

Rapid Deployment of DNNs for Edge Computing via Structured Pruning at Initialization

Bailey J. Eccles*, Leon Wong[†], and Blesson Varghese*

*School of Computer Science, University of St Andrews, UK

[†]Autonomous Networking Research & Innovation Department, Rakuten Mobile, Inc., Japan

Corresponding author: bje1@st-andrews.ac.uk

Abstract—Edge machine learning (ML) enables localized processing of data on devices and is underpinned by deep neural networks (DNNs). However, DNNs cannot be easily run on devices due to their substantial computing, memory and energy requirements for delivering performance that is comparable to cloud-based ML. Therefore, model compression techniques, such as pruning, have been considered. Existing pruning methods are problematic for edge ML since they: (1) Create compressed models that have limited runtime performance benefits (using unstructured pruning) or compromise the final model accuracy (using structured pruning), and (2) Require substantial compute resources and time for identifying a suitable compressed DNN model (using neural architecture search). In this paper, we explore a new avenue, referred to as Pruning-at-Initialization (PaI), using structured pruning to mitigate the above problems. We develop *Reconvene*, a system for rapidly generating pruned models suited for edge deployments using structured PaI. *Reconvene* systematically identifies and prunes DNN convolution layers that are least sensitive to structured pruning. *Reconvene* rapidly creates pruned DNNs within seconds that are up to $16.21\times$ smaller and $2\times$ faster while maintaining the same accuracy as an unstructured PaI counterpart.

Index Terms—Deep Neural Networks, Edge Computing, Model Compression, Structured Pruning

I. INTRODUCTION

Deep neural networks (DNNs) are used in many applications to process and analyze data at the network edge for mitigating the challenges in sending all the data to the cloud [1]. For instance, security cameras for facial recognition [2] and wearable health monitors [3] benefit from edge computing. The DNN models used in these settings are often over-parameterized for the application task, requiring a large amount of computing resources for training and inference [4], [5].

Embedded and mobile edge devices cannot support large cloud-based DNN models due to computational, memory, and energy constraints [6]. Therefore, model compression methods that reduce the resource requirements of training and inference while preserving task accuracy are used [7]. Compression methods include quantization [8], knowledge distillation [9], neural architecture search [10], and pruning [4], [11].

Model pruning removes specific parameters from over-parameterized and dense DNNs while tailoring models for specialized tasks. In contrast to other model compression methods, model pruning is beneficial in edge computing environments, where optimizing models is required for diverse applications with heterogeneous computational constraints and

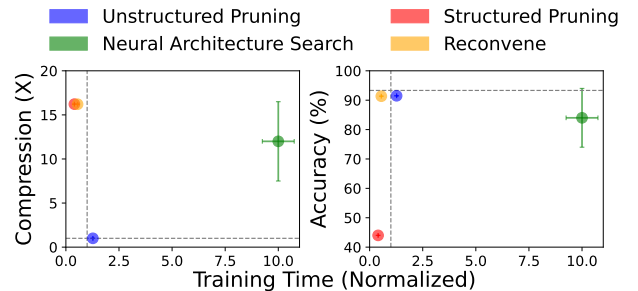


Fig. 1: Evaluating model compression methods to reduce parameter count of VGG-16 (on the CIFAR-10 dataset) by $50\times$. Dashed lines are the baseline values of an uncompressed dense VGG-16. The bar for neural architecture search includes the discovery time for generating a range of compressed models with different levels of compression and accuracy.

capabilities [10], [12]. For example, edge-based model training paradigms, such as federated learning, make use of model pruning to expedite the training time of straggler devices [13], [14]. Model pruning is categorized as unstructured pruning (UP) and structured pruning (SP). UP sets parameter weights to zero, while SP removes groups of parameters.

Figure 1 shows runtime differences between compression methods. UP retains model accuracy, whereas SP enhances compression and training speed. Compared to pruning, neural architecture search (NAS) finds a range of models from a vast search space but takes longer for searching and training [10].

Pruning and NAS offer many benefits for edge computing. However, there is a three-fold challenge that impacts the deployment of compressed models in the edge setting: (1) Retaining the accuracy of the compressed model similar to that of the original dense model, (2) Achieving model compression that empirically decreases training and inference latency and model size, and (3) Discovering a pruned model rapidly and efficiently. While existing methods can address up to two of these challenges simultaneously, they do not address all three at the same time (highlighted in Table I). For example, models generated by SP are smaller, faster, and easily discoverable but often have low accuracies, and therefore, lack usability for accuracy-critical edge applications. An ideal system for pruning will be underpinned by a method that addresses all three challenges considered above.

This paper introduces *Reconvene*, a system that addresses

TABLE I: Comparing unstructured/structured pruning (UP, SP), neural architecture search (NAS), and Reconvene.

Characteristics of compressed models	UP	SP	NAS	Reconvene
Maintain high accuracy	✓	✗	✓	✓
Smaller and faster	✗	✓	✓	✓
Rapidly discovered	✓	✓	✗	✓

the above three challenges for pruning large DNN models to create compressed models suitable for resource-constrained devices in edge computing environments. Reconvene achieves this by using a novel combination of both unstructured and structured pruning methods at model initialization (PaI; i.e., before training) and is the first system to maintain model accuracy up to extreme levels of model compression in this category [15], [16]. Other pruning systems apply pruning after model training, where a significant amount of computation is required to fine-tune the remaining parameters to regain accuracy [17]. Reconvene preserves model accuracy by retaining important parameters by applying unstructured pruning at model initialization. In addition, less significant layers of a DNN, which contain parameters that least contribute to accuracy, undergo structured pruning. Reconvene adopts a disciplined approach to determine the significance of parameters and their contribution to the importance of DNN layers. This allows for a more precise structured pruning approach that reduces model size and accelerates training and inference while maintaining accuracy.

Reconvene produces pruned models with seconds at model initialization that are up to $16.21\times$ smaller and $2\times$ faster while maintaining the same accuracy to the dense model. Our **research contributions** are as follows:

- The development of Reconvene, a system that is underpinned by a novel pruning at initialization (PaI) method for convolutional neural networks to determine which layers are sensitive to structured pruning systematically.
- The experimental demonstration that selective structured pruning based on layer sensitivity maintains accuracy on par with unstructured PaI methods.
- The empirical demonstration that structured PaI can be used to rapidly search for optimized pruned models and then train edge DNNs with lower resource overheads than neural architecture search.

The remainder of this paper is organized as follows. Section II discusses background content to model pruning. Section III presents the Reconvene system. Section IV presents the experimental results. Section V discusses related work, and Section VI concludes the paper.

II. BACKGROUND

In this section, we present pruning methods, specifically unstructured and structured pruning, in addition to pruning at initialization (PaI) and its relevance to neural architecture search (NAS) when searching for lightweight models within a large DNN suitable for deployment at the edge.

A. Unstructured and Structured Pruning

DNN pruning aims to reduce the computational complexity of models by removing redundant parameters (weights or connections). An ideal pruning method will prioritize the removal of parameters that contribute the least to model accuracy for maintaining usability after compression. Pruning methods are categorized as *Unstructured* and *Structured* pruning.

