Proxyless Neural Architecture Adaptation at Once

DO-GUK KIM AND HEUNG-CHANG LEE

1Department of Artificial Intelligence, Inha University, Incheon 22212, Republic of Korea
2AI Laboratory, Kakao Enterprise, Seongnam 13494, Republic of Korea

Corresponding author: Heung-Chang Lee (andrew.com@kakaoenterprise.com)

This work was supported in part by Kakao Enterprise and Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant through the Korea Government [Ministry of Science and ICT (MSIT)], Artificial Intelligence Convergence Research Center (Inha University), under Grant 2020-0-01389, and Artificial Intelligence Convergence Innovation Human Resources Development (Inha University) under Grant RS-2022-00155915.

ABSTRACT Recently, Neural Architecture Search (NAS) methods are introduced and show impressive performance on many benchmarks. Among those NAS studies, Neural Architecture Transformer (NAT) aims to adapt the given neural architecture to improve performance while maintaining computational costs. In the architecture adaptation task, we can utilize the known high-performance architectures, and the architecture adaptation results of NAT showed performance improvements on various architectures in their experiments. However, we verified that NAT lacks reproducibility through multiple trials of experiments. Moreover, it requires an additional architecture adaptation process before network weight training. In this paper, we propose proxyless neural architecture adaptation that is reproducible and efficient. The proposed method doesn’t need a proxy task for architecture adaptation. It directly improves the architecture during the conventional training process, and we can directly use the trained neural network. Moreover, the proposed method can be applied to both supervised learning and self-supervised learning. The proposed method shows stable performance improvements on various architectures and various datasets. Extensive experiments on two benchmark datasets, i.e., CIFAR-10 and Tiny Imagenet, present that the proposed method definitely outperforms NAT and be applicable to various models and datasets.

INDEX TERMS Deep learning, automated machine learning, neural architecture search, self-supervised learning, architecture adaptation during training.

I. INTRODUCTION Traditionally, neural architectures are manually designed by human experts: e.g., VGG [1], ResNet [2], and DenseNet [3]. Choosing proper neural architecture for a given dataset and task is important for getting high performance. Recently, architectures selected by Neural Architecture Search (NAS) algorithms achieved state-of-the-art performances on various benchmark datasets [4], [5], [6]. Even though NAS methods show great performance on various tasks and datasets, their prohibitively high computational cost makes NAS methods hard to be applied practically.

To overcome this limitation, many recent works are proposed to reduce the computational costs of NAS while maintaining the advantage of NAS approaches. Most of these works focused on search space [4], search strategy [7], [8], and network performance evaluation [5], [8]. Conventional NAS methods search neural architectures from scratch on the given search space. However, if we utilize the known superior neural architectures and reduce the search space, we can search high-performing neural architectures with a fraction of the cost compared to searching from the scratch.

Neural Architecture Transformer (NAT) [9] is the work that aims to utilize the known superior neural architectures and adapt them to the given dataset and tasks. The authors attempted to improve the given architecture so that the performance of the network is improved while maintaining or reducing the computational costs. However, NAT requires a proxy task to adapt the given architecture in addition to the conventional network weight training stage. As claimed in [10], using a proxy task in the architecture searching phase can degrade the performance of the final trained model.
To ensure that the adapted architecture performs well in the final training stage, the architecture adaptation method should be *proxyless* (i.e., using the same tasks for the architecture adaptation stage and final training stage).

Also, NAT has several other limitations. First, the reproducibility of the algorithm is not verified since the authors reported only one result for each model. In Section IV-B, we analyze the result of the multiple trials of NAT, and the result shows that NAT lacks reproducibility for some kinds of architectures. Second, it can only transform the neural networks with identical cell architectures. Recent NAS works focus on searching macroblock-based architectures that have various cell architectures [10], [11]. However, architectures of these types are hard to be adapted by NAT.

NAT uses a two-layer Graph Convolutional Network (GCN) as a controller to train the architectural parameters. Two-layer GCN can exploit relations between nodes within a distance of two, which is sufficient for cell-based search. To adapt the entire architecture composed of various cell architectures, we need to consider relations between cells. Two-layer GCN cannot analyze relations between cells since the distance of two nodes is the maximum range of analysis. We need another form of controller to adapt the architectures with various cells like [10].

