TR-NAS: MEMORY-EFFICIENT NEURAL ARCHITECTURE SEARCH WITH TRANSFERRED BLOCKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Architecture Search (NAS) is one of the most rapidly growing research fields in machine learning due to its ability to discover high-performance architectures automatically. Although conventional NAS algorithms focus on improving search efficiency (e.g., high performance with less search time), they often require a lot of memory footprint and power consumption. To remedy this problem, we propose a new paradigm for NAS that effectively reduces the use of memory while maintaining high performance. The proposed algorithm is motivated by our observation that manually designed and NAS-based architectures share similar low-level representations, regardless of the difference in the network's topology. Reflecting this, we propose a new architectural paradigm for NAS, called **Transfer-NAS**, that replaces several first cells in the generated architecture with conventional (hand-crafted) pre-trained blocks. As the replaced pre-trained blocks are kept frozen during training, the memory footprint can significantly be reduced. We demonstrate the effectiveness of the proposed method by incorporating it into Regularized Evolution and Differentiable ARchiTecture Search with Perturbation-based architecture selection (DARTS+PT) on NAS-Bench-201 and DARTS search spaces. Extensive experiments show that Transfer-NAS significantly decreases the memory usage up-to **50%** while achieving higher/comparable performance compared to the baselines. Furthermore, the proposed method is **1.98**$\times$ faster in terms of search time when incorporated to DARTS+PT on NAS-Bench-201 compared to the conventional method.

1 INTRODUCTION

Neural Architecture Search (NAS) has become an important domain in the machine learning field due to its superior performance. Many NAS algorithms have been developed (Zoph & Le, 2017; Zoph et al., 2018; Liu et al., 2019; Cai et al., 2019), and continue to raise in the future. The major advantage of NAS is to automatically discover the best architecture from a large search space on a target dataset. Since the solution can be found without human involvement, NAS has a wide range of applications such as image classification (Wu et al., 2019; Tan et al., 2019), object detection (Chen et al., 2020; Wang et al., 2020), and pruning (Dong & Yang, 2019).

Researchers have made a lot of attempts to improve the performance of NAS and reduce the searching time (Pham et al., 2018; Liu et al., 2019; Cai et al., 2019). Query-based NAS such as Regularized Evolution (RE) (Real et al., 2019) trains and evaluates thousands of small models before restoring the best model into the original size (enlarge the network's depth and number of channels) for evaluation. Gradient-based NAS algorithms (Liu et al., 2019; Xu et al., 2020) train a supernet which requires a lot of memory footprint. A natural question arises: *Can we perform the search step in NAS by training only a few cells rather than the whole network?* Technically, this paradigm shortens the training time and reduces the memory footprint, because the memory required for calculating the gradients in this case is smaller than the conventional approaches.

In this paper, we show that it is possible to perform efficient searching in NAS by replacing several first cells of a child network with pre-trained layers and let NAS search for the remaining cells. By analyzing the feature maps between the networks sampled from NAS-Bench-201 (Dong & Yang, 2020) search space and a hand-crafted one, namely ResNet (He et al., 2015), we observe that the representations are very similar among low-level features compared to their high-level features as

shown in Figure 1. It is noted that similar observation was reported in Kornblith et al. (2019). However, they compared the similarity among simple and similar architectures (ResNet's family and plain networks) which cannot directly lead us to our main motivation, while we compared similarity between a hand-crafted architecture with generated ones in NAS-Bench-201 that have high diversity in both topology and operations (e.g., $skip\ connection$, $conv3 \times 3$, $maxpooling$, ...).

Motivated from our preliminary experimental results in Figure 1, we propose to leverage several low-level features generated by a pre-trained network to speedup the search phase in NAS. This also helps to reduce the memory footprint significantly because we do not need to calculate the gradient for these pre-trained layers. To this end, the contributions of our paper are summarized as follows:

- We find out that the low-level features learned by a DARTS's supernet and networks sampled from NAS-Bench-201 search space, are similar to that of ResNet, regardless of their topology and operations.
- We leverage the features generated by a pre-trained baseline to improve the efficiency of NAS. Specifically, we replace several first layers of NAS-based networks with several pre-trained layers and freeze them, while leaving the other layers as trainable. This results in reduction of memory footprint and training time of the supernet and/or subnetworks.
- We demonstrate the effectiveness of our method by incorporating the proposed method into two search algorithms: evolutionary-based REA (Real et al., 2019) and gradient-based DARTS+PT (Wang et al., 2021). On NAS-Bench-201, we save up to **2.28**$\times$ memory footprint and run **1.98-1.32**$\times$ faster than the conventional method, while achieving a higher test accuracy. On DARTS search space, DARTS+PT using our proposed method can find the best cell in **0.53** GPU day and allocates **1.6**$\times$ less memory, while maintaining competitive compared to the conventional method.

