DASS: Differentiable Architecture Search for Sparse Neural Networks

HAMID MOUSAVI and MOHAMMAD LONI, School of Innovation, Design and Engineering, Mälardalen University, Sweden
MINA ALIBEIGI, Zenseact AB, Lindholmspiren 2, Sweden
MASOUD DANESHTALAB, School of Innovation, Design and Engineering, Mälardalen University, Sweden and Computer systems, Tallinn University of Technology, Estonia

The deployment of Deep Neural Networks (DNNs) on edge devices is hindered by the substantial gap between performance requirements and available computational power. While recent research has made significant strides in developing pruning methods to build a sparse network for reducing the computing overhead of DNNs, there remains considerable accuracy loss, especially at high pruning ratios. We find that the architectures designed for dense networks by differentiable architecture search methods are ineffective when pruning mechanisms are applied to them. The main reason is that the current methods do not support sparse architectures in their search space and use a search objective that is made for dense networks and does not focus on sparsity.

This paper proposes a new method to search for sparsity-friendly neural architectures. It is done by adding two new sparse operations to the search space and modifying the search objective. We propose two novel parametric `SparseConv` and `SparseLinear` operations in order to expand the search space to include sparse operations. In particular, these operations make a flexible search space due to using sparse parametric versions of linear and convolution operations. The proposed search objective lets us train the architecture based on the sparsity of the search space operations. Quantitative analyses demonstrate that architectures found through DASS outperform those used in the state-of-the-art sparse networks on the CIFAR-10 and ImageNet datasets. In terms of performance and hardware effectiveness, DASS increases the accuracy of the sparse version of MobileNet-v2 from 73.44% to 81.35% (+7.91% improvement) with a 3.87× faster inference time.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Continuous space search**; **Supervised learning by classification;**

Additional Key Words and Phrases: Neural architecture search, network sparsification, optimization, image classification

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), 2023.
This work was supported in part by the European Union through European Social Fund in the frames of the "Information and Communication Technologies (ICT) program" and by the Swedish Innovation Agency VINNOVA project "AutoDeep", "SafeDeep", and "KKS DPAC".
Authors' addresses: H. Mousavi and M. Loni, School of Innovation, Design and Engineering, Mälardalen University, PO Box 102, Västerås, Sweden, 72126; emails: seyedhamidreza.mousavi@mdu.se; mohammad.loni@mdu.se; M. Alibeigi, Zenseact AB, Lindholmspiren 2, Göteborg, Sweden; email: mina.alibeigi@zenseact.com; M. Daneshtalab, School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden and Computer systems, Tallinn University of Technology, Tallinn, Estonia; email: masoud.daneshtalab@mdu.se.

1539-9087/2023/09-ART105 $15.00
https://doi.org/10.1145/3609385

ACM Reference format:
Hamid Mousavi, Mohammad Loni, Mina Alibeigi, and Masoud Daneshtalab. 2023. DASS: Differentiable Architecture Search for Sparse Neural Networks. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 105 (September 2023), 21 pages.
https://doi.org/10.1145/3609385

1 INTRODUCTION

Deep Neural Networks (DNNs) provide an excellent avenue for obtaining the maximum feature extraction capacities required to resolve highly complex computer vision tasks [1–4]. There is an increasing demand for DNNs to become more efficient in order to be deployed on extremely resource-constrained edge devices. However, DNNs are not intrinsically designed for the limited computing and memory capacities of tiny edge devices, prohibiting their deployment in such devices [5–9].

To democratize DNN acceleration, a variety of optimization approaches have been proposed, including network pruning [10–13], efficient architecture design [6, 8], network quantization [14–16], knowledge distillation [17, 18], and low-rank decomposition [19]. In particular, network pruning is known to provide remarkable computational and memory savings by removing redundant weight parameters in the unstructured scenario [10–12, 20, 21], and the entire filter in the structured scenario [22–26]. Recently, unstructured pruning methods have been reported to provide extreme reductions in network size. The state-of-the-art unstructured pruning methods [10] provide up to 99% pruning ratio, which is an excellent scenario for tiny edge devices.

However, these methods suffer from a substantial accuracy drop, preventing them from being applied in practice (≈19% accuracy drop for MobileNet-v2 compared to dense one [10]). Current pruning methods use handcrafted architectures designed without concern about sparsity. [10–12, 20, 25]. We hypothesize that the backbone architecture may not be optimal for scenarios with extreme pruning ratios. Instead, we can learn more efficient backbone architectures that are adaptable to pruning techniques by exploring the space of sparse networks.

Neural Architecture Search (NAS) has achieved great success in the automated design of high-performance DNN architectures. Differentiable architecture search (DARTS) methods [27–29] are popular NAS methods that use a gradient-based search algorithm to accelerate search speed. Motivated by the promising results of NAS, we came up with the idea of designing custom backbone architectures compatible with pruning methods. Nevertheless, the search space of current DARTS algorithms comprises dense convolution and linear operations that are incapable of exploring the correct backbone for pruning. To demonstrate this issue, we first prune 99% of the weights from the best architecture designed by the NAS method [27] with base search space without considering sparsity into account. Disappointingly, after applying the pruning method to the final architecture, it performs poorly with up to ≈21% accuracy loss in compression to DASS (our proposed method) that extends the search space by sparse operations. (Section 4). This failure is due to a lack of support for specific sparse network characteristics, leading to low generalization performance. Based on the above hypothesis and empirical observations, we formulate a search space that includes sparse and dense operations. Therefore, the original convolution and linear operations in the search space of the NAS have been extended by parametric `SparseConv` and `SparseLinear` operations, respectively. Moreover, to make a consistency between the proposed search space and search objective function, we modify the bi-level optimization problem to take sparsity into account. In this way, the search process tries to find the best sparse operation by optimizing both architecture and pruning parameters. This modification creates a complex bi-level optimization problem. To tackle this difficulty, we split the complex bi-level optimization into two simple bi-level optimization problems and solve them.

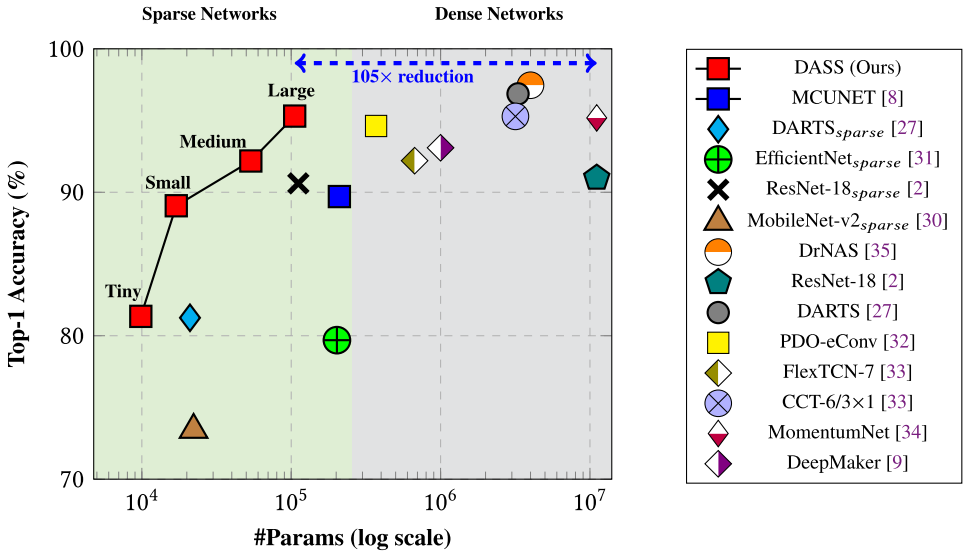

Fig. 1. Top-1 accuracy (%) vs. number of network parameters (#Params) trained on CIFAR-10 for various sparse and dense architectures.

We show that explicitly integrating pruning into the search procedure can lead to finding sparse network architectures with significant accuracy improvement. In Figure 1, we compare the CIFAR-10 Top-1 accuracy and the number of parameters of the found architecture by DASS with the state-of-the-art sparse (unstructured pruning) and dense networks. The results show that the architecture designed by DASS outperforms all competing architectures that employ the pruning method. DASS-Small demonstrates its consistent effectiveness by achieving 15%, 10%, and 8% accuracy improvement over MobileNet-v2$_{sparse}$ [30], EfficientNet-v2$_{sparse}$ [31], and DARTS$_{sparse}$ [27], respectively. In addition, compared to networks with similar accuracy, DASS-Large has a significant reduction in network complexity (#Params) by 3.5×, 30.0×, 105.2× over PDO-eConv [32], CCT-6/3×1 [33], and MomentumNet [34], respectively. Section 6 provides a comprehensive experimental study to evaluate different aspects of DASS. Our main contributions are summarized as follows:

(1) We perform extensive experiments to identify the limitations of applying pruning methods with extreme pruning ratios to dense architecture as a post-processing step.
(2) We define a new search space by extending the base search space in DARTS with a new set of parametric operations (`SparseConv` and `SparseLinear`) to consider the sparse operations in the search space.
(3) We modify the bi-level optimization problem to be consistent with the new search space and propose a three-step gradient-based algorithm to split the complex bi-level problem and learn architecture parameters, network weights, and pruning parameters.