Unstructured Pruning (UP) masks individual parameters by setting their value to zero [4], [18]. A ranking algorithm determines which parameters to mask using simple metrics such as the magnitude of the weights [19], to more complex criteria using training information [20], [21]. By masking parameters, the model becomes sparse, referred to as a sparse model, and the original model is referred to as a dense model. While UP maintains model accuracy between ~ 50 - 90% sparsity depending on the model, dataset, and pruning method [5], sparse models only provide runtime performance improvements in cloud scenarios [22] or where libraries for sparse matrix formats are available [23], [24].

Edge devices may not be equipped with hardware accelerators, such as GPUs [25], or may not support sparse matrix representations and libraries [13], [23]. Consequently, sparse models on the edge have limitations. Firstly, scattered sparsity in dense convolutions leads to irregular memory access patterns, which hinder both model training and inference [26]. Secondly, since zeroed parameters still consume the same memory as non-zero parameters, there is no gain in memory efficiency [12], [23]. Figure 1 summarizes UP: high accuracy at the cost of little performance benefits.

Structured Pruning (SP) removes groups of parameters such as filters, channels, or layers [11], [27]. SP results in a spatially smaller pruned model [28], beneficial to edge scenarios with a high demand for models with low memory, energy, and inference footprints [17]. However, obtaining high-quality pruned models is challenging since: (1) SP is oriented towards runtime performance improvements. Therefore, profiling every prospective model from a large search space can take hours [29] to days [17] to find a single high-quality pruned model. (2) At higher sparsities, essential parameters are inevitably removed; fine-tuning is required to regain accuracy, which can take many times longer than the original model training time for complex datasets [4]; instead, training a new model of the same size from scratch may result in better accuracies [30]. Figure 1 summarizes SP: Improved runtime performance at the cost of model accuracy.

B. Pruning at Initialization

Typically pruning occurs after [11], [18] or during [22] model training. However, recent pruning literature explores pruning at network initialization (PaI), where before training, it is possible to discover a sub-network of randomly initialized parameters that, when fully trained, can match the accuracy of the original dense network [19]. Existing PaI literature focuses on unstructured PaI. However, recently, the feasibility of structured PaI has been explored [16], [21].

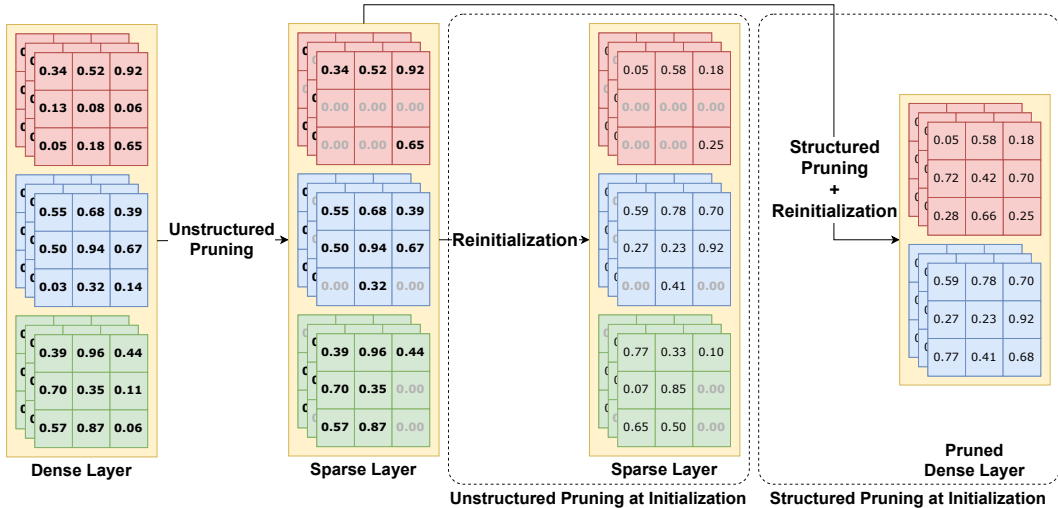


Fig. 2: Different pruning at initialization (PaI) methods applied to a convolutional layer. UPaI prunes and then reinitializes the remaining parameters. SPaI redistributes the parameters such that a smaller layer of only dense channels is created.

Unstructured Pruning at Initialization (UPaI) involves the UP of a dense network, then reinitializing the remaining parameters before training [19]. UPaI can match the accuracy within 1% of a dense model up to $\sim 98\%$ sparsity [15], [20], [31]. The first three stages in Figure 2 show the generalized approach of UPaI. While UPaI presents the opportunity to accelerate training using the sparse model as a drop-in replacement to the original dense model, it encounters challenges in edge scenarios for the same reasons as UP [12].

Structured Pruning at Initialization (SPaI) extends UPaI for improved runtime performance. While UPaI produces a sparse model, SPaI introduces an additional step before reinitialization: the model is pruned using SP. First, this SP spatially compresses the model, and second, sparse layers are converted into dense layers of the same parameter count, which improves hardware utilization [32]. The first two and the last column in Figure 2 show SPaI. This example converts a 33% sparse 3-channel layer into a dense 2-channel layer with the same parameter count. SPaI presents an opportunity for edge-compatible pruned models to be discovered within seconds, significantly outperforming NAS in search time [10]. In addition, SPaI has considerably lower overheads than NAS [16], allowing for execution on an edge device to create pruned models tailored for the device [12], [33].

Structured PaI Challenges: When implementing SPaI as described in Figure 2 raises the following questions: *Do dense layers from SPaI achieve the same accuracy as sparse layers from UPaI when both have the same number of parameters?* Recent literature suggests that where an individual parameter is located within a layer holds no significance for UPaI; instead, the layer-wise sparsity ratio is more critical to model accuracy [15], [16]. Therefore, SPaI, in theory, should achieve close to, or the same, accuracy as UPaI. Figure 3 shows that SPaI maintains accuracy close to UPaI up to $\sim 90\%$ before quickly collapsing. This generally holds true for many models and datasets [15]. However, achieving higher sparsity

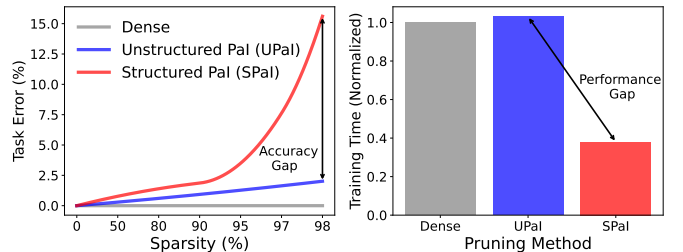


Fig. 3: Pruning at initialization (PaI) methods for VGG-16 (CIFAR-10). UPaI maintains model accuracy without improving runtime performance, while SPaI improves performance but reduces accuracy.