In this paper, we propose a proxyless neural architecture adaptation technique that consecutively adapts the given architecture and trains the weights of the adapted architecture. The proposed method is *proxyless* since the same tasks are used in both architecture adaptation and final training stages. Moreover, the proposed method consecutively processes the training of architecture parameters, adapting the given architecture, and training the weights of the final architecture to be prepared for inference. Therefore, the proposed method doesn’t require a separated architecture adaptation stage and network weight training stage and it can be easily combined with the existing deep neural network training processes. Especially, we examined the effectiveness of the proposed method not only for supervised learning but also self-supervised representation learning.

Our contributions are summarized as:

- We propose a proxyless neural architecture adaptation algorithm that consecutively adapts given neural architecture and trains the weights of the network.
- We carried out extensive experiments on both supervised learning and self-supervised learning, and the results show consistent performance improvements.
- The proposed method overcomes the limitation of NAT such as poor reproducibility and limited types of the base architecture.

**II. RELATED WORK**

**A. NEURAL ARCHITECTURE SEARCH**

After the NAS is introduced by [12], NAS methods with various kinds of search strategy have been proposed; reinforcement learning (RL)-based methods [4], [5], [13], [14], evolutionary algorithm (EA)-based methods [6], [15], [16], gradient-based methods [7], [10], [17], [18], [19], predictor-based methods [20], [21] and others [8], [22], [23], [24]. RL-based methods and EA-based methods solve architecture search problem which is inherently discrete by using reinforcement learning and evolutionary algorithm, respectively. In gradient-based methods, they convert the architecture search problem as a continuous and differentiable form and apply gradient-based optimization techniques to solve the problem. With the advancement of the search strategy, the weight sharing concept is proposed and it dramatically reduces the computational cost of NAS [5], [7], [10]. Also, some researchers studied training and utilizing predictors that can predict the performance of neural architecture [20], [21].

Though the majority of the NAS studies are focused on convolutional neural networks, recent NAS studies include architecture search for various types of architectures such as graph neural network [25] and Transformer [26]. Besides, [24] shows that NAS can be carried out without labels. The authors attempt to NAS with a hand-crafted self-supervised learning objective and the result shows that searched architectures show good performance for not only the hand-crafted self-supervised tasks but also typical supervised learning tasks. Even though the weight sharing approach dramatically reduces the computational cost of NAS [5], still a couple of GPU-days are required for the majority of NAS methods. Recent studies about zero-cost performance estimation proxies for NAS [27], [28], [29] make NAS can search architectures even in just a few seconds.

Meanwhile, NAT [9] proposed the architecture adaptation that optimizes given neural architecture while typical NAS methods search the architecture without any pre-defined base architecture. NAT aims to adapt the given architecture for the dataset and task so that the resulting architecture has better performance with fewer parameters than the original architecture. To achieve the goal, NAT only changes the original operations into none or identity operations. Although NAT showed impressive results in the paper, there are several limitations of the NAT as we claimed in Section 1.

**B. SELF-SUPERVISED REPRESENTATION LEARNING**

The proposed method can be applied to not only supervised learning but also self-supervised representation learning. Traditionally, self-supervised learning focuses on learning representations from unlabeled data and pre-defined hand-crafted tasks like image colorization [30], solving jigsaw puzzles [31], and rotation prediction [32]. Recently suggested contrastive learning enables learning representations without defining specific tasks. In contrastive learning, the neural network learns representations based on different views of given inputs that are produced by a set of augmentations. Especially, SimCLR [33] achieved remarkable performance on various benchmarks with the utilization of effective augmentations on contrastive loss.
C. DIFFERENCES BETWEEN EXISTING METHODS AND THE PROPOSED METHOD

We summarize the main differences between existing methods and the proposed method in this subsection.

The proposed method enhances the given architecture while most existing NAS methods [7], [10], [19] search architecture from scratch in a given search space. The objective of the majority of NAS methods is finding the optimal architecture in the given search space. The objective of the proposed method is to adapt the given base architecture and to find enhanced final architecture. Therefore, the proposed method is an improved technique from NAT and is different from typical NAS methods.