2 Related work

Similarity of neural network representations. Recently, studies on learning the similarity of neural network representations have widely been conducted (Raghu et al., 2017; Morcos et al., 2018; Wang et al., 2018; Kornblith et al., 2019; Nguyen et al., 2021). Especially, Kornblith et al. (2019) provides a powerful similarity index for comparing neural network representations, namely Centered Kernel alignment (CKA). We adopt CKA to analyze the output feature similarity between hand-crafted and NAS-generated layers.

Given n examples, let $X \in \mathbb{R}^{n \times p_1}$ and $Y \in \mathbb{R}^{n \times p_2}$ are the representations of p_1 and p_2 neuron for this n examples. Let $K = XX^{\mathrm{T}}$ and $L = YY^{\mathrm{T}}$ be the Gram matrices, which are the similarities between a pair of examples in X and Y. Let $H = I_n - \frac{1}{n}11^{\mathrm{T}}$ denotes the centering matrix. Hilbert-Schmidt Independence Criterion (HSIC) is defined as $\mathrm{HSIC}(K, L) = \mathrm{vec}(HKH) \cdot \mathrm{vec}(HLH)/(n-1)^2$. The normalization index is applied to HISC to make it invariant to isotropic scaling, which is denoted as CKA and is defined as:

$$\mathrm{CKA}(K, L) = \frac{\mathrm{HSIC}(K, L)}{\sqrt{\mathrm{HSIC}(K, K)\mathrm{HSIC}(L, L)}}. \tag{1}$$

Neural Architecture Search. The goal of NAS is to automatically discover high-performance networks on a specific task. Reinforcement Learning (RL)-based, evolutionary-based, and gradient-based search algorithms are widely used for NAS (Zoph & Le, 2017; Pham et al., 2018; Zoph et al., 2018; Real et al., 2019; Liu et al., 2019). In the work of Zoph & Le (2017), the authors use RL and train a controller to generate the network's configurations (e.g., topology and operations). This method requires a lot of computational resources. Evolutionary-based NAS such as (Real et al., 2019) outperforms RL-based NAS in terms of accuracy and efficiency as it reaches higher accuracy given the same amount of time during searching. The major drawback of RL-based and evolutionary-based is the repetition process of training and evaluation of candidate networks, which demands a huge amount of GPU hours.

On the other hand, Gradient-based NAS (Liu et al., 2019; Xu et al., 2020; Chen et al., 2019) utilizes the back-propagation process to find the optimal network where the optimal model parameters and operations are found during training in an alternative manner. For example, Differentiable ARchiTecture Search (DARTS) (Liu et al., 2019) introduces architectural weights α beside network's

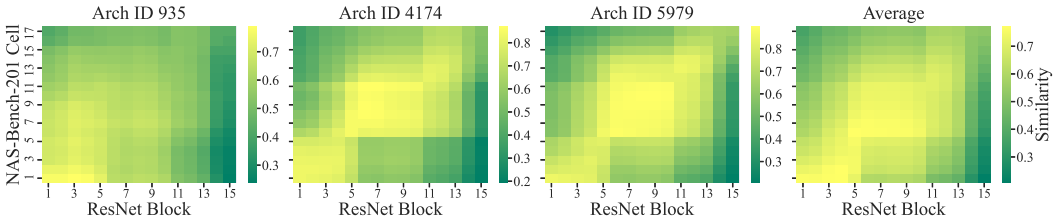


Figure 1: The similarity of representations between ResNet-32 blocks (x-axis) and NAS-Bench-201 cells (y-axis) for different architectures using CKA. The first, second, and third plots are the similarity for individual network. These networks obtain 80.36%, 89.63%, and 93.75% test accuracy on CIFAR-10. The last plot is the average similarity of 1000 architectures selected randomly.

weights w, forming a supernet consisting of large candidate networks. In order to find the optimal α and w, the process requires a bi-level optimization (Anandalingam & Friesz, 1992; Colson et al., 2007), which is computationally expensive. DARTS gets rid of this issue by approximating the architecture gradient by using only a single training step, which optimizes α and w alternately.

Relation between similarity of representations and NAS. Even though studying the similarity of neural network representations and NAS are two different research areas, it is important to understand whether the findings from the previous works on the similarity of feature maps between networks still hold for NAS. Not surprisingly, NAS-based architectures and hand-crafted networks generate similar representations for some layers after training, as depicted in Figure 1. This motivates us to develop a simple yet effective method to reduce the memory footprint in NAS. In our work, we use CKA proposed by Kornblith et al. (2019) for calculating the similarity[1].

3 METHOD

In this part, we first study the similarity of neural network representations from NAS perspective. We first show that NAS-based networks and hand-crafted networks learn and generate similar representations for several shallow layers. We then propose a simple method that reuses the features from a pre-trained hand-crafted network, which can help to improve the efficiency of NAS in terms of memory footprint and training time.

3.1 KEY OBSERVATIONS

We consider comparing the similarity for two cases: (i) query-based NAS algorithms (Zoph & Le, 2017; Real et al., 2019) which train a lot of subnetworks; (ii) gradient-based NAS methods (Liu et al., 2019; Chen et al., 2019) which train a supernet.

