid, Spain, in 2017, and the M.S. degree in information science and technology from Osaka University, Osaka, Japan, in 2021. He is currently pursuing the Ph.D. degree in information and communication engineering with the Tokyo Institute of Technology, Yokohama, Japan.

His graduate studies are sponsored by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan, since 2018. His research interests include neural network architecture, non-linear systems, and digital design.



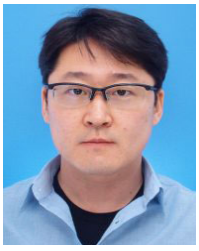
YASUYUKI OKOSHI received the B.E. degree in engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 2017, where he is currently pursuing the M.S. degree.

His research interests include deep learning and computer architecture.

MASANORI HASHIMOTO (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in communications and computer engineering from Kyoto University, Kyoto, Japan, in 1997, 1999, and 2001, respectively. He is currently a Professor with the Graduate School of Informatics, Kyoto University. His current research interests include design for manufacturability and reliability, timing and power integrity analysis, reconfigurable computing, soft error characterization, and low-power circuit design. He was on the Technical Program Committees of international conferences, including the Design Automation Conference (DAC), the International Conference on Computer Aided Design (ICCAD), the International Test Conference (ITC), the Symposium on VLSI Circuits, the Asia and South Pacific Design Automation Conference (ASP-DAC), and the Design, Automation and Test in Europe Conference (DATE). He serves/served as the Editor-in-Chief for *Microelectronics Reliability* (Elsevier) and an Associate Editor for IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, and *ACM Transactions on Design Automation of Electronic Systems* (TODAES).

MASATO MOTOMURA (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Kyoto University, Kyoto, Japan, in 1985, 1987, and 1996, respectively.

In 1987, he joined NEC central research laboratories, where he worked on various hardware architectures, including string search engines, multi-threaded on-chip parallel processors, embedded DRAM field-programmable gate array (FPGA) hybrid systems, memory-based processors, and reconfigurable systems. From 1991 to 1992, he was a Visiting Researcher at the MIT Laboratory for computer science. From 2001 to 2008, he was with NEC Electronics where he led research and business development of dynamically reconfigurable processor (DRP) that he invented. From 2011 to 2019, he was a Professor at Hokkaido University. He has been a Professor at the Tokyo Institute of Technology, Japan, since 2019. His current research interests include reconfigurable and parallel architectures for deep neural networks, machine learning, annealing machines, and intelligent computing in general. He is a member of IEICE, IPSJ, and EAJ. He was a recipient of the IEEE JSSC Annual Best Paper Award, in 1992, the IPSJ Annual Best Paper Award, in 1999, the IEICE Achievement Award, in 2011, and the ISSCC Silkroad Award as a corresponding author in 2018.



JAEHOON YU (Member, IEEE) received the B.E. degree in electrical and electronic engineering and the M.S. degree in informatics (communications and computer engineering) from Kyoto University, Kyoto, Japan, in 2005 and 2007, respectively, and the Ph.D. degree in informatics (information systems engineering) from Osaka University, Osaka, Japan, in 2013.

From 2013 to 2019, he was an Assistant Professor at Osaka University. He is currently an Associate Professor at the Tokyo Institute of Technology, Japan. His research interests include computer vision, machine learning, and system-level design. He is a member of IEICE and IPSJ.