In comparison with NAT, the proposed method uses the same tasks in both architecture adaptation and network weight training while NAT has a proxy task for architecture adaptation. Gradient-based optimization is used to adapt architectures in the proposed method and NAT uses an RL-based search strategy. The proposed method consecutively adapts the given architecture and trains the weights of the adapted architecture while NAT [9] and most existing NAS methods have separated architecture searching phase and weight training phase.

III. METHODOLOGY

The proposed method adapts the given neural architecture and improves the performance by using gradient-based optimization. After the entire learning process, we directly get the trained network, and there is no need to train a new network from scratch. There are two consecutive stages in the proposed method: the architecture train stage and the network train stage. In the architecture train stage, both the architecture parameters and network parameters are trained. After the architecture train stage, only network parameters are trained. The overall processes of the proposed method for both supervised learning and self-supervised learning are shown in FIGURE 1.

A. DIFFERENTIABLE ARCHITECTURE PARAMETERS

To adapt the given neural architecture, we use differentiable architecture parameters and gradient-based learning. The architecture parameters $\theta$ is defined in the network architecture graph. Each edge in the network architecture graph contains original operation, identity operation, and none operation. Computation of each edge is carried out based on the architecture parameters:

$$o_e(x) = \theta_{e,\text{none}} \cdot Z + \theta_{e,\text{id}} \cdot x + \theta_{e,\text{same}} \cdot o_e(x),$$

where $x$ means input, $o_e(x)$ means output of the edge, $Z$ means zero tensor, $o_e(x)$ means original operation of the edge, $\theta_{e,\text{none}}$ means the weight of the none operation of the edge, $\theta_{e,\text{id}}$ means the weight of the identity operation of the edge, and $\theta_{e,\text{same}}$ means the weight of the original operation of the edge. We set initial $\theta_{\text{none}}$ and $\theta_{\text{id}}$ as zero, and $\theta_{\text{same}}$ as one. Therefore, the initialized network works the same as the original architecture.

The proposed method can be used for two different objectives; one is improving the cell architecture and the other...
is improving the entire network architecture. If we aim to adapt the cell architecture for the given dataset and task, then the architecture parameter $\theta$ is shared between all cells in the network. When the proposed method is used to adapt the entire network architecture, the $\theta$ is not shared and each layer has its own $\theta$. FIGURE 2 represents $\theta$ on two different objectives.

B. ARCHITECTURE TRAIN STAGE

After the network is initialized, the architecture train stage begins to adapt the neural architecture for the given dataset and task. In this stage, both the network weight parameters $\omega$ and the architecture parameters $\theta$ are trained alternately. For every input mini-batches, $\omega$ is trained first, and $\theta$ is trained after the update of $\omega$. Note that the proposed method doesn’t require any separated dataset for architecture optimization, and the network can utilize a full dataset to train its weights $\omega$. The architecture train stage is carried out for the pre-defined epochs.

When the architecture train stage is finished, architecture is transformed based on the trained $\theta$. For each edge, the operation that has the highest weight in $\theta$ is selected to construct the final architecture. In the example of FIGURE 1, two edges maintain the original operations, one edge changed its operation into identity, and one edge is removed because none operation is selected.

In the experiments, we set the number of architecture train stage epochs as 50. The results show consistent performance enhancement with the fixed value of the number of architecture train stage epochs. Nevertheless, the number of architecture training epochs can be optimized for a given dataset and given base architecture by hyperparameter optimization.

C. NETWORK TRAIN STAGE

The architecture parameters trained at the previous stage is fixed, and only $\omega$ of the network is trained in this stage. This stage is the same as the traditional neural network training process, and it is continuously carried out after the architecture train stage. At the end of this stage, we can get the trained network and use it to infer unseen input data or test the performance of the model.

Algorithm 1 describes the overall training process of the proposed method. First, $w$ and $\theta$ are initialized. During the architecture train stage, both $w$ and $\theta$ are trained using the same mini-batch. The value of $\theta$ is fixed after the architecture train stage, and only $w$ is trained for the remaining network train stage.