2 RELATED WORK

2.1 Neural Architecture Search and DARTS Variants

Neural Architecture Search (NAS) has recently attracted remarkable attention by relieving human experts from the laborious effort of designing neural networks. Early NAS methods mainly utilized evolutionary-based [9, 36–38] or reinforcement-learning-based methods [39–41]. Despite the

efficiencies of the architecture designed by evolutionary-based and reinforcement-learning-based methods, they require tremendous computing resources. For example, the proposed method in [39] evaluates 20,000 neural candidates in 500 NVIDIA® P100 GPUs over four days. One-shot architecture search methods [42–44] have been proposed to identify optimal neural architectures in a few GPU days (>1 GPU day [45]). In particular, Differentiable Architecture Search (DARTS) [27–29] is a variation of one-shot NAS methods that relaxes the search space to be continuous and differentiable. The detailed description of DARTS can be found in Section 3.1. Despite the broad successes of DARTS in advancing NAS applicability, achieving optimal results remains a challenge for real-world problems. Many subsequent works investigate some of these challenges by focusing on (i) increasing search speed [46, 47], (ii) improving generalization performance [35, 48], (iii) addressing robustness issues [49–51], (iv) reducing quantization error [14, 16], and (v) designing hardware-aware architectures [52–54]. On the other hand, few works attempt to prune the search space by removing inferior network operations. [55–60]. These works utilized the pruning mechanism to progressively remove some operations from the search space. Unlike them, our method aims to extend the search space to improve the performance of the sparse network by searching for the best operations with sparse weight structures. Technically, our method extends the search space by adding the parametric sparse version of convolution and linear operations to find the best sparse architecture. Therefore, there is a lack of research on sparse weight parameters when designing neural architectures. Our proposed method (DASS) searches for the operations that are most effective for sparse weight parameters in order to achieve higher generalizing performance.

2.2 Network Pruning

Network pruning is an effective method for reducing the size of DNNs, enabling them to be effectively deployed on devices with limited resource capacity. Prior works on network pruning can be classified into two categories: structured and unstructured pruning methods. The purpose of structured pruning is to remove redundant channels or filters to preserve the entire structure of weight tensors with dimension reduction [22–25, 61, 62]. While structured pruning is famous for hardware acceleration, it sacrifices a certain degree of flexibility as well as weight sparsity [63].

On the other hand, unstructured pruning methods offer superior flexibility and compression rate by removing parameters with the least impact on the network accuracy of the weight tensors [10–12, 20, 22, 63–66]. In general, unstructured pruning entails three stages to make a sparse network, including (i) pre-training, (ii) pruning, and (iii) fine-tuning. Prior unstructured pruning methods used various criteria to select the lowest pruning weight parameters. [67, 68] pruned weight parameters based on the second-derivative values of the loss function. Several studies proposed to remove the weight parameters below a fixed pruning threshold, regardless of the training objective [64–66, 69–71]. To address the limitation of fixed thresholding methods, [20, 72] proposed layer-wise trainable thresholds to determine the optimal value for each layer separately. The lottery-ticket hypothesis [66, 73, 74] is a different line of the method that identifies the pruning mask for an initialized CNN and trains the resulting sparse model from scratch without changing the pruning mask. HYDRA [10] formulate the pruning objective as empirical risk minimization and integrate it with the training objective. Unlike other methods, optimization-based pruning criteria improve the performance of sparse networks in comparison to other metrics. Despite the success of optimization-based pruning in achieving a significant compression rate, the classification accuracy is compromised, notably when the pruning ratio is extremely high (up to 99%). We show that the main reason for this issue is due to the non-optimal backbone architecture. We extend the search space of DASS by parametric sparse operations and formulate pruning as an empirical risk minimization problem and integrate it into the bi-level optimization problem to find the best sparse network.

3 PRELIMINARIES

3.1 Differentiable Architecture Search

Differentiable Architecture Search (DARTS) [27] is a NAS method that significantly reduces the search cost by relaxing the search space to be continuous and differentiable. DARTS cell template is represented by a Directed Acyclic Graph (DAG) containing N intra-nodes. The edge (i, j) between two nodes is associated with an operation $o^{(i,j)}$ (e.g., skip connection or 3×3 max-pooling) within O search space. Equation (1) computes the output of intermediate nodes.

$$\bar{o}^{(i,j)}(x^{(i)}) = \sum_{o\in O} \frac{exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in O} exp\left(\alpha_{o'}^{(i,j)}\right)} \cdot o(x^{(i)}) \tag{1}$$

where O and $\alpha_o^{(i,j)}$ denote the set of all candidate operations and the selection probability of o, respectively. The output node in the cell is the concatenation of all intermediate nodes. DARTS optimizes architecture parameters (α) and network weights (θ) with the following bi-level objective function:

$$\min_{\alpha} \mathcal{L}_{val}(\theta^{\star}, \alpha) \;\; s.t. \; \theta^{\star} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}_{train}(\theta, \alpha) \tag{2}$$

where

$$\mathcal{L}_{train} = \frac{\sum_{(\boldsymbol{x},y)\in(X_{train},Y_{train})} l(\theta, \boldsymbol{x}, y)}{|X_{train}|}$$

and

$$\mathcal{L}_{val} = \frac{\sum_{(\boldsymbol{x},y)\in(X_{val},Y_{val})} l(\theta, \boldsymbol{x}, y)}{|X_{val}|}$$

The operation with the largest α_o is selected for each edge. X_{train} and Y_{train} represent the training dataset and corresponding labels, respectively. Similarly, the validation dataset and labels are indicated by X_{val} and Y_{val}, respectively. After the search process has been completed, the final architecture is re-trained from scratch to obtain maximum accuracy.

3.2 Unstructured Pruning

Pruning is considered unstructured if it removes low-importance parameters from weight tensors and makes sparse [63]. This paper uses the optimization-based unstructured network pruning method to provide greater flexibility and an extreme compression rate compared to structured pruning methods. The pruning method includes three main optimization stages: (i) pre-training: training the network on the target dataset, (ii) pruning: pruning unimportant weights from the pre-trained network, and (iii) fine-tuning: the sparse network is re-trained to recover its original accuracy. For the pruning stage, we consider an optimization-based method with the following steps: First, we define the pruning parameters that show the importance of each weight of the network (s^0) and initialize them according to Equation (3).

$$s_i^0 \propto \frac{1}{\max(|\theta_{pre,i}|)} \times \theta_{pre,i} \tag{3}$$

where $\theta_{pre,i}$ denotes the weight of i_{th} layer in the pre-trained network. Next, to learn the pruning parameters ($\hat{s}$), we formulate the optimization problem as Equation (4), which is then solved by stochastic gradient descent (SGD) [75].

$$\hat{s} = \underset{s}{\operatorname{argmin}} \mathbb{E}_{(x,y)\sim D}\left[\mathcal{L}_{prune}(\theta_{pre}, s, x, y)\right] \tag{4}$$

θ_{pre} and $\mathbb{E}$ refer to the pre-trained network parameters and mathematical expectation, respectively. By solving this optimization problem, we are able to determine the effect of each weight

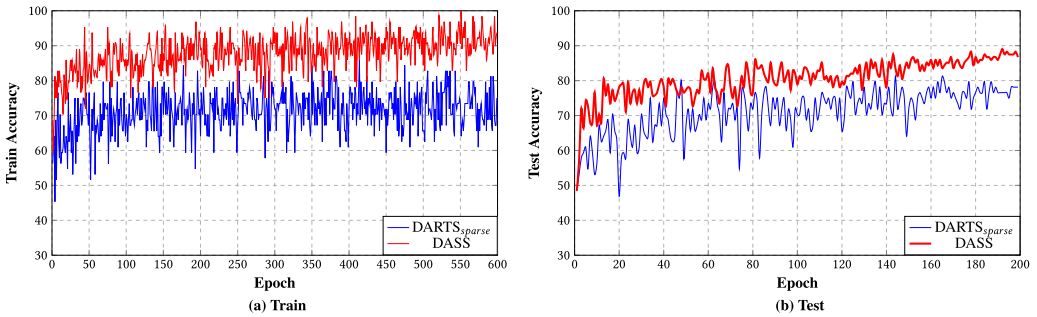

Fig. 2. Comparison of DASS-Small and DARTS$_{sparse}$ on CIFAR-10 for (a) train and (b) test learning curves.

parameter on the loss function and, consequently, the accuracy of the network. Finally, we convert the floating values of the pruning parameters to a binary mask based on selecting top-k weights with the highest magnitude of pruning parameters.