SPaI ($>90\%$) with matching accuracy to UPaI will allow for smaller pruned models to be discovered (within seconds) and trained (in a fraction of the time of dense) on edge devices. Currently, insights are limited on SPaI [16]. *This paper focuses on addressing this challenge - minimizing the accuracy and performance gap between UPaI and SPaI.*

III. RECONVENE

Reconvene is the proposed system to facilitate rapid pruning of DNN models for edge deployment. It is underpinned by an SPaI method that closely matches UPaI accuracies. This produces highly compressed models with low training/inference latencies. Reconvene can be used to deploy a model across a range of heterogeneous edge devices where each model instance is selectively pruned on initialization for that device. Reconvene is suitable for use cases that require models to be tailored to resource availability or capability for federated learning using small devices [13], [14], [34].

A. Motivation

Existing model pruning systems generally comprise only one of the two types of pruning. In cloud-based systems, the

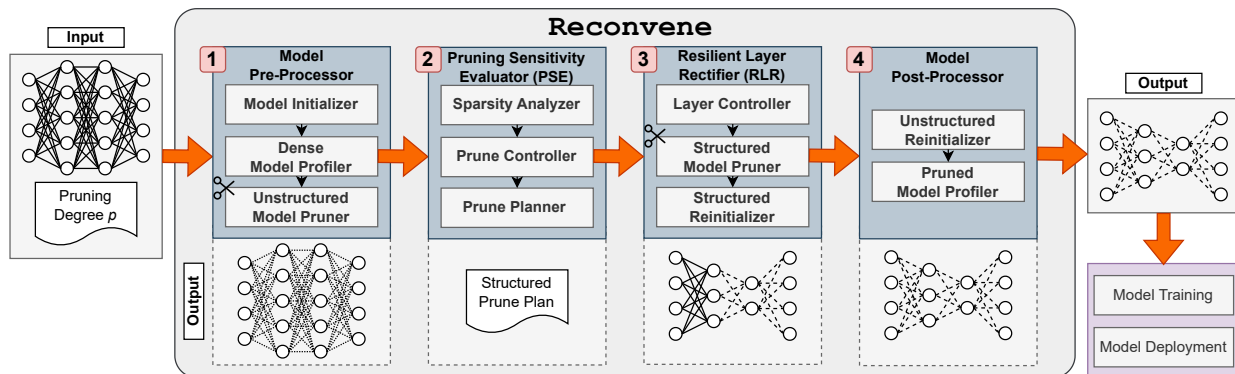


Fig. 4: System overview of Reconvene.

emphasis is on model accuracy since significant computational resources are available [19]. However, the efficiency of model discovery and model latency must be considered given the operational costs and constraints [10]. On the other hand, when it comes to edge computing, model pruning methods aim to balance model accuracy with the stringent resource constraints of devices. While accuracy is reduced, the goal is typically to achieve significant model compression without compromising performance [4]. In recent years, hybrid systems that utilize both unstructured and structured pruning have been shown to strike a balance between accuracy and computational demands post-training [12], [35]. SPaI aims to achieve a balance similar to post-training hybrid pruning methods. However, PaI also offers the advantage of improved training efficiency [16], [21]. Thus, SPaI enables models to be trained on edge devices that have limited computational and memory resources. In cloud environments, it can reduce operational costs [10].

Existing SPaI methods [16] have the following limitations that significantly reduce model accuracy. First, they fully reparameterize sparse models into pruned models, thereby removing the fine-grained accuracy-preserving properties of unstructured pruning [15], [16]. Second, they apply the same pruning method to all model layers. This inherently prunes important layers while under-pruning redundant layers [15], [16]. These limitations have led to SPaI models with worse accuracy than training a smaller model from scratch [16].

Reconvene addresses the above limitations by incorporating a two-step process to SPaI that determines how sensitive each layer is to structured pruning. This allows for Reconvene to control the amount and type of pruning of each layer to maximize model compression while minimizing accuracy loss.

B. System Overview

Figure 4 provides an overview of Reconvene comprising four modules: *Model Pre-Processor*, *Pruning Sensitivity Evaluator (PSE)*, *Resilient Layer Rectifier (RLR)*, and *Model Post-Processor* as part of the SPaI pipeline. The PSE and RLR modules prepare a pruned model for edge training and deployment, especially considering that SPaI, on its own, does not maintain model accuracy with increased levels of pruning (Figure 3). Each module is detailed below.

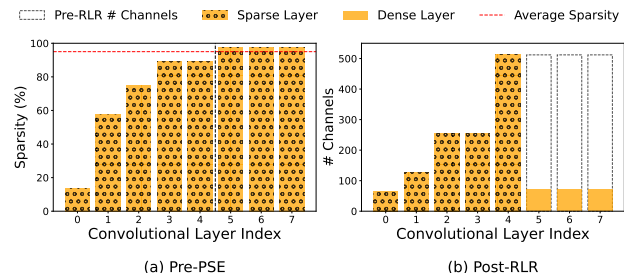


Fig. 5: VGG-11 (CIFAR-10, $p = 0.95$) before and after Reconvene. Layers to the left of the vertical dashed line are considered to be sensitive to pruning and, thereby, do not undergo structured pruning in the RLR.

Input: The end-user chooses a dense model and a pruning degree $p \in [0, 1]$. For example, $p = 0.8$ prunes 80% of the model parameters.

Model Pre-Processor: The input model undergoes UP to the pruning degree p in this module using the following three components. First, the model is initialized in memory as a dense model via the *Model Initializer* component that loads the dense input weights into the chosen model architecture. Next, the *Dense Model Profiler* is used to gather runtime metrics of the original model. Using a synthetic image input over several samples, the dense model is profiled for memory consumption, model size, and CPU/GPU latency. Finally, the *Unstructured Model Pruner* utilizes magnitude pruning [19] to apply unstructured pruning to the dense model. The output from this component (and module) is a sparse model pruned to the degree p . Reconvene is designed to be interoperable with existing model pruning systems. As such, the unstructured pruning method is fully configurable. Magnitude pruning is used by default as it is effective across a wide range of model architectures and datasets [15]. However, this can be substituted for any other UP method, such as SynFlow [20], by user choice.

Pruning Sensitivity Evaluator (PSE): PSE evaluates the sensitivity of each layer to pruning by contrasting its sparsity with the average sparsity of the global model. First, the *Sparsity Analyzer* calculates the per-layer sparsity of the model by examining the sparsity pattern of the sparse model created

by the *Unstructured Model Pruner*. Next, the *Prune Controller* determines the sensitivity of each layer to pruning by comparing the sparsity of that specific layer with the average sparsity of the sparse model. Layers with minimal unstructured pruning (low sparsity, under the average) are regarded as sensitive to pruning, while resilient layers contain a large number of redundant parameters and are unaffected by unstructured pruning. Figure 5 (a) shows the state of various convolution layers after unstructured pruning but before PSE, named pre-PSE. Each layer has a different sparsity, and the red line is the average sparsity of the model. In the *Prune Controller*, sensitive layers are those which have a lower sparsity than the model average, whereas resilient layers are those which have a higher sparsity than the average. Determining layer sensitivity is captured in the following equation where S_l is the sparsity of layer l and S_{Avg} is the average model sparsity:

$$PSE(l) \begin{cases} False & S_l \geq S_{\text{Avg}} \\ True & S_l < S_{\text{Avg}} \end{cases} \quad (1)$$

Finally, the *Prune Planner* calculates how much each resilient layer should be pruned via structured pruning as the product of the number of channels in the layer by the sparsity of the layer (Section III-C) and passes this prune plan to the RLR.