D. SUPERVISED LEARNING AND SELF-SUPERVISED LEARNING

The proposed method can be applied to both supervised learning and self-supervised learning. The objective of the proposed method can be formulated as:

\[
\min_{\theta} \mathcal{L}(w^*(\theta), \theta)
\]

s.t. $w^*(\theta) = \arg \min_w \mathcal{L}(w, \theta)$,

where $\theta$ denotes the architecture parameter, $w$ is the weight of the network, $\mathcal{L}(\cdot)$ is the loss function. Unlike $w$, $\theta$ does not apply a softmax while calculating the loss. The reason is that transitioning into the softmax eliminates the effect of being normalized. The magnitude between the updated value and the original value may be insufficient if the normalized value is reflected in $\theta$ using a small learning rate and SGD optimizer with cosine scheduler.

In case of the supervised learning, we used the cross-entropy (CE) loss which uses data and labels to calculate the loss. In self-supervised learning experiments, we utilized SimCLR [33] objective which uses two different augmented views of images to calculate the loss. The SimCLR objective can be formulated as:

\[
\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} 1_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)},
\]

where $z_i$, $z_j$ means positive pair which means originally same image but augmented differently, $\text{sim}(\cdot)$ is the cosine similarity, and $\tau$ is the temperature hyperparameter. Although we used the CE loss and SimCLR objective for supervised learning and self-supervised learning, the

![FIGURE 2. Two types of architectural parameters $\theta$. The proposed method can be used to adapt the cell architecture or to adapt the entire network architecture. $\theta$ is shared when the proposed method is used to adapt the cell architecture. In the case of the adaptation of the entire network architecture, each layer has its own $\theta$.](image-url)
Algorithm 1: Training Process of the Proxyless Neural Architecture Adaptation

**Input:** number of total training epochs $T$, number of architecture training epochs $P$, number of iteration per epoch $N$, architecture parameter $\theta$, network weight parameter $w$, train dataset $D_{\text{train}}$, loss function $L$

Initialize $w$

Initialize $\theta$ as $\theta_{\text{none}} = 0, \theta_{\text{id}} = 0, \theta_{\text{same}} = 1$

for $t = 1 \ldots P$ do

for $i = 1 \ldots N$ do

Get $i$-th mini-batch $m_i$ from $D_{\text{train}}$

Update $w$ by descending $\nabla w(L(m_i))$

Update $\theta$ by descending $\nabla \theta(L(m_i))$

Fix each element of $\theta$ as $1$ if it is the maximum value of the $\theta_e$, otherwise, fix as $0$.

for $t = P + 1 \ldots T$ do

for $i = 1 \ldots N$ do

Get $i$-th mini-batch $m_i$ from $D_{\text{train}}$

Update $w$ by descending $\nabla w(L(m_i))$

proposed method can be easily adapted to optimize other objectives. As described earlier, the network train stage of the proposed method is the same as the conventional deep neural network training process. Adding architecture parameters and an architecture train stage is all we need for using the proposed method for other datasets or tasks.

**IV. EXPERIMENTS**

We carried out extensive experiments to verify the performance and the reproducibility of comparison methods. NAS studies typically utilize NAS benchmarks [34], [35], [36] to evaluate various NAS methods at fair environments. In this paper, we aim to enhance the given architecture and we set comparison methods as the original architecture and NAT. We intend to compare the performances of various methods on common image classification benchmarks. Therefore, we tested various models on CIFAR-10 and Tiny Imagenet in the supervised learning experiments. In self-supervised learning experiments, various models are trained on CIFAR-10 to learn representations by contrastive learning with the SimCLR objective. Regarding the baseline, the performance of the original architecture is used as a baseline in the experiments following the experimental setting in [9]. Different from conventional NAS methods, the proposed method aims to enhance the performance of the given architecture, and performance gain between adapted architecture and the original architecture is important.