4 RESEARCH MOTIVATION

The dense network architectures that were originally designed using conventional NAS methods are inaccurate when integrated with pruning methods, particularly at high pruning ratios. To demonstrate this assertion, we first apply the unstructured pruning method explained in Section 3.2 to the best architecture designed by DARTS [27] for CIFAR-10 and generate a sparse network. We call this solution DARTS$_{sparse}$. Then, we compare the performance of the sparse architecture designed by DASS with DARTS$_{sparse}$. Figure 2 illustrates the train and test accuracy curves for the DASS and DARTS$_{sparse}$ architectures trained on the CIFAR-10 dataset. Disappointingly, the network designed by DARTS$_{sparse}$ results in reduced test accuracy. This implies that the dense backbone architectures designed by NAS methods without considering sparsity are ineffective (DASS delivers 8% higher test accuracy compared to DARTS$_{sparse}$). According to our investigations, we find two issues involved in the training failure of DARTS$_{sparse}$: (i) DARTS does not support sparse operations in its search space, and (ii) DARTS optimizes the search objective without considering sparsity into account. Section 5.2 addresses the first issue, while the second issue is addressed in Section 5.3.

We investigate DASS in two modes to demonstrate the significance of including sparse operations and reformulating the objective function based on sparsity. The first mode extends the search space with sparse operations solely (DASS$_{Op}$) and does not optimize the pruning parameters, while the second mode adds sparsity to the optimization process and optimizes the architecture and pruning parameters in a bi-level optimization problem. (DASS$_{Op+Ob}$). Figure 3 indicates the test accuracy for DASS$_{Op}$, DARTS$_{sparse}$ and (DASS$_{Op+Ob}$) architectures with various pruning ratios. As results show, DASS$_{Op}$ has ≈3.4% lower accuracy compared to DASS$_{Op+Ob}$ and ≈4.47% higher accuracy compared to DARTS$_{sparse}$. In conclusion, extending the search space with proposed sparse operations (our first contribution) in DASS produces a better architecture than DARTS$_{sparse}$, but combining it with the sparsity-based optimization objective (our second contribution) enhances performance.

5 DIFFERENTIABLE ARCHITECTURE SEARCH FOR SPARSE NEURAL NETWORKS (DASS) METHOD

5.1 DASS: Overview

We propose DASS, a differentiable architecture search method for sparse neural networks. DASS at first extends the search space of the NAS with parametric sparse operations. Then it modifies the

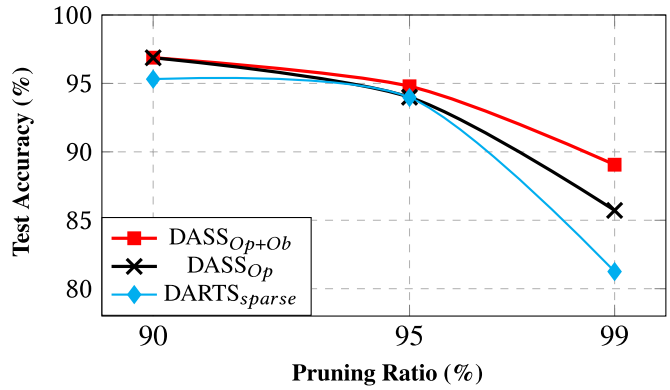

Fig. 3. DASS$_{Op+Ob}$ vs. DARTS$_{sparse}$ and DASS with only adding sparse operations to the search space (DASS$_{Op}$).

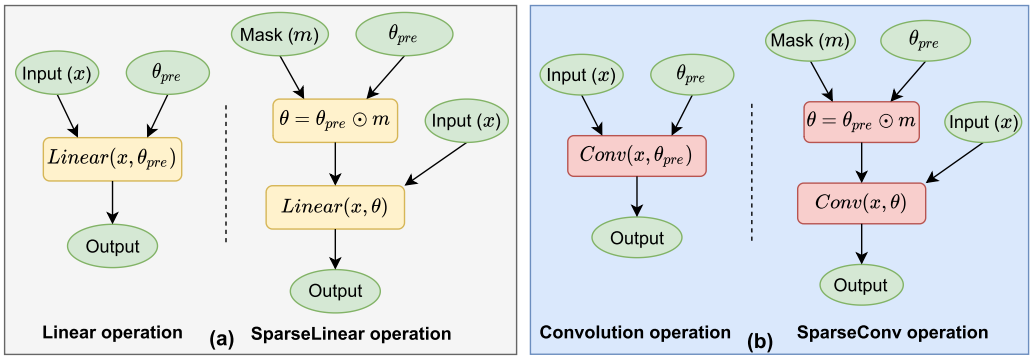

Fig. 4. Illustrating the (a) `SparseLinear` and (b) `SparseConv` operations.

bi-level optimization problem to learn the architecture, weights, and pruning parameters. DASS employs a three-step approach to solve the complicated bi-level optimization problem, which consists of (1) *Pre-training*: find the best dense architecture (pruning parameters equal to zero) from the search space and pre-train it (2) *Pruning and sparse architecture design*: find the best pruning mask (optimizing pruning parameters) and update the architecture parameters based on the sparse weights and finally (3) *Fine-tuning*: re-train the sparse architecture to achieve the maximum classification performance.

5.2 DASS Search Space

To support sparse operations, DASS proposes the parametric sparse version of convolution and linear operations called `SparseConv` and `SparseLinear`, respectively. These operations have a sparsity mask (m) to remove redundant weight parameters from the network. Figure 4 illustrates the functionality of these two operations. In addition, Table 1 summarizes the operations of the DASS search space.

To empirically investigate the efficiency of the proposed sparse search space, we compare the similarity of the feature maps of high-performance dense architecture (with a large number of parameters) with the sparse architecture discovered by DASS and the architecture designed from the original search space; DARTS$_{spasre}$; methods. We use Kendall's τ [76] metric to measure the

Table 1. Operations of the DASS Search Space

Operation Type	Separable Sparse Convolution	Dilated sparse Convolution	Max Pooling	Average pooling	Skip connect
Kernel Size	$3 \times 3, 5 \times 5$	$3 \times 3, 5 \times 5$	3×3	3×3	N/A

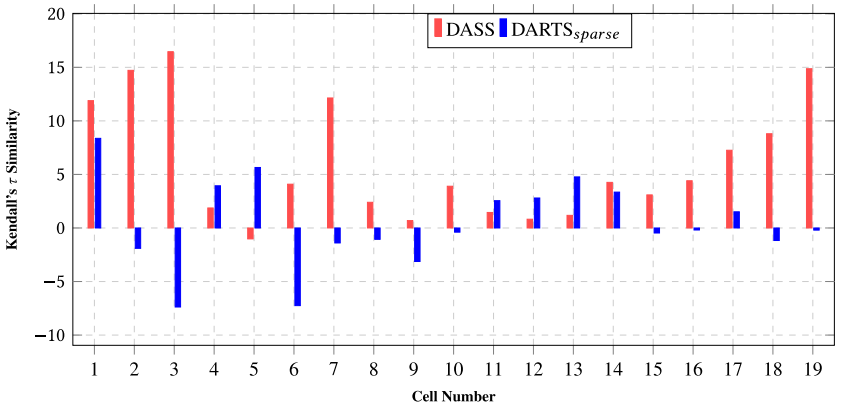

Fig. 5. Comparing the Kendall's τ similarity metric of architectures designed by both DARTS$_{sparse}$ and DASS methods with high-performance dense architecture.

similarity between output feature maps. The τ correlation coefficient returns a value between -1 and 1. To present the outcome more clearly, we scale up these values between -100 and 100. Closer values to 100 indicate stronger positive similarity between the feature maps. Figure 5 summarizes the results. Our observations reveal a similarity between DASS feature maps and dense architecture (up to 16%). On the other hand, the correlation between DARTS$_{sparse}$ and dense architecture is insignificant. Therefore, it shows that the architecture designed by DASS based on new search space can extract features more similar to high-performance dense architecture while DARTS$_{sparse}$ that use dense search space lost important features after pruning. The level of similarity is not very high because DASS is a sparse network with a pruning ratio of 99%. However, it can demonstrate that DASS retrieves useful features.