Resilient Layer Rectifier (RLR): The RLR applies structure pruning to the resilient layers. Using the structured pruning plan from PSE and *Layer Controller*, RLR applies SPaI to only the resilient layers. First, the number of channels is reduced proportionally to the layer density using the *Structured Model Pruner*. Next, the remaining channels are reinitialized as dense parameters using the *Structured Reinitializer*. For instance, a sparse layer with 100 channels, exhibiting a 95% sparsity following UP, is transformed into a dense layer consisting of 5 channels. Figure 5 (b) illustrates the post-RLR state of the convolutional layers in Figure 5 (a). The extent to which each resilient layer is pruned following SPaI is illustrated by the dashed outline, which indicates the number of channels present before the application of SPaI.

Model Post-Processor: Post-RLR, the remaining sensitive layers are reinitialized so that the model is completely reinitialized with pseudo-random Kaiming weights [15] to complete pruning at initialization using the *Unstructured Reinitializer*. After which, the remaining model contains a mix of reinitialized sparse and pruned layers and is ready for training and deployment. In addition, the pruned model is profiled for the metrics seen in the pre-processor to gather empirical speedup and compression metadata. The metadata determines what category of edge device the pruned model can be trained and deployed to.

Output: Reconvene outputs an initialized model pruned by p , which is ready for training and edge deployment using typical training hyperparameters (see Table II).

C. Implementation

Reconvene is implemented using Python 3.11.4, PyTorch 2.0.1, Torchvision 0.15.2, and CUDA 11.7 and is intended to be used as a straightforward and lightweight

Algorithm 1: Pruning Sensitivity Evaluator (PSE)

Data: Convolutional layer l with a weights matrix W with shape $(N_{\text{channels}}, N_{\text{weights}})$, Global average sparsity of the model S_{Avg}

Result: Number of channels for layer l

```

1  $N_{\text{total}} \leftarrow 0; N_{\text{zero}} \leftarrow 0; i \leftarrow 0$ 
2 while  $i < N_{\text{channels}}$  do
3    $j \leftarrow 0$ 
4   while  $j < N_{\text{weights}}$  do
5      $N_{\text{total}} \leftarrow N_{\text{total}} + 1$ 
6     if  $W_{i,j} == 0$  then
7        $N_{\text{zero}} \leftarrow N_{\text{zero}} + 1$ 
8     end
9      $j \leftarrow j + 1$ 
10  end
11   $i \leftarrow i + 1$ 
12 end
13  $S_l \leftarrow \frac{N_{\text{zero}}}{N_{\text{total}}}$  // Pruning degree of  $l$ 
14 if  $S_l \geq S_{\text{Avg}}$  then
15   return  $\lceil N_{\text{channels}} \cdot S_l \rceil$  // Resilient layer
16 end
17 else
18   return  $N_{\text{channels}}$  // Sensitive layer
19 end

```

Algorithm 2: Resilient Layer Rectifier (RLR)

Data: Original unstructured sparse model M , List of resilient layer indices R , List of pruned channel sizes for resilient layer indices C

Result: Reinitialized pruned model M'

```

1  $M' \leftarrow M$  // Initialize  $M'$  based on  $M$ 
2 foreach layer  $l$  in  $M$  do
3   if  $l$  is in  $R$  then
4      $M'_l \leftarrow \text{Structured Prune}(l, C_l)$ 
5      $M'_l \leftarrow \text{Reinitialize}(M'_l)$ 
6   end
7 end
8 return  $M'$ 

```

system for determining the amount to which each layer needs to be pruned during SPaI with the objective of edge-centric pruned models. The Reconvene PSE and RLR algorithms are provided below as Algorithm 1 and Algorithm 2, respectively.

Pruning Sensitivity Evaluator (PSE): Algorithm 1, is the first step in preparing a sparse model M for SPaI. PSE has two sub-steps: Step 1 - Calculate the layer-wise sparsity of each layer, and Step 2 - Adjust the number of channels in each SPaI layer. Step 1 is achieved by iterating through the unstructured pruned layers and totaling the number of non-zero and zero parameters (sparse parameters) - Algorithm 1 Line 1-12. The sparsity of the layer, or S_l , is the fraction of zero parameters divided by total parameters - Algorithm 1 Line 13. At this point, layer l only contains unstructured sparsity. Nonetheless,

TABLE II: Baseline model results and training hyperparameters for a production quality dense VGG-16, ResNet-20, and ResNet-50 models.

	VGG-16	ResNet-20	ResNet-50
Dataset	CIFAR-10	CIFAR-10	Tiny ImageNet
Parameters (M)	14.72	0.27	25.56
Size (MB)	56.2	1.1	100.1
Accuracy (%)	93.32	91.68	55.48
# Epochs	160	160	200
Batch Size	128	128	256
Learning Rate	0.1	0.1	0.2
Milestone Steps ¹	80, 120	80, 120	100, 150

through SPaI, the objective is to create a structured pruned layer, denoted l' , while maintaining the same parameter count as l . This is achieved by reducing the number of channels, N_{channels} in l proportionally to S_l . To accomplish this, PSE initially assesses whether l should be pruned by computing its sensitivity according to Equation 1, as outlined in Algorithm 1 Line 14. If l is resilient, then N_{channels} is scaled by S_l and rounded up to the nearest full channel - Algorithm 1 Line 15. Otherwise, the layer is sensitive and remains untouched - Algorithm 1 Line 18.

Resilient Layer Rectifier (RLR): Algorithm 2, is the second step in Reconvne by applying SPaI and reinitialization of the remaining parameters. A copy of which layers are resilient to pruning and the number of channels (N_{channels}) they should be reduced to are provided as lists R and C , respectively. Next, sparse model M is iterated, and for each resilient layer, l in R undergoes structured pruning to the size C_l - Algorithm 2 Line 4. Finally, the pruned layer is fully reinitialized with random dense parameters - Algorithm 2 Line 5. Each step updates the pruned model reference M' and is then returned - Algorithm 2 Line 8.

IV. EXPERIMENTS

This section first presents the experimental setup in Section IV-A and then considers four aspects of Reconvne:

(1) **Method Validation:** Validating Reconvne modules, namely *Pruning Sensitivity Evaluator (PSE)* and *Resilient Layer Rectifier (RLR)* across a range of sparsities, models, and datasets for evaluating model accuracy, compression, and CPU/GPU speedup (Section IV-B).

(2) **Training Benefits:** Evaluating the accelerated training time and final accuracy of pruned models created with Reconvne and comparing them against existing SPaI/UPaI methods and training a smaller model from scratch (Section IV-C).

(3) **Pruned Model Quality:** Comparing Reconvne to existing SPaI pruning systems and the runtime performance metrics of the pruned models across a range of sparsities (Section IV-D).