**A. DATA AND EXPERIMENTAL SETUP**

1) **SUPERVISED LEARNING**

CIFAR-10 dataset consists of 50,000 train images and 10,000 test images with ten classes. The size of images is $32 \times 32$, and images have RGB color channels. Tiny ImageNet dataset has 100,000 train images and 10,000 test images with 200 classes. The input size of Tiny Imagenet dataset is $64 \times 64$, and all images are RGB color images.

| Model   | Method    | Avg Acc (%) | Std (%) | Total Cost (GPU hours) |
|---------|-----------|-------------|---------|------------------------|
| ResNet20| Original  | 91.66       | 0.16    | 8                      |
|         | NAT       | 55.78       | 42.34   | 14                     |
|         | Ours(Cell)| 93.29       | 0.11    | 11.2                   |
|         | Ours(Full)| 93.12       | 0.12    | 8.7                    |
| MobileNet V2 | Original  | 93.91       | 0.12    | 20.5                   |
|          | NAT       | 91.97       | 5.10    | 27.8                   |
|          | Ours(Cell)| 95.02       | 0.31    | 24.8                   |
|          | Ours(Full)| 94.93       | 0.13    | 22.1                   |
| DARTS   | Original  | 96.75       | 0.11    | 38.3                   |
|         | NAT       | 96.95       | 0.09    | 47                     |
|         | Ours(Cell)| 96.97       | 0.15    | 45                     |
|         | Ours(Full)| 96.82       | 0.13    | 41.1                   |
| ProxylessNAS | Original | 94.19       | 1.08    | 15.3                   |
|           | NAT       | -           | -       | -                      |
|          | Ours(Full)| 95.09       | 0.23    | 19.8                   |

We experimented with various models on CIFAR-10 and Tiny Imagenet dataset. These models include ResNet20, MobileNet V2, DARTS, and ProxylessNAS. The former two models are manually designed, and the latter two models are NAS-based models. To compare the performance of NAT and our algorithm, we trained the NAT controller on each dataset and then trained the resulting architecture which is inferred from the controller. In the case of our algorithm, we test both cell-based architecture adaptation and entire network architecture adaptation. We used 0.025 learning rate, 600 epoch, and Stochastic Gradient Descent (SGD) optimizer with cosine scheduler as the same hyper-parameters to all models and methods. Exceptionally, we applied 300 epochs to MobileNet V2 and DARTS on Tiny Imagenet dataset, and utilized cut-out [37] for NAS models such as DARTS and ProxylessNAS.
TABLE 2. Reproducibility of original, NAT and Ours with different random seeds on CIFAR-10.

| Model       | Method       | Random Seed 1 | Random Seed 2 | Random Seed 3 | Random Seed 4 | Random Seed 5 |
|-------------|--------------|---------------|---------------|---------------|---------------|---------------|
| Resnet20    | Original     | 91.74         | 91.74         | 91.65         | 91.76         | 91.39         |
|             | NAT          | 10            | 75.14         | 91.05         | 10            | 92.68         |
|             | Ours(Cell)   | 93.21         | 93.34         | 93.4          | 93.15         | 93.37         |
|             | Ours(Pull)   | 93.07         | 93.26         | 93.22         | 93.06         | 92.97         |
| Mobilenet V2| Original     | 93.39         | 94.04         | 93.95         | 93.95         | 93.72         |
|             | NAT          | 83.36         | 95.13         | 94.95         | 91.18         | 95.21         |
|             | Ours(Cell)   | 94.57         | 94.97         | 95.41         | 94.97         | 95.18         |
|             | Ours(Pull)   | 95.13         | 94.81         | 94.85         | 95            | 94.85         |

FIGURE 3. Transformed Resnet20 cell architectures.

We tested five times with different random seeds for every experiment to get reliable evaluation results and verify the reproducibility of comparison algorithms [38], [39]. Therefore, we report the average accuracy and standard deviation of each method and each model in both CIFAR-10 results (TABLE 1) and Tiny Imagenet results (TABLE 3). The total cost of NAT was calculated by adding GPU hours of the architecture transformation stage and network train stage, and the cost of our algorithms was computed by just measuring the cost of the whole training process.

2) SELF-SUPERVISED LEARNING

We conduct experiments on CIFAR-10 dataset to compare performances of original architectures and architectures adapted by the proposed method. Self-supervised representation learning requires deeper architecture than that typically used in supervised learning. In other words, it necessitates larger parameters to be converged. So we employed ResNet50 instead of ResNet20, and we adopted 24 cells for DARTS architecture. As a result, the size of the parameters was 24.6±0.1M for self-supervised learning and the dimensions of the channels were adjusted to maintain the parameters with similar sizes. Regrading the self-supervised learning settings, we used a temperature of 0.5, k of 200, epochs of 500, and 0.001 learning rate with Adam optimizer from SimCLR [33] basic training settings. In all experiments, we maintained proxyless feature of the proposed method, and the proposed method process the architecture adaptation and network weight training all at once.