5.3 DASS Search Objective

DASS aims to search for optimal architecture parameters ($\alpha^\star$) to minimize the loss of validation of the sparse network. Thus, to achieve a consistent search objective with the proposed sparse search space, we formulate the entire search objective as a complex bi-level optimization problem:

$$\begin{aligned} &\alpha^\star = \min_{\alpha}(\mathcal{L}_{val}(\hat{\theta}(\alpha), \alpha)) \\ &s.t. \quad \begin{cases} \theta^\star(\alpha) = \text{argmin}_\theta \, \mathcal{L}_{train}(\theta, \alpha) \\ \hat{m} = \text{argmin}_{m \in \{0,1\}^N} \left[\mathcal{L}_{prune}(\theta^\star(\alpha) \odot m, \alpha)\right] \\ \hat{\theta}(\alpha) = \theta^\star(\alpha) \odot \hat{m}. \end{cases} \end{aligned} \tag{5}$$

Here m denotes the parameters of the binary pruning mask. This formulation learns the architecture parameters based on the sparse weight parameters. However, Equation (5) is not a straightforward bi-level optimization problem because the lower-level problem consists of two optimization problems. To overcome this challenge, we break the search objective into three distinct steps. Thus, the problem is transformed into two bi-level optimization problems to determine the optimal

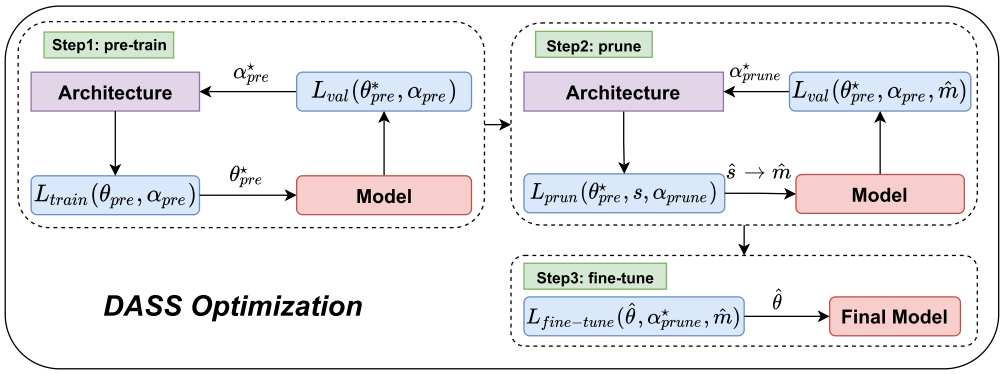

Fig. 6. The overview of the proposed optimization algorithm to find architecture parameters based on the sparse weight parameters. It consists of three main steps: (1) pre-training: search dense architecture (2) pruning: search sparse architecture (3) fine-tuning: re-train best sparse architecture.

architecture parameters for dense and sparse weights and an optimization problem to fine-tune the weight parameters. In addition, the lower-level optimization problem consists of a discrete optimization problem for pruning masks.

Section 5.4 proposes a multi-step optimization algorithm to solve the optimization problem and handle the discrete optimization problem by converting it to a continuous optimization problem.

5.4 Optimization Algorithm

As shown in Figure 6, the optimization algorithm consists of three main steps, which are described as follows.

Step 1: pre-train (learn $\theta^{\star}_{pre}$ and α^{*}_{pre}). In this step, we break Equation (5) into a bi-level optimization problem to find the best dense architecture. This pre-training is necessary for the next step which learns pruning mask parameters and modifying the sparse architecture.

$$\begin{aligned} \alpha^{\star}_{pre} &= \min_{\alpha_{pre}} (\mathcal{L}_{val}(\theta^{*}_{pre}(\alpha_{pre}), \alpha_{pre})) \\ s.t. \quad \theta^{\star}_{pre}(\alpha_{pre}) &= \operatorname*{argmin}_{\theta_{pre}} \mathcal{L}_{train}(\theta_{pre}, \alpha_{pre}) \end{aligned} \tag{6}$$

The first-order approximation technique use to update $\theta^{\star}_{pre}$ and α_{pre} alternately using gradient descent [27].

Step 2: prune (learn $\hat{m}$ and α^{*}_{prune}). To make the search process aware of the sparsity mechanism, we need to solve another bi-level optimization problem that alternately updates the pruning mask and architecture parameters. Pruning mask parameters are binary values. Therefore, learning the mask parameters (m) is a challenging binary optimization problem. We solve this binary optimization problem by introducing floating point pruning parameters s and initializing them. Then we use SGD to solve the optimization problem and find the best floating-point pruning mask parameters. Finally, based on the values of pruning parameters, we select the top-k weight parameters with the highest values and assign one value to them. This step aims to jointly learn architecture parameters α_{prune} and mask parameters $\hat{m}$ to consider sparsity in learning architecture parameters.

Therefore, we use another bi-level optimization problem:

$$\begin{aligned} \alpha^{\star}_{prune} &= \min_{\alpha_{prune}} (\mathcal{L}_{val}(\theta^{*}_{pre} \odot \hat{m}(\alpha_{prune}), \alpha_{prune})) \\ s.t. \quad \hat{s}(\alpha_{prune}) &= \operatorname*{argmin}_{s} \mathcal{L}_{prune}(\theta^{*}_{pre}, \alpha_{prune}, s), \\ \hat{m}(\alpha_{prune}) &= \mathbb{1}(|\hat{s}(\alpha_{prune})| > |\hat{s}(\alpha_{prune})|_k) \end{aligned} \tag{7}$$

similar to step 1, the first-order approximation method is used to alternately update $\hat{m}$ and α_{prune} by gradient descent.

Step 3: fine-tune (learn $\hat{\theta}$). In the fine-tuning step, we update the non-zero weight parameters using SGD for the best sparse architecture to improve the network accuracy (Equation (8)).

$$\hat{\theta}_{t+1} = \hat{\theta}_t - \eta_{\hat{\theta}} \nabla_{\hat{\theta}} \mathcal{L}_{fine\text{-}tune}(\hat{\theta}_t \odot \hat{m}, \alpha^{\star}_{prune}) \tag{8}$$

where $\eta_{\hat{\theta}}$ and $\mathcal{L}_{fine\text{-}tune}$ denote the learning rate and the loss function for the fine-tuning step. We

ALGORITHM 1: Search Process of the DASS

Input: Dataset D, loss objectives: $\mathcal{L}_{train}$, $\mathcal{L}_{prune}$, and $\mathcal{L}_{fine-tune}$, training iteration T
Output: Fine-tuned sparse model

Step1: Pre-train
for $i \leftarrow 1$ to T **do**
  keep α^{t}_{pre} fixed, and obtain θ^{t+1}_{pre} by gradient descent with $\nabla_{\theta_{pre}} \mathcal{L}_{train}(\theta^{t}_{pre}, \alpha^{t}_{pre})$
  keep θ^{t+1}_{pre} fixed, and obtain α^{t+1}_{pre} by gradient descent with $\nabla_{\alpha_{pre}} \mathcal{L}_{val}(\theta^{t+1}_{pre}, \alpha^{t}_{pre})$
end for
Step2: Prune
for $i \leftarrow 1$ to T **do**
  keep α^{t}_{prune} fixed, and obtain s^{t+1} by gradient descent with $\nabla_{s} \mathcal{L}_{prune}(s^{t}, \alpha^{t}_{prune})$
  Compute $m^{t+1} = (|s^{t+1}| > |s^{t+1}|_k)$
  keep m^{t+1} fixed, and obtain α^{t+1}_{prune} by gradient descent with $\nabla_{\alpha_{prune}} \mathcal{L}_{val}(\theta^{*}_{pre} \odot m^{t+1}, \alpha^{t}_{prune})$
end for
Step3: Fine-tune
for $i \leftarrow 1$ to T **do**
  keep α^{*}_{prune} and $\hat{m}$ fixed and obtain $\hat{\theta}^{t+1}$ by gradient descent with $\nabla_{\hat{\theta}} \mathcal{L}_{fine-tune}(\hat{\theta}^{t} \odot \hat{m}, \alpha^{*}_{prune})$
end for
return Fine-tuned sparse model

show that the proposed three-step optimization algorithm can solve the complex bi-level problem in Equation (5) and finds optimal architecture parameters with higher generalization performance for sparse networks. Figure 7 compares the learning curves of DASS with DARTS$_{sparse}$ on the CIFAR-10 dataset. As shown, the DASS optimization algorithm significantly reduces the validation loss for the sparse network. Figure 8 compares the behavior of the generalization gap (train minus test accuracy) for DASS and DARTS$_{sparse}$. DASS has a lower generalization gap (up to 22%), indicating DASS better regularizes the validation loss across all epochs compared to DARTS$_{sparse}$. Algorithm 1 outlines our DASS for the differentiable neural architecture search for sparse neural networks. We analyze the time complexity of the proposed algorithm as follows: Our algorithm consists of three main steps: *Pre-train*, *Prune*, and *fine-tune*. Each step includes a loop with T iterations. At each step, we need to fix a variable and run forward and backward propagation

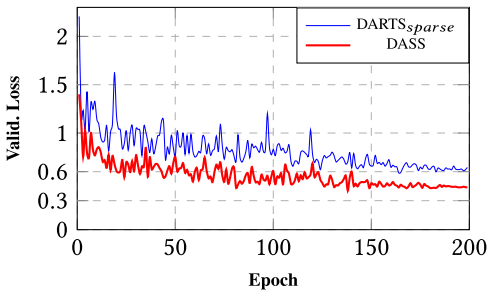

Fig. 7. Comparing learning curves (validation loss) of DASS and DARTS$_{sparse}$ on the searched architectures trained with the CIFAR-10 dataset.