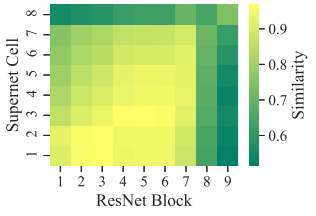


Figure 2: The similarity between ResNet-20 blocks (x-axis) and DARTS's supernet cells (y-axis) using CKA.

Case 1. We train a ResNet-32 on CIFAR-10 dataset (Krizhevsky & Hinton, 2009) using the same hyper-parameters used in NAS-Bench-201 (Dong & Yang, 2020). More details can be found in Appendix A.1. We randomly select 1000 architectures from NAS-Bench-201 and measure the similarity between the representations[2] generated by these architectures and the trained ResNet-32 via CKA (Kornblith et al., 2019) similarity metric. The weights of NAS-Bench-201 architectures are obtained from their official website. We take the average of 1000 similarities and plot them in Figure 1. It can be seen that the feature maps generated by NAS-Bench-201 architectures share similar characteristics to those from ResNet-32. Especially, the similarity is significantly higher for low-level representations while lower for high-level representations. This suggests that some of the initial

[1]Current work uses this method due to its simplicity. However, we believe that a proper metric will tell us how many pre-trained layers should be used before performance collapse (i.e., the searched network achieves inferior accuracy).

[2]The feature maps generated by a ResNet block or a cell from NAS-Bench-201 architectures.

layers of candidate networks in NAS can be replaced with layers from pre-trained hand-crafted networks. Moreover, this approach reduces the memory footprint since it reduces the search space size (during backpropagation).

Case 2. We further investigate the similarity between a pre-trained ResNet-20 and a supernet, which is used in DARTS. We compute the similarity for the representations generated by supernet cells and ResNet blocks. Figure 2 demonstrates that the supernet and ResNet generate similar low-level and mid-level patterns. This behavior is consistent with previous observation.

3.2 Proposed Memory-Efficient Neural Architecture Search

In this section, we introduce our method for reducing the memory footprint in NAS. Based on our observation about similarity of representations between hand-crafted and NAS based networks, we propose replacing a first few layers of NAS based networks with pre-trained layers of hand-crafted networks. Figure 3 shows the differences between the conventional and the proposed method for NAS.

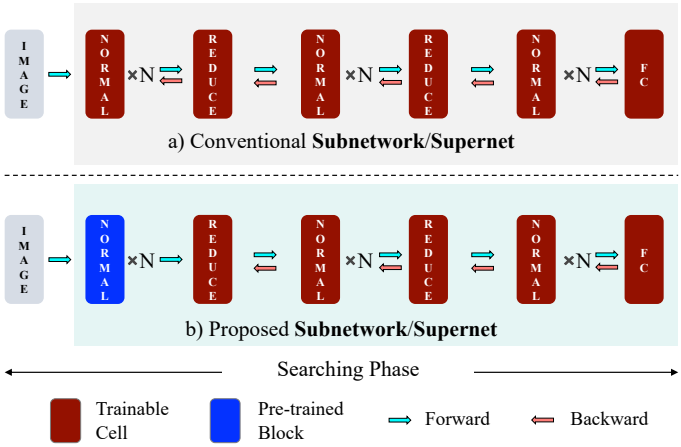


Figure 3: Comparison between the conventional method and the proposed method. (a) The conventional method trains all layers of a network, which requires memory for calculating the gradients of all layers. (b) The proposed method replaces first few layers of NAS-based network with pre-trained layers of a hand-crafted network. The weights in these pre-trained layers are frozen, while others are learned through backpropagation. Since we only need to compute the gradients for the remaining layers, the memory footprint is reduced while preserving the performance.

Let A be an untrained network with L cells and is expressed as $A = \{l_{A1}, l_{A2}, ..., l_{AL}\}$. Similarly, let B be a pre-trained baseline with K blocks and is expressed as $B = \{l_{B1}, l_{B2}, ..., l_{BK}\}$. Since A and B share similar representations for some cells and blocks after training, a straightforward way[3] to reuse the features from pre-trained network B is to plug these features to untrained network A. Hence, the proposed network which replaces i cells with j blocks is defined as $S = \{l_{B1}, ..., l_{Bj}, l_{A(i+1)}, ..., l_{AL}\}$, where i and j are sandwiched between 1 and $L-1$. The choice of j controls how many pre-trained blocks from B are used while i controls how many cells of A are replaced. If j and i are large meaning that we only train a few cells, the pre-trained blocks are dominating the NAS cells. Intuitively, this makes the proposed network S looks like the pre-trained network B, thus degrading the performance of NAS. During training the network S, we only update the weights from $l_{A(i+1)}$ to l_{AL}. The proposed method can be used for query-based NAS and gradient-based NAS. Here, we formally describe how to use the proposed method for NAS.

Query-based NAS. In many query-based NAS algorithms (Zoph & Le, 2017; Real et al., 2019), a large number of child networks are sampled and trained for limited epochs. Instead of training these networks, which requires computing the gradient for all cells, we replace them with the proposed paradigm and keep other settings as default. Let $\mathcal{A}$ be the search space and $a \in \mathcal{A}$ be an architecture.