(4) **System Overheads:** Contrasting the low overheads of Reconvne against exhaustive NAS methods examining both total time and memory requirements for creating compressed models (Section IV-E).

¹Learning rate drops by a factor of gamma, 0.1, at each milestone step. All models use the SGD optimizer, momentum of 0.9 and weight decay of 0.0001.

A. Experimental Setup

Three DNN models trained on the CIFAR-10 [36] and Tiny ImageNet [37] datasets are considered. The first is VGG-16 [38] trained on CIFAR-10, serving as a straightforward feedforward model. The other two, ResNet-20 and ResNet-50 [39], are trained on CIFAR-10 and Tiny ImageNet, respectively, representing models with branching structures. These models and datasets are widely recognized for their production quality and are frequently employed as benchmarks in pruning literature [5]. Some experiments utilize VGG-19 (including Reconvne) when other methods do not report VGG-16 as a baseline in the literature. In addition, alternate versions of each baseline are used to compare pruned large models to smaller dense versions. For example, VGG-11 and ResNet-8 are a shallower version of VGG-16 [38] and ResNet-20 [39], respectively.

Models, Datasets, and Hyperparameters: The VGG models are those that have one fully connected layer [19], and ResNets are the default configurations [39] for their respective datasets. CIFAR-10 consists of 50,000 training images and 10,000 test images of the dimension $32 \times 32 \times 3$ divided equally across 10 classes. Tiny ImageNet is a subset of ImageNet consisting of 100,000 training images and 10,000 test images of the dimension $64 \times 64 \times 3$ divided equally across 200 classes. The baseline results are obtained using the training routine of OpenLTH (a PaI framework) [19] using the hyperparameters in Table II.

Testbed: We use an AMD EPYC 7713P 64-core/128-thread CPU and two Nvidia RTX A6000 GPUs to train, profile, and prune the Tiny ImageNet models, as such resources are representative of those in a cloud data center. CIFAR-10 experiments are carried out with an Intel i9-13900KS 24-core/32-thread CPU and an Nvidia RTX 3080 GPU comparable to an edge server that may be used in a production setting.

Trial Counts and Reporting Methods: All experiments were carried out three times. Model performance metrics, such as accuracy, memory usage, and latency, are presented in tables and figures as the mean from all experiments accompanied by confidence intervals spanning one standard deviation.

Pruning Setup: Pruning experiments are carried out across six different sparsities $\{50, 80, 90, 95, 97, 98\}$ grouped into three difficulties from easy to hard. Trivial sparsities (Easy) $\{50, 80\}$ are those in which even random pruning will match unstructured pruning. Matching sparsities (Medium) $\{90, 95\}$ are those in which benchmark methods perform well and can still match the unpruned dense model accuracy. Extreme sparsities (Hard) $\{97, 98\}$ are those in which the accuracy of the models generated by unstructured pruning methods is lower than unpruned dense models.

Pruning Metrics: The metrics reported in this paper are similar to those adopted in the literature [5], [22], [40], [41]. Accuracy is reported as absolute top-1 test accuracy when comparing pruned models from the same baseline model or relative top-1 test accuracy δ (Δ) when the baseline model accuracy can not be replicated across different pruning

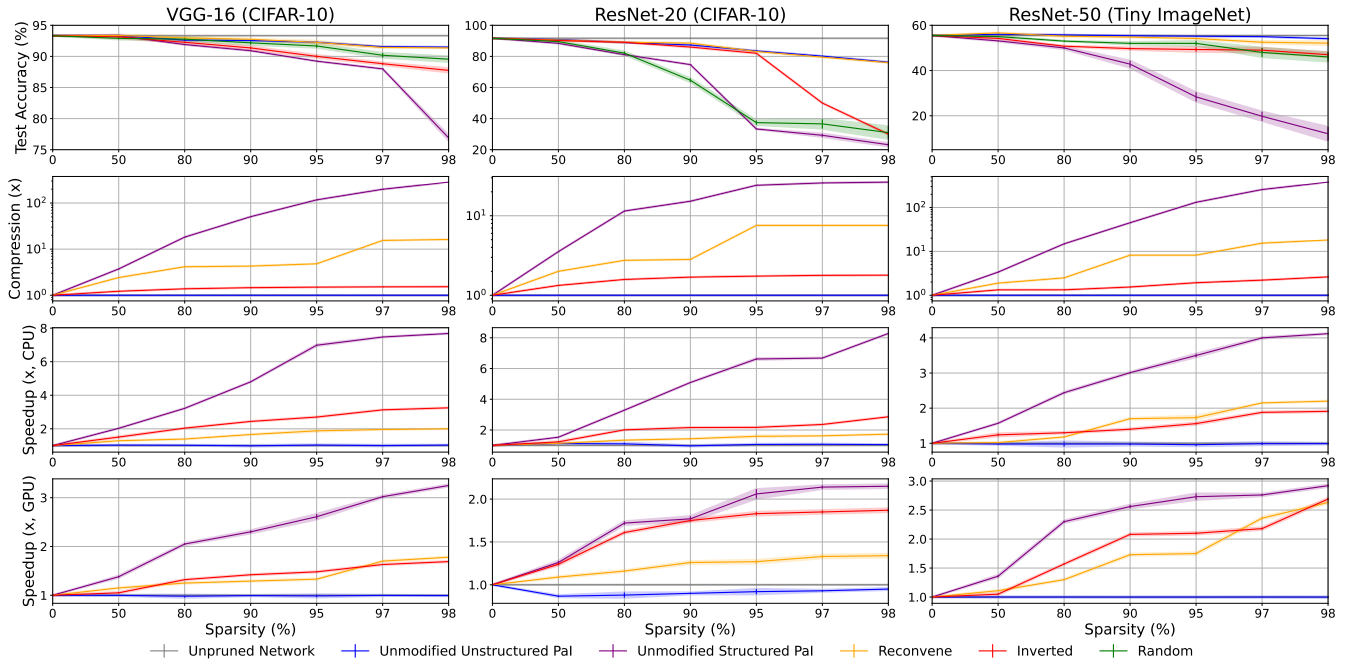


Fig. 6: Comparing test accuracy, model compression, and CPU/GPU speedup across a range of sparsities for a selection of UPaI and SPaI pruning methods for VGG-16 and ResNet-20 models on CIFAR-10 and ResNet-50 on Tiny ImageNet datasets. Random is plotted only for accuracy tests because of its large variance in compression and speedup during random pruning.

systems [5], [41]. The mean model inference speedup of three trials is calculated as $\frac{\text{Mean Dense Model Latency}}{\text{Mean Compressed Model Latency}}$. Compression is calculated as $\frac{\text{Dense Model Size}}{\text{Compressed Model Size}}$ where model size is the size of the model in storage. Note that this is different from using FLOPs and parameter count to calculate theoretical speedup and model size, respectively [5]. We have utilized performance metrics that are empirically observed to showcase the practical benefits in real-world deployment scenarios [42].