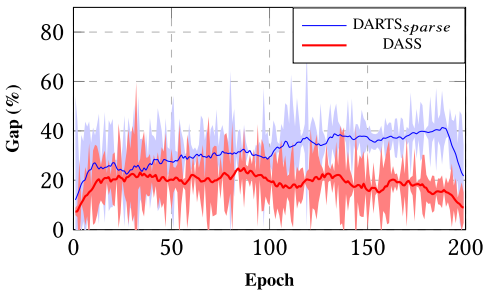

Fig. 8. Comparing the generalization gap of DASS and DARTS$_{sparse}$ over the CIFAR-10 dataset. The lower values for the generalization gap are better.

algorithms to train the architecture, weight, and pruning parameters. The time complexity of the forward and backward algorithms depends on the complexity of the architecture (the number of layers and neurons in the network). Therefore, for each step, the time complexity is calculated as follows: the complexity of forward propagation for a network with n layer and m nodes in each layer and T iterations is $O(T \cdot n \cdot m^3)$. The time complexity for the backward algorithm is $O(T \cdot n \cdot m^4)$. Therefore, Algorithm 1 with three distinct steps has total time complexity equal to $3 \cdot [(T \cdot n \cdot m^3) + (T \cdot n \cdot m^4)] = O(T \cdot n \cdot m^4)$. If we assume that the number of layers is equal to the number of neurons in each layer, then the total time complexity of Algorithm 1 is equal to $O(T \cdot n^5)$.

6 EXPERIMENTS

6.1 Experimental Setup

(1) DATASET: to evaluate DASS, we use CIFAR-10 [77] and ImageNet [78] public classification datasets. For the search process, we split the CIFAR-10 dataset into 30k data points for training and 30k for validation. We transfer the best-learned cells on CIFAR-10 to ImageNet [27] and re-train the final sparse network from scratch.

(2) Details on Searching Networks: we create a network with 16 initial channels and eight cells. Each cell consists of seven nodes equipped with a depth-wise concatenation operation as the output node. The `SparseConv` operations follow the ReLU+`SpasreConv`+Batch Normalization order. We train the network using SGD for 50 epochs with a batch size of 64 in the DASS pre-train step. Then, we update the values of pruning and architecture parameters for 20 epochs in the DASS pruning step. Finally, we fine-tune the network for 200 epochs. The initial learning rate for the DASS in pre-train, pruning, and fine-tuning steps is 0.025, 0.1, and 0.01, respectively. In our experiments, we use the cosine annealing learning rate [79]. We use weight decay = 3×10^{-4} and momentum = 0.9 in all steps. The search process for CIFAR-10 takes ≈3 GPU-days on a single NVIDIA® RTX A4000 that produces 4.35 Kg CO_2. We compare the sparse architecture designed by our method, DASS, with other dense and sparse networks. NAS-Bench-101 [80] and NAS-Bench-201 [81] are examples of NAS algorithm evaluation benchmarks. They consist of numerous dense designs and their respective performance. Due to the fact that they do not support sparse architectures, we cannot evaluate DASS using these benchmarks. Creating sparse benchmarks for evaluating NAS algorithms is a suggestion for future work.

(3) DASS variants and Hardware Configuration: Table 2 provides the configuration details of the DASS variants. Each variation is built by stacking a different number of DASS cells and the output channels of the first layer to generate networks for various resource budgets. Table 3 presents specifications of hardware devices utilized for evaluating the performance of DASS at inference time.

Table 2. Configuration of the DASS Variants

DASS	CIFAR-10				ImageNet		
	Tiny	Small	Medium	Large	Small	Medium	Large
#Cells	16	20	12	14	14	15	16
#Channels	30	36	86	108	48	86	128

#Cells: the number of stacked cells. #Channels: the number of output channels for the first `SparseConv` operation.

Table 3. Hardware Specification

Platform	Specification	Value
Search & Train	GPU	NVIDIA® RTX A4000 (735 MHz)
	GPU Memory	16 GB GDDR6
	GPU Compiler	cuDNN version 11.1
	System Memory	64 GB
	Operating System	Ubuntu 18.04
	CO_2 Emission/Day†	1.45 Kg
Real Hardware	Embedded GPU	NVIDIA® Jetson TX2 (735 MHz) 256 CUDA Cores
		NVIDIA® Quadro M1200 (735 MHz) 640 CUDA Cores
	Embedded CPU	ARM® Cortex™-A7 (1.2 GHz) 4/4 (Cores/Total Thread)
		Intel® i5-3210M Mobile CPU 5/4 (Cores/Total Thread)
Estimation‡	Xiaomi Mi9 GPU	Adreno 640 GPU (750 MHz) 986 GFLOPs FP32 (Single Precision)
	Myriad VPU	Intel® Movidius NCS2 (700 MHz) 28-nm Co-processor

† Calculated using the ML CO_2 impact framework: https://mlco2.github.io/impact/ [82].
‡ Performance Estimation using the nn-Meter framework [83].

6.2 DASS Compared to Dense Networks

Table 4 compares the performance of DASS against the state-of-the-art and the state-of-the-practice DNNs. We select the architecture with the highest accuracy, DrNAS [35], as the baseline for comparing compression rates. In comparison with DrNAS [35], DASS-Large provides 37.73× and 29.23× higher network compression rates while delivering a comparable accuracy (less than 2.5% accuracy loss) on the CIFAR-10 and ImageNet datasets, respectively. Compared to the best handcrafted designed network [33] on the CIFAR-10 (CCT-6/3x1), DASS-Large significantly decreases the parameters of the network by 29.9× with providing slightly higher accuracy.

6.3 DASS Compared to Sparse Networks

As we focus on improving the accuracy of sparse networks at extremely high pruning ratios, we compare DASS with other sparse networks with the unstructured pruning method at 99% pruning ratio (Table 5). In comparison with DARTS$_{sparse}$, DASS-Small yields 7.81% and 7.81% higher top-1 accuracies with 1.23× and 1.05× reduction in network size on the CIFAR-10 and ImageNet datasets, respectively. It indicates that the network design based on new search space and sparse

Table 4. Comparing the DASS Method with the State-of-the-art Dense Networks on the CIFAR-10 and ImageNet Datasets

Architecture	Year	Search Method	CIFAR-10 Top-1 Acc.(%)	CIFAR-10 #Params ($\times 10^6$)	CIFAR-10 #Params Compression	ImageNet Top-1 Acc.(%)	ImageNet Top-5 Acc.(%)	ImageNet #Params ($\times 10^6$)	ImageNet #Params Compression
ResNet-18‡ [2]	2016	-	91.0	11.1	−2.77×	72.33	91.80	11.7	−2.05×
PDO-eConv [32]	2020	-	94.62	0.37	+10.81×	-	-	-	-
FlexTCN-7 [33]	2021	-	92.2	0.67	+5.97×	-	-	-	-
CCT-6/3x1 [33]	2021	-	95.29	3.17	+1.26×	-	-	-	-
MomentumNet [34]	2021	-	95.18	11.1	−2.77×	-	-	-	-
DARTS (1^{st} order) [27]	2018	gradient	96.86	3.3	+1.21×	-	-	-	-
DARTS (2^{nd} order) [27]	2018	gradient	97.24	3.3	+1.21×	74.3	91.3	4.7	+1.21×
SGAS (Cri 1. avg) [84]	2020	gradient	97.34	3.7	+1.08×	75.9	92.7	5.4	+1.05×
SDARTS-RS [85]	2020	gradient	97.39	3.4	+1.17×	75.8	92.8	3.4	+1.67×
DrNAS [35]	2020	gradient	**97.46**	4.0	1.0×	**76.3**	92.9	5.7	1.0×
DASS-Small	2023	gradient	89.06	**0.017**	**+235.29×**	46.48	68.36	**0.029**	**+196.55×**
DASS-Medium	2023	gradient	92.18	0.054	+74.07×	68.34	82.24	0.082	+69.51×
DASS-Large	2023	gradient	95.31	0.106	+37.73×	73.83	85.94	0.195	+29.23×
DASS-Huge	2023	gradient	97.78	2.6	+1.54×	-	-	-	-

† The baseline for comparing the #params compressing rate is DrNAS [35] as the most accurate architecture.
‡ ResNet-18 results are trained in https://github.com/facebook/fb.resnet.torchTorch (July 10, 2018).
We highlight the best results in blue color.