[3]There are other ways to reuse features. Despite directly plug-in, we find that, with a simple learning rate, warmup paradigm can make training stable as it makes the untrained cells to adapt pre-trained blocks gradually.

Instead of training a, we train $s = \{l_{b1}, ..., l_{bj}, l_{a(i+1)}, ..., l_{aL}\}$ where b is a pre-trained architecture and is shared for all networks in $\mathcal{A}$. The performance of s on the validation set is used for updating the controller in Zoph & Le (2017) or selecting the parent for mutation in Real et al. (2019). We term s as Transfer-Net-i-j (Tr-Net-i-j).

Gradient-based NAS. Differentiable architecture search (Liu et al., 2019; Xu et al., 2020) trains a supernet from which subnetworks are generated. Although the supernet comprises of a highly complex topology, the starting cells still produce similar representations to a hand-crafted one after training as demonstrated in Figure 2. Thus, the proposed method is also applicable for Gradient-based NAS. Let A be the orginal supernet. The proposed network S now acts as the supernet, which we denote as Transfer-Supernet-i-j (Tr-Supernet-i-j). The search phase is now conducted on Tr-Supernet-i-j, which requires less memory footprint. Since Tr-Net and Tr-Supernet utilize pre-trained blocks, they belong to Tr-NAS family.

4 EXPERIMENTS

In this section, we first compare the rank correlation between the validation accuracy and final test accuracy of the proposed method and the conventional one on NAS-Bench-201. Then, we evaluate the performance of REA and DARTS+PT using the proposed method. Unless stated otherwise, the training time and memory footprint are measured on a single Nvidia Geforce 1080 Ti with PyTorch deep learning framework.

4.1 RANK CORRELATION COMPARISON

Dataset and baseline. We use NAS-Bench-201 (Dong & Yang, 2020), a benchmark dataset for NAS. NAS-Bench-201 consists of 15,625 networks where five operations exist in the search space, namely $none$, $average\ pooling$, $conv1 \times 1$, $conv3 \times 3$, and $skip\ connection$. These networks are trained on CIFAR-10, CIFAR-100, and ImageNet-16-120. The baseline chosen for comparison is the conventional one, which trains for 12 epochs. We term this as 'Original'.

Architectures. We randomly sample 1000 networks from NAS-Bench-201. We replace these networks with the proposed paradigm. The pre-trained network is a ResNet-32[4]. We set i and j to 5, meaning that we replace the first 5 cells of these networks with the first 5 pre-trained ResNet blocks.

Table 1: Rank correlation of 1000 networks sampled randomly on NAS-Bench-201.

Dataset	KROCC			SROCC			Rank Preserved
	Original	Tr-Net-5-5	Gain↑	Original	Tr-Net-5-5	Gain↑	
CIFAR-10	0.6128	0.7883	0.1755	0.7888	0.9381	0.1493	✓
CIFAR-100	0.6600	0.6864	0.0264	0.8384	0.8698	0.0314	✓
ImageNet-16-120	0.6656	0.7018	0.0362	0.8394	0.8786	0.0392	✓

Performance criteria. Once the training is finished, we compute Spearman Rank Order Correlation Coefficient (SROCC) and Kendall Tau Rank-Order Correlation Coefficient (KROCC) between the accuracy on the validation set and the test set for the Original and the proposed method. We obtain the accuracy on the validation set and test set for the original networks from NAS-Bench-201 dataset.

Table 2: Average allocated memory in GB.

Method	Original	Tr-Net-5-5
CIFAR-10	1.12	0.53
CIFAR-100	1.12	0.53
ImageNet-16-120	0.28	0.13

Training setup. We use the same training set and validation set in our experiments following NAS-Bench-201 for fair comparison. We use the same hyper-parameters as in NAS-Bench-201, except that we train our methods for 18 epochs (including 13 warmup epochs). It should be noted that even we train our networks with more epochs, the total training time of our networks is still less than the conventional method. Please refer to Appendix A.2 for more information.

[4]A good starting point for choosing the depth of the pre-trained baseline is to minimize the difference between the receptive field of blocks and cells. We did not conduct extensive tuning the depth of the baseline because we want to utilize a pre-trained model, which is already available to speedup NAS.