B. RESULTS ON SUPERVISED LEARNING

The results of TABLE 1 have average accuracy, standard deviation, and total cost by various methods with different models on CIFAR-10 dataset. We trained and inferred five times to get average accuracy and standard deviation. As shown in TABLE 1, the results of NAT is unstable in the case of manually designed models. The results of our algorithms have better average accuracy and standard deviation than original and NAT in all cases. Moreover, the total computational cost is lower than NAT. Note that NAT cannot transform the architecture of ProxylessNAS, since it has various cell architectures in the network. However, the proposed method successfully improves the performance of ProxylessNAS architecture.

TABLE 2 shows the reproducibility of various methods with different random seeds on CIFAR-10 dataset. In NAT [9], there is only one result for each experiment. Therefore we experimented five times to get reliable evaluation results for all comparison methods.

In the case of Resnet20 experiments, the results of seed 1 and 4 of NAT is the performance of the random guesses. These results are caused by transforming identity edges into none operation. Transformed Resnet20 architectures of seed 1 are presented in FIGURE 3. Changed edges are notated as red colors. As shown in FIGURE 3(b), NAT
TABLE 3. Comparison of Average Accuracy, Standard Deviation and Total Cost between original, NAT and Ours on Tiny Imagenet.

| Model    | Method     | Avg Acc (%) | Std (%) | Total Cost (GPU hours) |
|----------|------------|-------------|---------|------------------------|
| Resnet20 | Original   | 50.72       | 0.41    | 16.1                   |
|          | Ours(Cell) | 52.86       | 0.49    | 22                     |
|          | Ours(Pull) | 52.71       | 0.23    | 17.5                   |
| Mobilenet V2 | Original | 51.57       | 0.76    | 20.3                   |
|          | Ours(Cell) | 53.17       | 1.03    | 25.5                   |
|          | Ours(Pull) | 52.92       | 0.50    | 23.7                   |
| DARTS    | Original   | 59.23       | 0.44    | 39.2                   |
|          | Ours(Cell) | 60.24       | 0.35    | 47                     |
|          | Ours(Pull) | 60.63       | 0.50    | 43.8                   |

TABLE 4. Comparison of Accuracy between original and Ours on Self-Supervised Learning.

| DataSet   | Model     | Method     | Accuracy (%) |
|-----------|-----------|------------|--------------|
| CIFAR-10  | Resnet50  | Original   | 82.39        |
|           | Ours      | 86.8       |
| DARTS     | Original  | 88.56      |
|           | Ours      | 89.04      |

FIGURE 6. Resnet20 cell transformed by our method on Tiny Imagenet.

FIGURE 7. MobileNetV2 cell transformed by our method on Tiny Imagenet.

FIGURE 8. DARTS V2 normal cell transformed by our method on Tiny Imagenet.

transformed all edges to node 5 into none operation. Therefore, zero tensors are passed to the next layer.

This low reproducibility of NAT is caused by the characteristic of the RL-based search strategy. Although it is a low possibility, a controller with the stochastic policy sometimes suggests final architecture with poor performance, even if it is properly trained. This phenomenon does not occur for gradient-based search strategies.

The proposed method shows consistent performance improvements on the results presented in TABLE 2. Like the final architecture example in FIGURE 3(a), the proposed method doesn’t change convolution operation in the case of Resnet architecture.

Regarding the result of Mobilenet V2 experiments, the performance of NAT is degraded when it transforms convolution operation into identity operation. FIGURE 4 shows the transformed Mobilenet V2 architectures of seed 1. Additionally, we represent the transformed DARTS normal cell architecture of our algorithm in FIGURE 5. There is no change in the edges of the reduction cell.

TABLE 3 shows the results of various methods with different models on Tiny Imagenet dataset. The results of this table describe that both our algorithms have better average accuracy and standard deviation than the original method upon all models. FIGURE 6, 7, 8 describes final architectures found by the proposed method on Tiny Imagenet.