Table 5. Comparing the DASS Method with Sparse Networks on the CIFAR-10 and ImageNet Datasets

Architecture	CIFAR-10 Top-1 Acc. (%)	CIFAR-10 #Params ($\times 10^3$)	CIFAR-10 Compression Rate†	CIFAR-10 NID‡	ImageNet Top-1 Acc. (%)	ImageNet Top-5 Acc. (%)	ImageNet #Params ($\times 10^3$)	ImageNet Compression Rate†	ImageNet NID‡
$DARTS_{sparse}$ [27]	81.25	21.0	100.47×	3.86	38.67	61.33	33.0	100×	1.11
$MobileNet\text{-}v2_{sparse}$ [30]	73.44	22.2	95.04×	3.30	17.97	36.72	34.87	94.63×	0.515
$ResNet\text{-}18_{sparse}$ [2]	90.62	111.6	18.90×	0.81	67.58	80.86	116.84	28.24×	0.578
$EfficientNet_{sparse}$ [31]	79.69	202.3	10.43×	0.39	-	-	-	-	-
MCUNET [8]	89.7	210.1	15.70	0.42	72.34	84.86	562.64	5.86×	0.128
DASS-Small	89.06	**17.0**	**124.11×**	**5.23**	46.48	68.36	**28.94**	**114.02×**	**1.606**
DASS-Medium	92.18	53.65	39.32×	1.71	68.34	82.24	81.95	40.26×	0.841
DASS-Large	**95.31**	105.5	20×	0.90	**73.83**	**85.94**	194.6	16.95×	0.38

† The baseline for comparing the compressing rate is full-precision and dense DARTS architecture.
‡ NID = Accuracy/#Parameters [86]. NID measures how efficiently each network uses its parameters.
We highlight the best results in blue color.

objective function finds better sparse architecture. In comparison with $ResNet\text{-}18_{sparse}$ on the CIFAR-10 dataset, we provide 1.56% and 4.7% higher accuracy with 2.08× and 1.05× network size reduction for DASS-Medium and DASS-Large, respectively. Compared to $ResNet\text{-}18_{sparse}$ on the ImageNet dataset, DASS-Medium provides 0.76% higher accuracy with 1.42× network size reduction. MCUNET [8] is a lightweight neural network for microcontrollers. It is designed by a tiny neural architecture search mechanism. Compared to MCUNET on the ImageNet dataset, DASS-Large provides 1% higher accuracy with 2.89× network size reduction. This result shows that only optimizing the size of the filters without considering the sparsity can not generate the best architecture. DASS directly search for the best operations in sparse version to design high-performance lightweight network. We can conclude that DASS increases sparse networks' accuracy at high pruning ratios compared to NAS-based and handcrafted networks.

6.4 Evaluation of DASS with Various Pruning Ratios

Table 6 compares DASS and the $DARTS_{sparse}$ method with three different pruning ratios including 90%, 95%, and 99% on the CIFAR-10 dataset. DASS achieves 1.57%, 1.04%, and 7.8% higher accuracies with 7%, 6.9%, and 23% network size reduction compared to the $DARTS_{sparse}$ at 90%, 95%, and 99% pruning ratios, respectively. Thus, DASS is significantly more effective at extremely higher pruning ratios (99%) than lower pruning ratios (90%).

Table 6. Evaluating the Effectiveness of DASS at Various Pruning Ratios

Architecture	90%		95%		99%	
	Accuracy	#Params ($\times 10^3$)	Accuracy	#Params ($\times 10^3$)	Accuracy	#Params ($\times 10^3$)
DARTS$_{sparse}$	95.31%	421	93.75%	210.5	81.25%	21.0
DASS-Small	**96.88%**	**391**	**94.79%**	**196.75**	**89.06%**	**17.0**

We highlight the best results in blue color.

Table 7. Comparing DASS with other Pruning Algorithms

Pruning Method	CIFAR-10			ImageNet			
	Backbone Arch.	Top-1 Acc.(%)	#Params ($\times 10^6$)	Backbone Arch.	Top-1 Acc.(%)	Top-5 Acc.(%)	#Params ($\times 10^6$)
SFP [22]	ResNet-20	92.08	0.269	ResNet-18	67.10	87.78	6.46
FPGM [23]		92.31	0.269		68.41	88.48	6.46
TAS$_{Pruning}$ [87]		93.16	0.232		69.15	88.48	7.40
Sparse [88, 89]	MobileNet-v2	91.53	0.671	MobileNet-v2	53.6	78.9	0.25
Sparse [89]	ShuffleNet	93.05	0.879	-		-	-
DASS-Small	-	89.06	**0.017**	-	46.48	68.36	**0.029**
DASS-Medium	-	92.18	0.054	-	68.34	82.24	0.082
DASS-Large	-	**95.31**	0.106	-	**73.83**	85.94	0.194

We highlight the best results in blue color.

6.5 DASS Compared to Other Pruning Methods

Table 7 compares DASS with state-of-the-art pruning algorithms. The results indicate that DASS outperforms other pruning algorithms with different backbone architectures on CIFAR-10 and ImageNet datasets. On CIFAR-10, DASS-Large shows a 1.6% higher accuracy and 3.8× reduction in the network size compared to the most accurate results provided by TAS$_{Pruning}$ [87]. DASS-Large also provides 4.68% accuracy improvement with 38.14× reduction in the network size over TAS$_{Pruning}$ [87] on ImageNet. In light of DASS' higher efficiency compared to other pruning methods, we can conclude that the pruning method was not the only reason for the DASS's effectiveness and it is independent of the pruning algorithm.

6.6 DASS Compared to Quantized Networks

Network quantization emerged as a promising research direction to reduce the computation of neural networks. Recently, [14–16] proposed to integrate the quantization mechanism into the differentiable NAS procedure to improve the performance of quantized networks. Table 8 compares DASS with the best results of NAS-based quantized networks. The compression rate is calculated as $\frac{\sum_{l=1}^{L} \#W_l \times 32}{\sum_{l=1}^{L} \#W_l^t \times q}$ where $\#W_l$ and $\#W_l^t$ are the number of weights in layer l for full-precision (32-bit) and quantized network with q-bit resolution [14]. DASS-Medium yields 0.24% and 3.24% higher accuracies and significantly higher compression rate by 2.7× and 4.24× compared to TAS [14] as the most accurate quantized network on the CIFAR-10 and ImageNet datasets, respectively.

6.7 Hardware Performance Results of DASS

We extensively study the effectiveness of DASS in the context of hardware efficiency by computing the inference time (latency) of various state-of-the-art sparse networks for a wide range of resource-constrained edge devices on the CIFAR-10 dataset (Figure 9). The batch size is equal to 1 for all experiments. It is worth noting that we did not utilize any simplification techniques, such as [90], to compact the sparse filters by fusing weight parameters. Our results reveal that the Pareto-frontier of DASS consistently outperforms all other counterparts by a significant margin, especially on CPUs that have very limited parallelism. DASS-Tiny as the fastest network improves the

Table 8. Comparing the DASS Method with Quantized Networks on CIFAR-10

Architecture	#bits (W/A)‡	CIFAR-10 Top-1 Acc.(%)	CIFAR-10 #Params (×10^6)	CIFAR-10 Compression Rate†	ImageNet Top-1 Acc.(%)	ImageNet Top-5 Acc.(%)	ImageNet #Params (×10^6)	ImageNet Compression Rate†
Binary NAS (A) [16]	1/1	90.66	2.4	44.0×	57.69	79.89	5.57	32.74×
TAS [14]	2/2	91.94	2.4	22.0×	65.1	86.3	5.57	16.37×
DASS-Small	32/32	89.06	**0.017**	**194.11×**	46.48	68.36	**0.029**	**196.55×**
DASS-Medium	32/32	92.18	0.054	61.11×	68.34	82.24	0.082	69.51×
DASS-Large	32/32	**95.31**	0.106	31.13×	**73.83**	85.94	0.194	29.38×

† The baseline for comparison is full-precision DARTS with 3.3M and 5.7M parameters for CIFAR-10 and ImageNet.
‡ (Weights/Activation Function).
We highlight the best results in blue color.