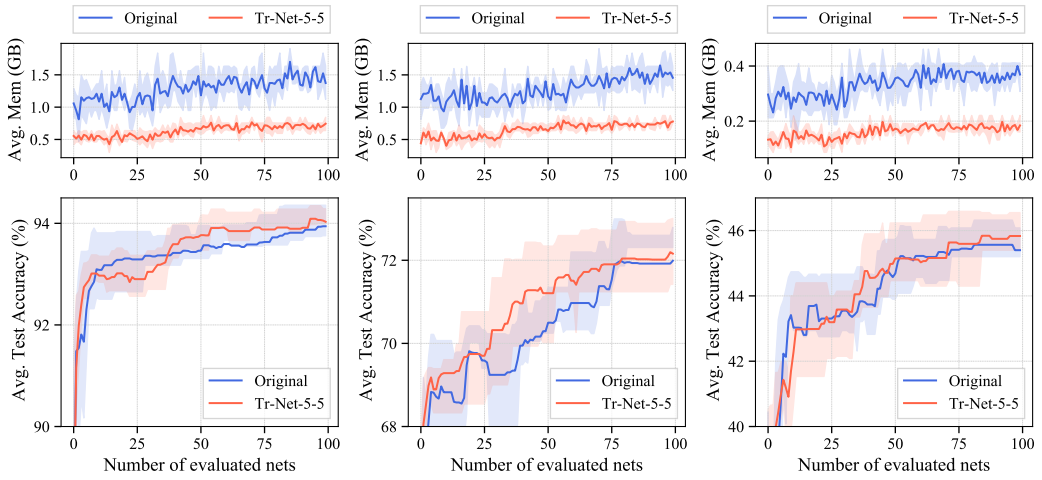
Result. Table 1 summarizes the results on three datasets i.e., CIFAR-10, CIFAR-100, and ImageNet-16-120. It can be seen that the proposed method achieves a higher correlation than the conventional method for all of the datasets. Notably, there is a huge improvement in rank correlation on CIFAR-10 when using the proposed method. The results from Table 1 indicate that, despite sharing the same representations for several first layers, the proposed method is able to preserve the ranking (even better). We obtain the allocated memory on GPU during training and present in Table 2. The results suggest that the proposed method significantly reduces the memory footprint (**2.11**$\times$ reduction) compared to the conventional method. We further evaluate the rank correlation for top-k architectures, which is crucial for assessing the performance as good methods should have a strong correlation for top-performing networks. We set k to 1000 and summarize the results in Table 3.

Table 3: Rank correlation of top-k networks on NAS-Bench-201

Dataset	**KROCC**			**SROCC**			**Rank Preserved**
	Original	Tr-Net-5-5	Gain↑	Original	Tr-Net-5-5	Gain↑	
CIFAR-10	0.4135	0.4476	0.0341	0.5909	0.6351	0.0442	✓
CIFAR-100	0.3189	0.3652	0.0463	0.4623	0.5173	0.0550	✓
ImageNet-16-120	0.1628	0.2087	0.0459	0.2391	0.3021	0.0630	✓

As shown in Table 3, the proposed method performs consistently well for all datasets. In general, the proposed method achieves a higher rank correlation than the conventional method. These sets of experiments indicate that the proposed method is well-suited for NAS.

4.2 Result on query-based NAS



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet-16-120

Figure 4: Performance comparison of REA on NAS-Bench-201. **Top:** The average memory footprint during training. **Bottom:** The average test accuracy on NAS-Bench-201.

We now demonstrate the effectiveness of the proposed method by incorporating to NAS algorithms. For this purpose, we use Regularized Evolution REA (Real et al., 2019), a query-based NAS algorithm. We train ours for 18 epochs (including 13 warmup epochs), other hyper-parameters are identical to those used in NAS-Bench-201. The search phase is terminated when the number of evaluated networks exceeds 100. We conduct the experiment 10 times with different seeds and plot the average test accuracy in Figure 4.

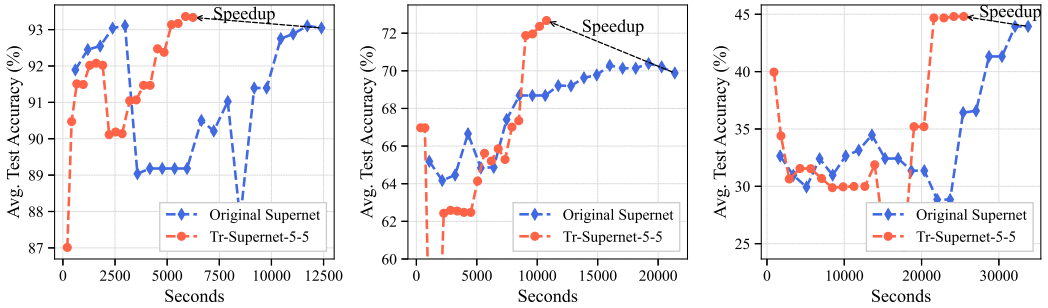
As shown in Figure 4, using the proposed method, REA is able to find the top-performing network. Compared to the original method, the proposed method achieves similar performance (Figure 4. Bottom) while using roughly **2**$\times$ less memory footprint (Figure 4. Top). It is noticeable that REA outputs higher average test accuracy during the search phase using our method. This behavior is natural since our method achieves a good rank correlation than the conventional one.

4.3 RESULT ON DIFFERENTIABLE NAS

We shift our evaluation to another type of NAS algorithms, which uses the gradient to guide the search. We incorporate the proposed method to DARTS+PT (Wang et al., 2021), a perturbation-based architecture selection that performs on top of DARTS. The authors of DARTS+PT show that the architectural weights α do not represent the strength of the operations. Thus, they introduce an alternative way to derive the final architecture, which relies on the contribution of the operations to the supernet's accuracy. Specifically, after the supernet converged, the operation which has less impact on the supernet's accuracy is removed from the supernet. Then, we tune the supernet for some epochs and repeat the process until the stopping criteria is met (e.g., becoming a network with a single-path).