C. RESULTS ON SELF-SUPERVISED LEARNING

The results reported in TABLE 4 are evaluated by k-nearest neighbor (KNN) during the representation learning. Therefore, it may not be better than the performance of linear evaluation reported in [33] but we evaluated all architectures in fair conditions. According to the results of TABLE 4, the proposed method outperforms the basic architecture of ResNet50 and DARTS even with similar parameter sizes.

FIGURE 9 depicts the architecture modified by the proposed method of ResNet50. As shown the figure, all none operations are changed to identity operations. It was determined that connections of ResNet50 were insufficient to converge self-supervised learning. FIGURE 10 is an architecture that transforms DARTS using our method. Normal cell is not changed and two connections in the reduction cell are
changed to none connections. Two changed operations are originally skip-connections. Normal cells compose most of the parameters of the entire network. Therefore, it appears that the architecture of DARTS is minimally modified for self-supervised learning by changing only two operations in the reduction cell. It is a known fact that self-supervised learning starves a lot of parameters than supervised learning. And this difference between supervised learning and self-supervised learning makes differences in final architectures.

**TABLE 5.** Comparison of Average Accuracy, Model Parameter Size, and FLOPs between original, NAT, and Ours on CIFAR-10 with random seed 1.

| Model   | Method  | Avg Acc (%) | Param (M) | FLOPs (M) |
|---------|---------|-------------|-----------|-----------|
| Resnet20| Original| 91.74       | 3.80      | 552       |
|         | NAT     | 93.21       | 3.80      | 552       |
|         | Ours(Cell)| 94.57       | 3.65      | 593       |
| Mobilenet V2| Original| 93.9       | 3.62      | 590       |
|         | NAT     | 83.36       | 0.6       | 103       |
|         | Ours(Cell)| 96.72       | 3.35      | 539       |
|         | Ours(Cell)| 96.96       | 2.68      | 432       |

**V. DISCUSSION**

The objective of the proposed method is to enhance the performance of the given neural architecture while maintaining or reducing the computational costs by architecture adaptation. To analyze the computational costs of resulting architectures, the number of parameters and FLOPs are presented in **TABLE 5**. The proposed method maintains or reduces the value of FLOPs in the case of Resnet20 and DARTS. The value of FLOPs is slightly increased in Mobilenet V2, however, the increment is only 0.05% of the original FLOPs. The accuracy of the resulting architectures of the proposed method is enhanced in all three architectures, while NAT failed to enhance the accuracy due to the changing essential operations in the cell architecture.

The architecture parameter $\theta$ in the proposed method is not large in the case of the cell-based architecture adaptation. As presented in the results, optimization of $\theta$ with just first-order information show consistent performance estimation in various seed architectures, various datasets, and even in both supervised learning task and self-supervised learning task. Nevertheless, optimization of $\theta$ using second-order information [7] or neuroevolution [40] can be further improving the result of the proposed method. We plan to adapt these optimization techniques to the proposed method for future work.

**VI. CONCLUSION**

We proposed a novel gradient-based neural architecture adaptation algorithm that is reproducible and effective for architecture improvement. Thanks to the differentiable architecture parameters, our algorithm can train both the architecture parameters and the network weight parameters at once. Thus, the proposed method can easily be combined with the conventional neural network training process. Rather than using an RL-based controller with stochastic policies, the proposed method uses architecture parameters for each operation and avoids performance degradation caused by changing important operations into none operations.

The results of the experiments demonstrate that the proposed algorithm has high reproducibility and stably improves
the performance of various models on various datasets. The proposed method can improve the performance of both manually designed architectures and NAS-based complex architectures. Moreover, the proposed method can be applied to both supervised learning and self-supervised learning and achieve performance improvement on both learning schemes.

**REFERENCES**

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. ICLR*, 2015.

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

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

[4] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proc. IEEE Conf. CVPR*, Jun. 2018, pp. 8697–8710.

[5] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing,” in *Proc. ICML*, 2018.

[6] E. Real, A. Aggarwal, G. Huang, and Q. V. Le, “Regularization of evolution for image classifier architecture search,” in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[7] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in *Proc. ICLR*, 2019.

[8] C. Zhang, M. Ren, and R. Urtasun, “Graph HyperNetworks for neural architecture search.” 2018, arXiv:1810.05749.