B. Validation of Reconvene

In this experiment, the PSE module for determining layer sensitivity (Equation 1) is empirically validated against baseline methods considered below, as well as an inverted sensitivity method to demonstrate that Reconvene maintains the same model accuracy as UPaI methods across a range of sparsities. Thereby, all confounding variables are eliminated, such as differences in training hyperparameters [40] or model architecture [5]. Each method is tested on the same baseline model, testbed, and set of hyperparameters. The methods considered in Figure 6 are as follows: (1) *Unpruned Network*: Dense fully trained model representing maximum model accuracy, and is used as the reference for compression and speedup calculations. (2) *Unmodified UPaI*: Sparse model pruned with a magnitude-based unstructured pruning method [19] as described in Section II-B. (3) *Unmodified SPaI*: Pruned model following reparameterization of the *Unmodified UPaI* model as described in Figure 2. (4) *Reconvene*: Reconvene as described in Section III. (5) *Inverted*: Reconvene, however, the underlying method of the PSE Prune Controller component is inverted. In other words, structured pruning occurs on sensitive

layers in the RLR. (6) *Random*: Reconvene, however, the PSE Prune Controller randomly assigns layers as sensitive.

Figure 6 shows the results for each method across multiple model architectures and datasets. For VGG-16 (CIFAR-10), for all levels of sparsity, Reconvene is on par with UPaI for accuracy. The SPaI, inverted, and random methods, however, see a decline in accuracy beyond trivial sparsity levels of 80%. At 98% sparsity, the accuracy of the random method is 2% less than both Reconvene and UPaI, while SPaI and inverted lag behind by 15% and 4%, respectively. This data supports the observation that Reconvene is selectively pruning the right layers to maintain accuracy. In contrast, the inverted method, which prunes the opposite set of layers, reduces accuracy. Additionally, employing unmodified SPaI to prune all layers results in the highest decline in model accuracy.

In terms of model compression, UPaI remains at $1\times$ for all sparsity levels since zero parameters are the same size in storage as non-zero parameters. SPaI obtains the highest compression, but the model becomes impractical to use at extreme sparsity levels because of its diminished accuracy. Reconvene achieves the second highest compression, primarily because resilient layers are typically found deeper in the DNN, and these layers are usually larger than the initial ones. On the other hand, the inverted method targets the more sensitive layers, which are generally smaller and positioned at the beginning of the DNN. Consequently, this results in a compression rate lower than Reconvene.

For CPU/GPU speedup, a similar trend is observed for SPaI, achieving the highest speedup but at the cost of model usability. Since the early layers, which the inverted method

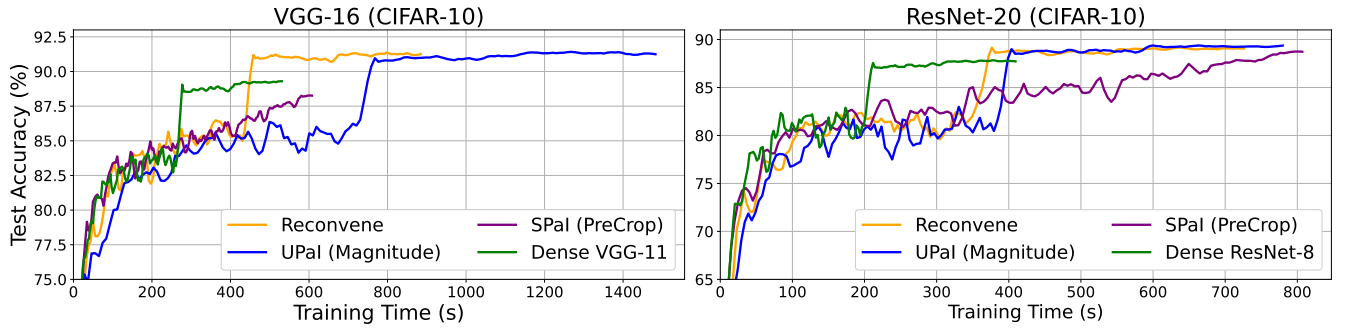


Fig. 7: Comparison of test accuracy over training time for VGG-16 (CIFAR-10, $p = 0.98$) and ResNet-20 (CIFAR-10, $p = 0.8$) for various UPaI and SPaI pruning methods and alternative smaller models (VGG-11 and ResNet-8).

prunes, are typically slower than later layers, the inverted method obtains a higher speedup than Reconvene. However, this is only observed in the CPU inference tests. The GPU inference test shows that Reconvene and the inverted method obtain nearly the same speedup at most sparsity levels. For UPaI, without specialized hardware accelerators, sparse models cannot be effectively utilized. When accelerators are not available, a sparse model can sometimes be slower than its dense counterpart (up to 2% slower; Figure 6).

The same trend is observed for both ResNet models on CIFAR-10 and Tiny ImageNet. Reconvene maintains the same accuracy as UPaI for ResNet-20 with a small divergence at extreme sparsities for ResNet-50. In addition, Reconvene reaches $8\times$ and $18.2\times$ compression for ResNet-20 and ResNet-50 at 98% sparsity with $\sim 2\times$ CPU speedup on both models and $1.34\times$ and $2.63\times$ GPU speedup respectively.

Observation 1: Without sacrificing the model accuracy achieved by UPaI, Reconvene obtains up to $16.21\times$ compression, $2\times$ and $1.78\times$ CPU and GPU speedup, respectively.

C. Improvements during training

In this experiment, Reconvene is compared against a SPaI method, namely PreCrop [16], UPaI, and training a smaller model of the same architecture from scratch. The aim is to demonstrate that: (1) Reconvene trains a model to full accuracy faster than UPaI, (2) Reconvene trains to a higher accuracy than SPaI, and (3) Reconvene creates pruned models which train faster and to a higher accuracy than manually choosing a smaller model and training it from scratch.

Figure 7 shows the results for VGG-16 and ResNet-20. For VGG-16, UPaI takes 1,482 seconds to train to an accuracy of 91.24%. Reconvene takes 884 seconds, $1.68\times$ faster than UPaI, to train to an accuracy of 91.26%. PreCrop, which applies SPaI to all layers, trains to an accuracy of 88.26% in 607 seconds. Within the same timeframe, Reconvene has already trained to an accuracy of 90.6%. Furthermore, PreCrop trains more slowly and achieves a lower accuracy compared to VGG-11 at 89.3% after 531 seconds. In this scenario, choosing VGG-11 for deployment instead of pruning VGG-16 with PreCrop would yield a higher-quality model in a shorter time. Conversely, Reconvene achieved 91.1% accuracy in just 440 seconds of training, outperforming both VGG-11 and PreCrop.

Similarly, for ResNet-20, UPaI takes 780 seconds to train to an accuracy of 89.37%. Reconvene takes 725 seconds, $1.08\times$ faster than UPaI, to train to an accuracy of 89.2%. Notably, PreCrop takes longer than UPaI to train a ResNet-20 model at 807 seconds to a lower accuracy of 88.73%. Compared to a smaller model, ResNet-8, Reconvene reaches peak accuracy $1.09\times$ faster. Meanwhile, ResNet-8 outperforms PreCrop in both accuracy and training time.