4.3.1 NAS-BENCH-201 SEARCH SPACE

In this section, we evaluate the performance of DARTS+PT using our proposed method on NAS-Bench-201 search space.



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet-16-120

Figure 5: Performance of DARTS+PT on CIFAR-10, CIFAR-100, and ImageNet-16-120.

Supernet. The original supernet is a cell-based paradigm, which repeatedly stacks cell. Each cell has 6 edges and is stacked for 5 times for the first, second, and third stages. Our Tr-Supernet-5-5 is formed by replacing the first stage of the original supernet with those from a pre-trained ResNet-32 (i.e., the first 5 residual blocks).

Training setup. We train Tr-Supernet-5-5 and Original Supernet for 50 epochs (including 13 warmup epochs for ours) and then perform architecture selection followed DARTS+PT. After the operation is removed from the supernet, we tune the supernet for 10 epochs. The search phase is finished when all edges are processed. Other hyper-parameters are the same as in previous work (Wang et al., 2021). Additionally, we enable Cutout (Devries & Taylor, 2017) during training and tuning the proposed supernet and the original one. The results obtained without Cutout is presented in Ablation 4.4.2.

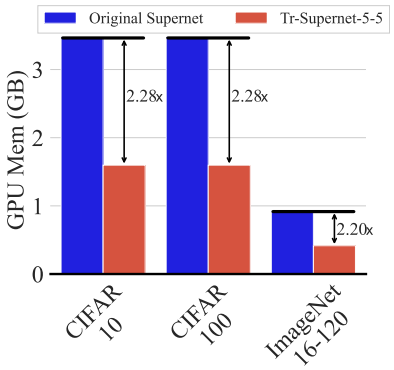


Figure 6: The allocated memory of the supernet with a batch size of 128.

Result. We run experiments 25 times on CIFAR-10/CIFAR-100 and 5 times on ImageNet-16-120 with different random seeds and report the mean test accuracy. Figure 5 illustrates our results on NAS-Bench-201. As we can see from Figure 5, using the proposed supernet, DARTS+PT finishes the search phase faster than the conventional supernet (about **1.98**× reduction on CIFAR-10/CIFAR-100 and **1.32**× on ImageNet-16-120). We summarize the average test accuracy in Table 4. Specifically, the proposed method achieves a per-

Table 4: Test Accuracy using DARTS+PT on NAS-Bench-201.

Dataset	Original	Ours
CIFAR-10	93.04	93.49
CIFAR-100	69.88	72.67
ImageNet-16-120	43.94	44.8

formance gain of **0.45%**, **2.79%**, and **0.86%** on CIFAR-10, CIFAR-100, and ImageNet-16-120, respectively. We plot the allocated memory required to train the supernet in Figure 6. Notably, the proposed supernet reduces the memory footprint by **2.2**× for all datasets while achieving higher test accuracy compared to the conventional supernet.

4.3.2 DARTS SEARCH SPACE

We perform experiments on DARTS search space, which is much larger than NAS-Bench-201 search space. The supernet is formed by stacking the cells for 8 times. The position of reduction cell is at 1/3 and 2/3 of the depth of the supernet. We use a pre-trained ResNet-20 to provide intermediate features for the supernet. We replace the first stage of supernet with those from ResNet-20 (i.e., the first two supernet cells are replaced with the first three ResNet-20 blocks). The configuration for the pre-trained ResNet can be found in Appendix A.1. We use the same hyper-parameters for training and tuning the supernet (Liu et al., 2019; Wang et al., 2021). Additionally, we apply warmup for 5 epochs and enable Cutout (Devries & Taylor, 2017) during the search phase. We run the experiment 4 times with different random seeds and report the average (best) test error in Table 5. The result of DARTS+PT is obtained by running the official code provided by authors. From Table 5, we can see that the proposed method works well on a larger search space. The conventional DARTS takes 0.4 day to finish the search phase and 0.85 day when applying perturbation-based architecture selection. Using the proposed method, DARTS+PT finishes the search phase in **0.53** day and allocates **1.6**× less memory, while maintaining the performance.

Table 5: Performance comparison on DARTS search space.

Method	**Test Error (%)**	**Search Cost***	**GPU Mem***
DARTS+PT	2.78 (2.68)	0.82	8.12
+ Cutout	2.77 (2.71)	0.85	8.12
+ Ours	2.94 (2.70)	0.50	4.64
+ Ours + Cutout	2.81 (2.60)	0.53	4.64

* in GB
* GPU day

4.4 ABLATION STUDIES

4.4.1 STABILIZING THE TRAINING VIA LEARNING RATE WARMUP

Since the proposed method utilizes some pre-trained blocks while others are randomly initialized, the network may fail to converge using the conventional optimization technique. We suggest stabilizing the network via learning rate warmup for a few epochs. It is noted that the warmup phase does not introduce any extra cost. The warmup phase begins with a learning rate of 0, and gradually increases to the initial learning rate for n epochs. This step encourages the untrained cells to adapt to the pre-trained blocks slowly. To find the best warmup epochs for our setting, we randomly sample 250 networks from NAS-Bench-201 search space, replace the first i cells with j pre-trained ResNet blocks, and train them on CIFAR-100 for 18 epochs with different warmup epochs n. We use KROCC as a criterion for comparison.