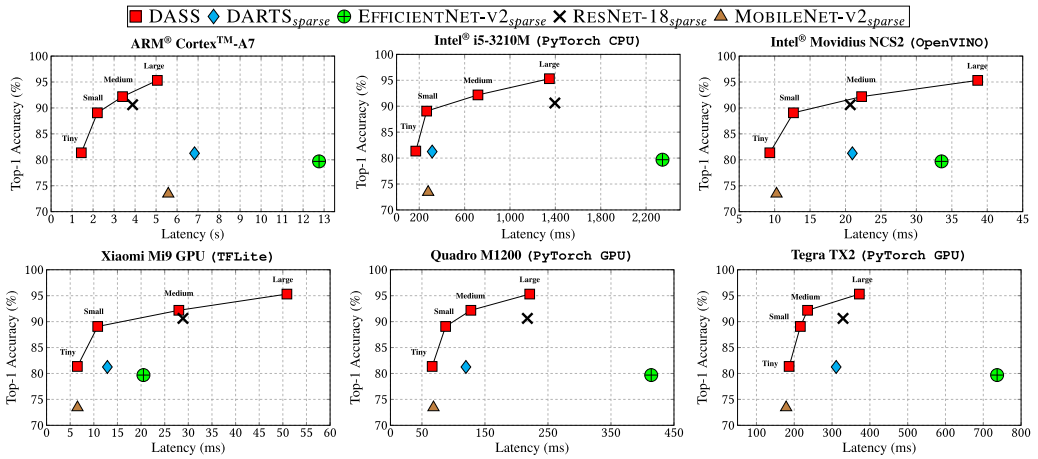

Fig. 9. Trade-off: accuracy v.s. measured latency. DASS-Tiny, DASS-Small, DASS-Medium, and DASS-Large are variants of DASS designed for different computational budgets (Table 2). DASS-Tiny consistently achieves higher accuracy with similar latency than MobileNet-v2$_{sparse}$ and provides lower latency while achieving better accuracy as DARTS$_{sparse}$.

accuracy from MobileNet-v2's 73.44% to 81.35% (+7.91% improvement) and accelerates the inference by up to 3.87×. More importantly, DASS-Tiny runs much faster than DARTS$_{sparse}$ by 1.67–4.74× with slightly better accuracy. Compared to ResNet-18$_{sparse}$ as the closest network to DASS in terms of accuracy, DASS-Medium provides 1.46% accuracy improvement and up to 1.94× acceleration on hardware.

6.8 Analyzing the Discrimination Power of DASS

We use the t-distributed stochastic neighbor embedding (t-SNE) method [91] for visualizing decision boundaries of dense high-performance architecture designed by DARTS, DARTS$_{sparse}$ (sparse dense DART architecture with pruning), and DASS (our sparse architecture) on the CIFAR-10 dataset. Figure 10 illustrates the decision boundaries of classification for each network. According to the results, DASS has a higher discrimination power than DARTS$_{sparse}$, and DASS with a 99% pruning ratio behaves very similarly to the dense and high-performance DARTS architecture.

6.9 Qualitative Analysis of the Searched Cell

Figure 11 shows the best cells searched by DASS-Small. An interesting finding is that, for the normal cell, DASS-Small tends to select SparseConv operation with larger kernel sizes (5×5), providing more pruning candidates to optimize the pruning mask. DASS-Small tends to leverage

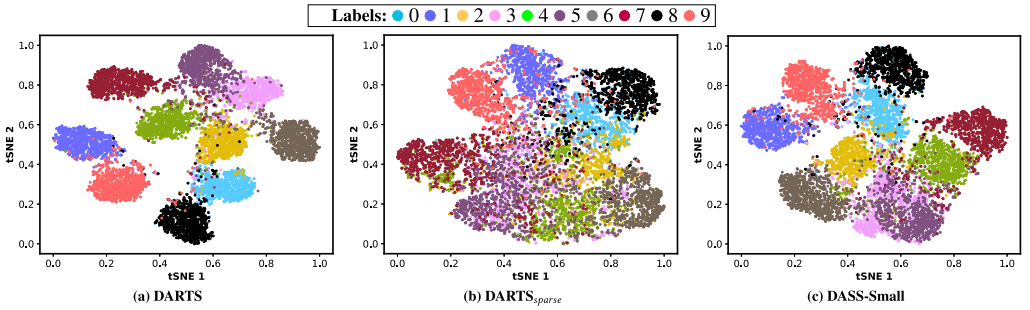

Fig. 10. Visualize decision boundary of (a) DARTS. (b) DARTS$_{sparse}$. (c) DASS-Large with the t-SNE embedding method.

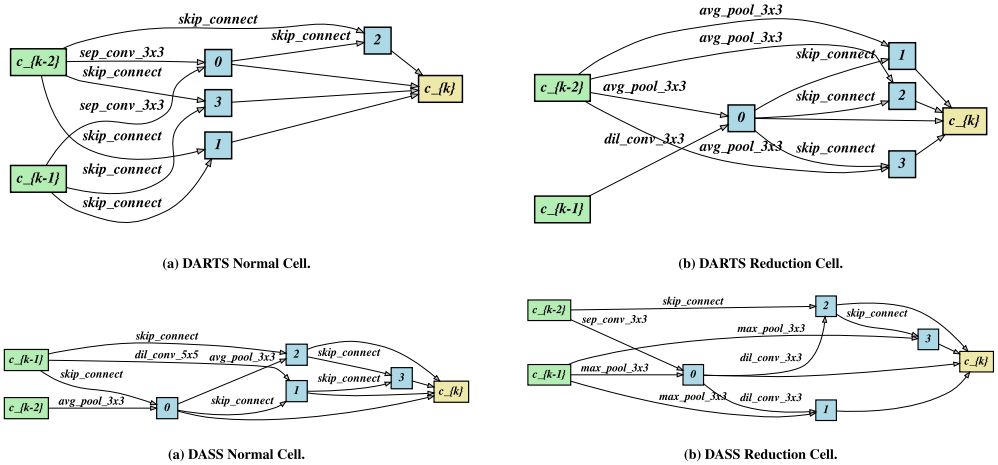

Fig. 11. The illustration of (a) normal cell and (b) reduction cell.

max-pooling operations in the reduction cell instead of avg-pooling operations. This is because the max-pooling operation has a higher feature extraction capability with sparse filters [92].

6.10 Reproducibility Analysis

To verify the reproducibility of results, the DASS-Small search procedure was run five times with different random seeds. Figure 12 plots the average of accuracy and loss variations as well as the shades to indicate the confidence intervals. Results show that, while the confidence interval is wide at first, the average of multiple runs converges to neural architectures with similar performance with an average standard deviation (STDEV) of 2.22%.

7 CONCLUSION AND FUTURE WORK

We propose DASS, a differentiable architecture search method, to design high-performance sparse architectures for DNNs. DASS significantly improves the performance of sparse architectures by proposing: (i) a new search space that contains sparse parametric operations; and (ii) a new search objective that is consistent with sparsity and pruning mechanisms. Our experimental results reveal that the learned sparse architectures outperform the architectures used in the state-of-the-art on both the CIFAR-10 and ImageNet datasets. In the long term, we foresee that our designed networks

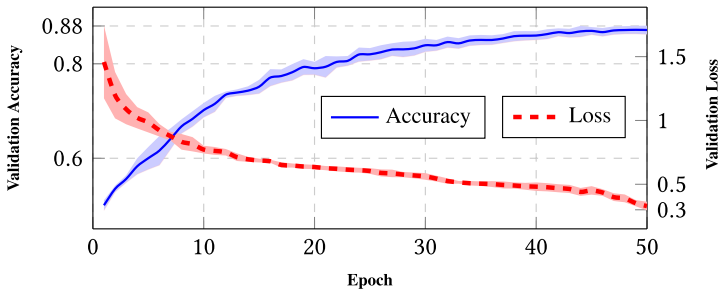

Fig. 12. Demonstrating the reproducibility of DASS results.

can effectively contribute to the goal of green artificial intelligence by efficiently utilizing resource-constrained devices as edge-accelerating solutions. We propose several promising directions for future research:

(1) Developing an indicator to evaluate sparse networks to enable training-free sparse NAS.
(2) Integration of quantization into the DASS framework to identify the optimal combination of sparsity and low-precision architecture.
(3) Decreasing the complexity of the optimization algorithm by combining pre-training and pruning steps.
(4) improving latency and energy efficiency by employing structured pruning techniques.

ACKNOWLEDGMENTS

The computations were enabled by the supercomputing resource Berzelius provided by National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.

REFERENCES

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems* 28 (2015).
[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[3] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. 2018. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience* 2018 (2018).
[4] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. 2018. How convolutional neural network see the world-A survey of convolutional neural network visualization methods. *arXiv preprint arXiv:1804.11191* (2018).
[5] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael Mahoney, and Kurt Keutzer. 2021. AI and memory wall. *RiseLab Medium Post* (2021).
[6] Mohammad Loni, Ali Zoljodi, Amin Majd, Byung Hoon Ahn, Masoud Daneshtalab, Mikael Sjödin, and Hadi Esmaeilzadeh. 2021. FastStereoNet: A fast neural architecture search for improving the inference of disparity estimation on resource-limited platforms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2021).
[7] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-device training under 256kb memory. *arXiv preprint arXiv:2206.15472* (2022).
[8] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319* (2020).
[9] Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. 2020. DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems* 73 (2020), 102989.
[10] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. 2020. Hydra: Pruning adversarially robust neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 19655–19666.
[11] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403.