Observation 2: Reconvene achieves a higher accuracy faster than other SPaI/UPaI methods and smaller models. In fact, Reconvene trains a pruned VGG-16 to within 0.1% of UPaI $3.37\times$ faster.

D. Comparison against other SPaI methods

In this experiment, Reconvene, three alternative SPaI methods, and random pruning are evaluated based on accuracy change, GPU speedup, and compression to compare pruned model quality across a range of sparsity values. VGG-19 (CIFAR-10) is the baseline dense model. The three alternative SPaI methods are PreCrop [16], ProsPr [21], and 3SP [43]. Table III presents the results for all methods².

At 80% sparsity, Reconvene creates the fastest and most compressed pruned model. However, ProsPr maintains 0.09% more accuracy. At 90% sparsity, PreCrop obtains a higher compression ratio and speedup, with a 0.14% lower accuracy than Reconvene. The same trend is observed at 95% sparsity. However, the accuracy gap between PreCrop and Reconvene is now 0.41%. 3SP has the lowest accuracy outside of random pruning for all sparsities except for 95%, where PreCrop is lower. ProsPr achieves the smallest speedup because it mainly prunes the fully connected layers of VGG-19 [21]. Pruning these layers does not decrease inference latency as much as pruning the convolutional layers. At higher sparsity levels, PreCrop outperforms all other methods in terms of speedup and compression because it prunes every layer in the DNN [16]. However, this comes at the expense of reduced model accuracy than other methods.

Observation 3: Reconvene effectively balances the quality of pruned models, making it suitable for edge deployments.

²Some results are missing since the source code for structured pruning used by 3SP and ProsPr are not publicly available, and we could not fully replicate their results. Nonetheless, these results are comparable to Reconvene.

TABLE III: Top-1 accuracy change from the dense baseline model, GPU speedup, and compression for SPaI methods using VGG-19 (CIFAR-10).

Sparsity	Method	Acc. Δ	Speedup (\times)	Comp. (\times)
80%	Random	-1.60	-	-
	3SP [43]	-0.20	-	-
	ProsPr [21]	+0.01	1.10	-
	PreCrop [16]	-0.07	1.35	4.55
	Reconvene	-0.08	1.36	4.66
90%	Random	-3.20	-	-
	3SP [43]	-0.50	-	-
	ProsPr [21]	0.00	1.26	-
	PreCrop [16]	-0.26	1.63	8.89
	Reconvene	-0.12	1.43	5.33
95%	Random	-4.60	-	-
	3SP [43]	-1.10	-	-
	ProsPr [21]	-0.28	1.30	-
	PreCrop [16]	-1.22	1.79	17.59
	Reconvene	-0.81	1.44	5.46

It maintains high accuracy while also speeding up and compressing the model significantly.

E. Overheads

In this experiment, Reconvene is assessed as a low-overhead NAS method for producing pruned models. It is contrasted with other methods that discover their compressed models from an expansive search space. Moreover, Reconvene is compared against traditional pruning after training (PaT). Table IV presents the results for Reconvene against four NAS methods and l^1 norm pruning after training (PaT). Search time is defined as the time to find the model candidate. The pruning methods achieve this in one shot by pruning the dense model into a smaller pruned model. NAS methods create many hundreds to thousands of candidate compressed models and then evaluate each for optimality. Reconvene is $2.19\times$ faster than l^1 norm since Reconvene prunes based on layer metrics whereas l^1 norm prunes based on the l^1 norm of each convolutional filter. In addition, Reconvene produces a pruned model which trains to a higher accuracy than l^1 norm PaT. Compared to the NAS methods, Reconvene is $1,000\times$ to $15,000\times$ faster than NAS at discovering a pruned model and uses up to $38\times$ less system memory. In addition, the model accuracy is higher than all other NAS methods except for ZenNAS [44]³.

Observation 4: Reconvene serves as a search method to identify new pruned models suitable for edge devices, which traditionally relied on their larger cloud counterparts. Reconvene operates significantly faster and yields models of similar accuracy. Its reduced memory requirements further enable its execution on resource-constrained edge devices.

V. RELATED WORK

Other Compression Methods such as quantization reduces the bit precision of DNN parameters to shrink the model and increase inference speed [8]. However, it often leads to accuracy loss and might require specialized hardware for lower-precision inference. Knowledge distillation trains a smaller

³ZenNAS has a post-search training regime $10\times$ longer than typical, which results in a much higher final accuracy than all other methods.

TABLE IV: Search time and memory usage for various search methods to create a compressed model trained on CIFAR-10. Neural architecture search (NAS) methods target 0.5M parameters. Pruning after training (PaT) and pruning at initialization (PaI) methods prune VGG-16 to a sparsity of 95% ($p = 0.95$, $\sim 0.5M$ parameters). The top-1 accuracy change is based on a reference VGG-16 (CIFAR-10).

Method	Type	Search Time (s)	Memory Usage (GB)	Acc. Δ
ZenNAS [44]	NAS	19,944	4.8	+2.70
DARTSv2 [45]	NAS	1,957	8.61	-1.30
NASNet [46]	NAS	1,468	11.73	-1.32
NASWOT [47]	NAS	306	-	-0.88
l^1 norm [48]	PaT	3	5.12	-2.39
Reconvene	PaI	1.37	0.31	-0.81

student model using training knowledge from a larger teacher model [9]. Although the student can often match the accuracy of the teacher model and take up less space, it is not easily adaptable to different model architectures. Consequently, it does not scale for the heterogeneous edge settings.

Pruning Systems are employed to compress DNN models, creating a range of compact models suited for edge devices. These pruned models optimize resource constraints while maintaining performance on edge deployments. Existing work focuses on pruning after model training [12], [29], or target model architectures for hardware accelerators [23].

Neural Architecture Search (NAS) automates the process of finding optimal DNN model architectures. It is conventionally used to discover larger models that train to higher accuracies [46]. However, NAS has also been employed to discover smaller models optimized for edge devices [10]. While NAS is effective at discovering high-quality models, repeating the NAS pipeline for a new dataset can be both time-consuming and resource-intensive [10].

Model Reparameterization aims to optimize model structures, enhancing hardware utilization and thereby improving inference efficiency. For example, RepVGG [32] reparameterizes ResNet architectures into VGG-style models. However, these methods only work on specific pairs of model architectures and require hardware accelerators, such as GPUs, to maximize utilization.

VI. CONCLUSIONS

Pruning at initialization (PaI) allows for compressed models to be discovered rapidly. However, existing structured PaI methods sacrifice model accuracy to achieve the necessary performance increase required for resource-constrained edge computing. Reconvene addresses this concern by closing the performance gap between unstructured and structured PaI while maintaining the accuracy of unstructured PaI through selective structured pruning of non-sensitive model layers. Reconvene has been shown to work across a range of model architectures and datasets and serves as a foundation for future structured PaI methods.

ACKNOWLEDGMENTS

This research is funded by Rakuten Mobile, Inc., Japan.