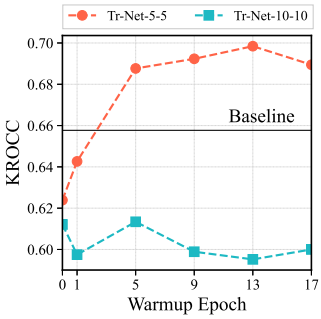


Figure 7: Rank correlation under different warmup epochs on CIFAR-100, NAS-Bench-201. The baseline is the conventional method, which trains the network for 12 epochs.

We investigate two cases: (i) i and j are equally set to 5, denoted as Tr-Net-5-5; (ii) i and j are set to 10, denoted as Tr-Net-10-10. We set n from 1 to 17 with a step size of 4 and show the results in Figure 7. For the first case, we can see that without the warmup phase (0 warmup epochs), the rank correlation of the proposed method is below the conventional method. This indicates that the training is unstable that causes the network failed to converge. When using learning rate warmup, the rank correlation gradually increases and reaches its peak when n is 13, then starts decreasing. In addition, the proposed method always outperforms the conventional method when we perform warmup for more than 3 epochs. For the second case, one can observe that if we replace too many cells with pre-trained blocks, the rank correlation is lower than the baseline. This suggests that choosing the right i and j can have a good trade-off between performance and memory reduction.

4.4.2 THE EFFECTIVENESS OF CUTOUT AUGMENTATION

We investigate how Cutout (Devries & Taylor, 2017) can help to improve the performance of the proposed method for gradient-based NAS algorithms. We study two cases: (i) i and j is set to 5 which is denoted as Tr-Supernet-5-5; (ii) i and j is set to 10 which is denoted as Tr-Supernet-10-10. We perform the experiment on NAS-Bench-201 using CIFAR-100 with different warmup epochs n. We compare the performance of DARTS+PT using our supernet and the conventional one trained with and without Cutout. We show the results in Figure 8. For the first case, DARTS+PT using our supernet outperforms the original supernet when n is greater than 7. Tr-Supernet-5-5 achieves the best performance with 13 warmup epochs. For the second case, Tr-Supernet-10-10 achieves comparable performance compared to the original supernet trained with Cutout. However, when comparing to the conventional method trained without Cutout, there is a little drop in accuracy.

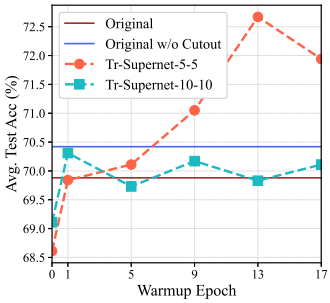


Figure 8: Performance of DARTS+PT with Cutout on CIFAR-100, NAS-Bench-201.

To find out the reason that Cutout helps to improve the performance of our method, we compute the full Hessian $\nabla^2_\alpha \mathcal{L}_{valid}$ of validation loss w.r.t the architectural parameters α and show the dominant eigenvalue in Figure 9. Zela et al. (2020) reveals that large dominant eigenvalue of the Hessian degrades the performance and suggests that using regularization technique may improve the performance. As demonstrated in Figure 9, applying Cutout to the original supernet does not help to reduce the dominant eigenvalue as it continues to raise. By contrast, the proposed method achieves better results because the dominant eigenvalue fluctuates around some values when enabling Cutout.

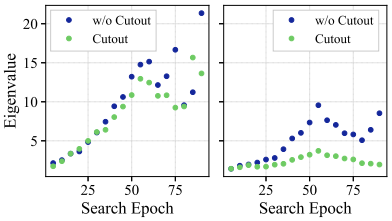


Figure 9: Dominant Eigenvalue during search (Left: Original Supernet, Right: Tr-Supernet-5-5, warmup 13 epochs).

5 DISCUSSION & FUTURE WORK

Since, the proposed method fuses pre-trained blocks and untrained cells in a straight-forward manner, there are several potential directions to further improve the performance of NAS while reducing the memory footprint and training time. First, a proper similarity index may help to determine how many untrained cells are suitable for replacement. Second, there is a possibility to develop a single pre-trained baseline, which can be used as a starting point for NAS to perform on other target tasks that are similar to each other (Kornblith et al., 2019). Also, the performance of NAS may depend on the performance of the pre-trained network. Thus, designing a good pre-trained baseline which can be applied for various NAS algorithms can be a promising work. In our work, we keep things as simple as possible to demonstrate the usefulness of the proposed method, and leave other improvements as future works.