[12] Xinyu Zhang, Ian Colbert, Ken Kreutz-Delgado, and Srinjoy Das. 2021. Training deep neural networks with joint quantization and pruning of weights and activations. *arXiv preprint arXiv:2110.08271* (2021).
[13] Enmao Diao, Ganghua Wang, Jiawei Zhan, Yuhong Yang, Jie Ding, and Vahid Tarokh. 2023. Pruning deep neural networks from a sparsity perspective. *arXiv preprint arXiv:2302.05601* (2023).
[14] Mohammad Loni, Hamid Mousavi, Mohammad Riazati, Masoud Daneshtalab, and Mikael Sjödin. 2022. TAS: Ternarized neural architecture search for resource-constrained edge devices. In *Design, Automation & Test in Europe Conference & Exhibition DATE'22, 14 March 2022, Antwerp, Belgium*. IEEE. http://www.es.mdh.se/publications/6351-
[15] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. 2020. Bats: Binary architecture search. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*. Springer, 309–325.
[16] Dahyun Kim, Kunal Pratap Singh, and Jonghyun Choi. 2020. Learning architectures for binary networks. In *European Conference on Computer Vision*. Springer, 575–591.
[17] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the Knowledge in a Neural Network (cite arxiv:1503.02531Comment: NIPS 2014 Deep Learning Workshop).
[18] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.
[19] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866* (2014).
[20] Kambiz Azarian, Yash Bhalgat, Jinwon Lee, and Tijmen Blankevoort. 2020. Learned threshold pruning. *arXiv preprint arXiv:2003.00075* (2020).
[21] Marwa El Halabi, Suraj Srinivas, and Simon Lacoste-Julien. 2022. Data-efficient structured pruning via submodular optimization. *arXiv preprint arXiv:2203.04940* (2022).
[22] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866* (2018).
[23] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4340–4349.
[24] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. 2020. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2009–2018.
[25] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. 2020. Neuron-level structured pruning using polarization regularizer. In *NeurIPS*.
[26] Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. 2022. Advancing model pruning via Bi-level optimization. *arXiv preprint arXiv:2210.04092* (2022).
[27] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[28] Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. 2022. b-darts: Beta-decay regularization for differentiable architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10874–10883.
[29] Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. 2022. beta-DARTS: Beta-decay regularization for differentiable architecture search. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'22)*. IEEE, 10864–10873.
[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.
[31] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.
[32] Zhengyang Shen, Lingshen He, Zhouchen Lin, and Jinwen Ma. 2020. Pdo-econvs: Partial differential operator based equivariant convolutions. In *International Conference on Machine Learning*. PMLR, 8697–8706.
[33] David W. Romero, Robert-Jan Bruintjes, Jakub M. Tomczak, Erik J. Bekkers, Mark Hoogendoorn, and Jan C. van Gemert. 2021. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059* (2021).
[34] Michael E. Sander, Pierre Ablin, Mathieu Blondel, and Gabriel Peyré. 2021. Momentum residual neural networks. *arXiv preprint arXiv:2102.07870* (2021).
[35] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. 2020. Drnas: Dirichlet neural architecture search. *arXiv preprint arXiv:2006.10355* (2020).
[36] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4780–4789.

[37] Mohammad Loni, Ali Zoljodi, Sima Sinaei, Masoud Daneshtalab, and Mikael Sjödin. 2019. Neuropower: Designing energy efficient convolutional neural network architecture for embedded systems. In *International Conference on Artificial Neural Networks*. Springer, 208–222.

[38] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Kay Chen Tan. 2021. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[39] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.

[40] Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[41] Yesmina Jaafra, Jean Luc Laurent, Aline Deruyver, and Mohamed Saber Naceur. 2019. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing* 89 (2019), 57–66.

[42] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2017. Smash: One-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).

[43] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*. Springer, 544–560.

[44] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*. PMLR, 550–559.

[45] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–34.

[46] Andrew Hundt, Varun Jain, and Gregory D. Hager. 2019. sharpdarts: Faster and more accurate differentiable architecture search. *arXiv preprint arXiv:1903.09900* (2019).

[47] Shahid Siddiqui, Christos Kyrkou, and Theocharis Theocharides. 2021. Operation and topology aware fast differentiable architecture search. In *2020 25th International Conference on Pattern Recognition (ICPR'21)*. IEEE, 9666–9673.

[48] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. 2021. Rethinking architecture selection in differentiable NAS. *arXiv preprint arXiv:2108.04392* (2021).

[49] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2019. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656* (2019).

[50] Zhixiong Yue, Baijiong Lin, Xiaonan Huang, and Yu Zhang. 2020. Effective, efficient and robust neural architecture search. *arXiv preprint arXiv:2011.09820* (2020).

[51] Ramtin Hosseini, Xingyi Yang, and Pengtao Xie. 2021. DSRNA: Differentiable search of robust neural architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6196–6205.

[52] Xiaojie Jin, Jiang Wang, Joshua Slocum, Ming-Hsuan Yang, Shengyang Dai, Shuicheng Yan, and Jiashi Feng. 2019. Rc-darts: Resource constrained differentiable architecture search. *arXiv preprint arXiv:1912.12814* (2019).

[53] Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. 2021. HELP: Hardware-adaptive efficient latency predictor for NAS via meta-learning. *arXiv preprint arXiv:2106.08630* (2021).

[54] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).

[55] Kevin Alexander Laube and Andreas Zell. 2019. Prune and replace nas. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA'19)*. IEEE, 915–921.

[56] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. 2020. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 493–503.

[57] Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. 2021. Dropnas: Grouped operation dropout for differentiable architecture search. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 2326–2332.

[58] Yadong Ding, Yu Wu, Chengyue Huang, Siliang Tang, Fei Wu, Yi Yang, Wenwu Zhu, and Yueting Zhuang. 2022. NAP: Neural architecture search with pruning. *Neurocomputing* (2022).

[59] Yanqing Hu, Qing Ye, Huan Fu, and Jiancheng Lv. 2022. Sparse DARTS with various recovery algorithms. In *Proceedings of the 8th International Conference on Computing and Artificial Intelligence*. 76–82.

[60] Ting Zhang, Muhammad Waqas, Hao Shen, Zhaoying Liu, Xiangyu Zhang, Yujian Li, Zahid Halim, and Sheng Chen. 2021. A neural network architecture optimizer based on DARTS and generative adversarial learning. *Information Sciences* 581 (2021), 448–468.

[61] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. 2019. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3977–3986.

[62] Yushuo Guan, Ning Liu, Pengyu Zhao, Zhengping Che, Kaigui Bian, Yanzhi Wang, and Jian Tang. 2022. Dais: Automatic channel pruning via differentiable annealing indicator search. *IEEE Transactions on Neural Networks and Learning Systems* (2022).
[63] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).
[64] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626* (2015).
[65] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).
[66] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).
[67] Yann LeCun, John S. Denker, and Sara A. Solla. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems*. 598–605.
[68] Babak Hassibi and David G. Stork. 1993. *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon*. Morgan Kaufmann.
[69] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067* (2019).
[70] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. 2019. Adversarial robustness vs. model compression, or both?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 111–120.
[71] Shupeng Gui, Haotao N. Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. 2019. Model compression with adversarial robustness: A unified optimization framework. *Advances in Neural Information Processing Systems* 32 (2019), 1285–1296.
[72] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. 2020. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the International Conference on Machine Learning*.
[73] Rebekka Burkholz, Nilanjana Laha, Rajarshi Mukherjee, and Alkis Gotovos. 2021. On the existence of universal lottery tickets. *arXiv preprint arXiv:2111.11146* (2021).
[74] Tianlong Chen, Zhenyu Zhang, Sijia Liu, Yang Zhang, Shiyu Chang, and Zhangyang Wang. 2022. Data-efficient double-win lottery tickets from robust pre-training. In *International Conference on Machine Learning*. PMLR, 3747–3759.
[75] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org
[76] Hervé Abdi. 2007. The Kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA* (2007), 508–510.
[77] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009), 7.
[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25 (2012), 1097–1105.
[79] Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).
[80] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*. PMLR, 7105–7114.
[81] Xuanyi Dong and Yi Yang. 2020. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326* (2020).
[82] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. 2019. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700* (2019).
[83] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 81–93.
[84] Guohao Li, Guocheng Qian, Itzel C. Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2020. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1620–1630.
[85] Xiangning Chen and Cho-Jui Hsieh. 2020. Stabilizing differentiable architecture search via perturbation-based regularization. In *International Conference on Machine Learning*. PMLR, 1554–1565.
[86] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark analysis of representative deep neural network architectures. *IEEE Access* 6 (2018), 64270–64277.
[87] Xuanyi Dong and Yi Yang. 2019. Network pruning via transformable architecture search. *Advances in Neural Information Processing Systems* 32 (2019).

[88] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).
[89] Kimessha Paupamah, Steven James, and Richard Klein. 2020. Quantisation and pruning for neural network compression and regularisation. In *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE, 1–6.
[90] Andrea Bragagnolo and Carlo Alberto Barbano. 2022. Simplify: A Python library for optimizing pruned neural networks. *SoftwareX* 17 (2022), 100907.
[91] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008).
[92] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. 2014. Mixed pooling for convolutional neural networks. In *International Conference on Rough Sets and Knowledge Technology*. Springer, 364–375.

Received 23 March 2023; revised 2 June 2023; accepted 10 July 2023