REFERENCES

- [1] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," in *IEEE International Conference on Smart Cloud*, 2016, pp. 20–26.
- [2] M. Z. Khan, S. Harous, S. U. Hassan, M. U. Ghani Khan, R. Iqbal, and S. Mumtaz, "Deep Unified Model For Face Recognition Based on Convolution Neural Network and Edge computing," *IEEE Access*, 2019.
- [3] X. Jin, L. Li, F. Dang, X. Chen, and Y. Liu, "A Survey on Edge Computing for Wearable Technology," *Digital Signal Processing*, 2022.
- [4] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," in *International Conference on Neural Information Processing Systems*, 2015, p. 1135–1143.
- [5] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Gutttag, "What is the State of Neural Network Pruning?" *Machine Learning and Systems*, pp. 129–146, 2020.
- [6] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, 2020.
- [7] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, pp. 485–532, 2020.
- [8] U. Kulkarni, A. S. Hosamani, A. S. Masur, S. Hegde, G. R. Vernekar, and K. Siri Chandana, "A Survey on Quantization Methods for Optimization of Deep Neural Networks," in *International Conference on Automation, Computing and Renewable Systems*, 2022, pp. 827–834.
- [9] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the Knowledge in a Neural Network," *arXiv:1503.02531*, 2015.
- [10] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for All: Train One Network and Specialize it for Efficient Deployment," in *International Conference on Learning Representations*, 2020.
- [11] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *IEEE International Conference on Computer Vision*, 2017, pp. 1398–1406.
- [12] B. J. Eccles, P. Rodgers, P. Kilpatrick, I. Spence, and B. Varghese, "DNNShifter: An Efficient DNN Pruning System for Edge Computing," *Future Generation Computer Systems*, 2023.
- [13] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model Pruning Enables Efficient Federated Learning on Edge Devices," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
- [14] X. Qiu, J. Fernandez-Marques, P. P. Gusmao, Y. Gao, T. Parcollet, and N. D. Lane, "ZeroFL: Efficient On-Device Training for Federated Learning with Local Sparsity," in *International Conference on Learning Representations*, 2022.
- [15] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, "Pruning Neural Networks at Initialization: Why Are We Missing the Mark?" in *International Conference on Learning Representations*, 2021.
- [16] Y. Cai, W. Hua, H. Chen, G. E. Suh, C. D. Sa, and Z. Zhang, "Structured Pruning is All You Need for Pruning CNNs at Initialization," *arXiv:2203.02549*, 2022.
- [17] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Inference," in *International Conference on Learning Representations*, 2017.
- [18] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, 1989.
- [19] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," in *International Conference on Learning Representations*, 2019.
- [20] H. Tanaka, D. Kumin, D. L. Yamins, and S. Ganguli, "Pruning Neural Networks Without Any Data by Iteratively Conserving Synaptic Flow," *Advances in Neural Information Processing Systems*, 2020.
- [21] M. Alizadeh, S. A. Taylor, L. M. Zintgraf, J. van Amersfoort, S. Farquhar, N. D. Lane, and Y. Gal, "Prospect Pruning: Finding Trainable Weights at Initialization using Meta-Gradients," in *International Conference on Learning Representations*, 2022.
- [22] T. Gale, E. Elsen, and S. Hooker, "The State of Sparsity in Deep Neural Networks," *arXiv:1902.09574*, 2019.
- [23] 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," *SIGARCH Computer Architecture News*, p. 243–254, 2016.
- [24] S. Muralidharan, "Uniform Sparsity in Deep Neural Networks," in *Machine Learning and Systems*, 2023.
- [25] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A Survey on the Edge Computing for the Internet of Things," *IEEE Access*, pp. 6900–6919, 2018.
- [26] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian, X. Lin, K. Ma, and Y. Wang, "Non-Structured DNN Weight Pruning—Is It Beneficial in Any Platform?" *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021.
- [27] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning Structured Sparsity in Deep Neural Networks," in *Advances in Neural Information Processing Systems*, 2016.
- [28] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," in *International Conference on Learning Representations*, 2017.
- [29] F. Yu, L. Cui, P. Wang, C. Han, R. Huang, and X. Huang, "EasiEdge: A Novel Global Deep Neural Networks Pruning Method for Efficient Edge Computing," *IEEE Internet of Things Journal*, pp. 1259–1271, 2021.
- [30] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the Value of Network Pruning," in *International Conference on Learning Representations*, 2019.
- [31] C. Wang, G. Zhang, and R. Grosse, "Picking Winning Tickets Before Training by Preserving Gradient Flow," in *International Conference on Learning Representations*, 2020.
- [32] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "RepVGG: Making VGG-style ConvNets Great Again," in *Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 733–13 742.
- [33] J. Turner, J. Cano, V. Radu, E. J. Crowley, M. O'Boyle, and A. Storkey, "Characterising Across-Stack Optimisations for Deep Convolutional Neural Networks," in *IEEE International Symposium on Workload Characterization*, 2018, pp. 101–110.
- [34] S. Yu, P. Nguyen, A. Anwar, and A. Jannesari, "Heterogeneous Federated Learning using Dynamic Model Pruning and Adaptive Gradient," in *IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing*, 2023.
- [35] S. Vahidian, M. Morafah, and B. Lin, "Personalized Federated Learning by Structured and Unstructured Pruning under Data Heterogeneity," in *International Conference on Distributed Computing Systems Workshops*, 2021, pp. 27–34.
- [36] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/cifar.html>
- [37] Y. Le and X. Yang, "Tiny Imagenet Visual Recognition Challenge," 2015. [Online]. Available: <http://vision.stanford.edu/teaching/cs231n/>
- [38] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015, pp. 1–14.
- [39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [40] H. Wang, C. Qin, Y. Bai, and Y. Fu, "Why is the State of Neural Network Pruning so Confusing? On the Fairness, Comparison Setup, and Trainability in Network Pruning," *arXiv:2301.05219*, 2023.
- [41] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, "DepGraph: Towards Any Structural Pruning," in *IEEE/Conference on Computer Vision and Pattern Recognition*, 2023.
- [42] J. Turner, J. Cano, V. Radu, E. J. Crowley, M. O'Boyle, and A. Storkey, "Characterising Across-Stack Optimisations for Deep Convolutional Neural Networks," in *IEEE International Symposium on Workload Characterization*, 2018, pp. 101–110.
- [43] J. van Amersfoort, M. Alizadeh, S. Farquhar, N. Lane, and Y. Gal, "Single Shot Structured Pruning Before Training," *arXiv:2007.00389*, 2020.
- [44] M. Lin, P. Wang, Z. Sun, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin, "Zen-NAS: A Zero-Shot NAS for High-Performance Deep Image Recognition," in *IEEE/CVF International Conference on Computer Vision*, 2021.
- [45] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *International Conference on Learning Representations*, 2019.
- [46] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.
- [47] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural Architecture Search without Training," in *International Conference on Machine Learning*, 2021.
- [48] T. Pragnesh and B. R. Mohan, "Compression of Convolution Neural Network Using Structured Pruning," in *IEEE International Conference for Convergence in Technology*, 2022, pp. 1–5.