6 CONCLUSION

In this work, we propose a simple yet effective method to reduce the memory footprint and shorten the training time in NAS, by replacing several first NAS-cells with those from a pre-trained hand-crafted network. Our work is motivated by the observation that once converged, both hand-crafted and NAS-based architectures learn similar representations especially at low-level layers. Our method outperforms the conventional method in terms of rank correlation of validation accuracy to test accuracy on NAS-Bench-201 while requiring less memory footprint. Additionally, our method can be incorporated into different types of NAS algorithms, such as query-based or gradient-based methods, without any modification. Overall, the proposed method uses less memory footprint and shortens the training time while achieving comparable or even higher performance than the conventional methods.

REFERENCES

G. Anandalingam and T. L. Friesz. Hierarchical optimization: An introduction. *Annals of Operations Research*, 34(1):1–11, Dec 1992. ISSN 1572-9338. doi: 10.1007/BF02098169. URL https://doi.org/10.1007/BF02098169.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HylVB3AqYm.

Bo Chen, Golnaz Ghiasi, Hanxiao Liu, Tsung-Yi Lin, Dmitry Kalenichenko, Hartwig Adam, and Quoc V. Le. Mnasfpn: Learning latency-aware pyramid architecture for object detection on mobile devices. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *CoRR*, abs/1904.12760, 2019. URL http://arxiv.org/abs/1904.12760.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, Sep 2007. ISSN 1572-9338. doi: 10.1007/s10479-007-0176-2. URL https://doi.org/10.1007/s10479-007-0176-2.

Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/a01a0380ca3c61428c26a231f0e49a09-Paper.pdf.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/kornblith19a.html.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/a7a3d70c6d17a73140918996d03c014f-Paper.pdf.

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KJNcAkY8tY4.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4095–4104, 2018.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6078–6087, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33 (01):4780–4789, Jul. 2019. doi: 10.1609/aaai.v33i01.33014780. URL https://ojs.aaai.org/index.php/AAAI/article/view/4405.

Oleg Semery. Pytorchcv convolutional neural networks for computer vision. https://github.com/donnyyou/torchcv, 2020.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Liwei Wang, Lunjia Hu, Jiayuan Gu, Yue Wu, Zhiqiang Hu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 9607–9616, Red Hook, NY, USA, 2018. Curran Associates Inc.

Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable NAS. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PKubaeJkw3.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJlS634tPr.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gDNyrKDS.

Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

A APPENDIX

A.1 DETAIL OF ARCHITECTURES AND TRAINING SETUP FOR PRE-TRAINED RESNET

Actually one can use a pre-trained ResNet available in PyTorchCV (Semery, 2020) database as we do not modify the ResNet architecture. We do train our ResNet for a fair comparison to the conventional method because on NAS-Bench-201, the original train and test sets of CIFAR-10, CIFAR-100 are split into new train, validation, test sets.

ResNet-32. The network is created by stacking basic residual block for 15 times. The downsampled block is located at 6-th and 11-st block. The number of channel is 16, 32, 64 for the first, second, and third stages, respectively.

ResNet-20. Similar to ResNet-32, we stack the basic residual block for 9 times, reduce the dimension by half at 4-th and 7-th block. The number of channel is set to 64, 128, and 256 for the first, second, and third stages, respectively.

The hyper-parameters for training ResNet are summarized in Table 6.

Table 6: The hyper-parameters for training.

Optimizer	SGD
Learning rate	0.1
Epoch	200
Nesterov	Yes
Momentum	0.9
LR Schedule	Cosine
Weight decay	0.0005
Batch size	256

A.2 TRAINING TIME OF THE PROPOSED METHOD

The training time of a random network sampled from NAS-Bench-201 is displayed in Figure 10. We can see that the proposed method require less time to finish one epoch. As a result, we can increase the number of epochs for training the proposed method as long as it does not exceed the training time of conventional method for a fair comparison. In addition, we observe a similar trend for other datasets (e.g.,CIFAR-10 and ImageNet-16-120) and networks. In our work, we set the number of epochs for training ours to 18.

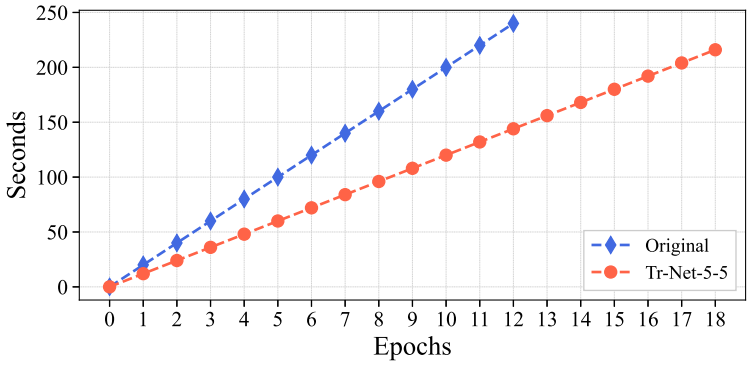


Figure 10: Training time comparison between the conventional method and proposed method on CIFAR-100.