[9] Y. Guo, Y. Zheng, M. Tan, Q. Chen, J. Chen, P. Zhao, and J. Huang, “NAT: Neural architecture transformer for accurate and compact architectures,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 735–747.

[10] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct neural architecture search on target task and hardware,” in *Proc. ICLR*, 2019.

[11] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, “Single-path NAS: Designing hardware-efficient convo- lutional networks in less than 4 hours,” in *Proc. ECML PKDD*, 2019.

[12] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in *Proc. ICLR*, 2017.

[13] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” in *Proc. Int. Conf. Learn. Represent.*, 2017.

[14] H.-C. Lee, D.-G. Kim, and B. Han, “Efficient decoupled neural architecture search by structure and operation sampling,” in *Proc. IEEE ICASSP*, May 2020, pp. 4222–4226.

[15] L. Xie and A. Yuille, “Genetic CNN,” in *Proc. IEEE Int. Conf. Comput. Vis.*, Jun. 2017, pp. 1379–1388.

[16] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, “Large-scale evolution of image classifiers,” in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[17] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, “Neural architecture optimization,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7816–7827.

[18] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, “PC-DARTS: Partial channel connections for memory-efficient architecture search,” in *Proc. Int. Conf. Learn. Represent.*, 2020.

[19] X. Dong and Y. Yang, “Searching for a robust neural architecture in four GPU hours,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Sep. 2019, pp. 1761–1770.

[20] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, “Neural predictor for neural architecture search,” in *Proc. Eur. Conf. Comput. Vis.*, Cham, Switzerland: Springer, 2020, pp. 660–676.

[21] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, “BRP-NAS: Prediction-based NAS using GCNs,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10480–10490.

[22] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “SMASH: One-shot model architecture search through HyperNetworks,” 2017, arXiv:1708.05344.

[23] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, “Understanding and simplifying one-shot architecture search,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 549–558.

[24] C. Liu, P. Dollár, K. He, R. Girshick, A. Yuille, and S. Xie, “Are labels necessary for neural architecture search?” in *Proc. Eur. Conf. Comput. Vis.*, Cham, Switzerland: Springer, 2020, pp. 789–815.

[25] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, “Graph neural architecture search,” in *Proc. IJCAI*, vol. 20, 2020, pp. 1403–1409.

[26] M. Chen, H. Peng, J. Fu, and H. Ling, “AutoFormer: Searching transformers for visual recognition,” in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Jan. 2021, pp. 12270–12280.

[27] J. Mellor, J. Turner, A. Starkey, and E. J. Crowley, “Neural architecture search without training,” in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 7588–7598.

[28] M. S. Abdelfattah, A. Mehrortra, L. Dudziak, and N. D. Lane, “Zero-cost proxies for lightweight NAS,” in *Proc. Int. Conf. Learn. Represent.*, 2020.

[29] J. Xu, L. Zhao, J. Lin, R. Gao, X. Sun, and H. Yang, “KNAS: Green neural architecture search,” in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 11613–11625.

[30] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization,” in *Proc. Eur. Conf. Comput. Vis.*, Cham, Switzerland: Springer, 2016, pp. 649–666.

[31] M. Noroozi and P. Favaro, “Unsupervised learning of visual representations by solving jigsaw puzzles,” in *Proc. ECCV*. Cham, Switzerland: Springer, 2016, pp. 1021–1035.

[32] N. Komodakis and S. Gidaris, “Unsupervised representation learning by predicting image rotations,” in *Proc. Int. Conf. Learn. Represent.*, 2018.

[33] T. Chen, S. Kornblith, M. Noroozi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1597–1607.

[34] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, “NAS-Bench-101: Toward reproducible neural architecture search,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.

[35] X. Dong and Y. Yang, “NAS-Bench-201: Extending the scope of reproducible neural architecture search,” in *Proc. Int. Conf. Learn. Represent.*, 2019.

[36] X. Dong, L. Liu, K. Musial, and B. Gabrys, “NATS-Bench: Benchmarking NAS algorithms for architecture topology and size,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3634–3646, Jul. 2021.

[37] T. DeVries and G. W. Taylor, “Improved regularization of convolutional neural networks with cutout,” 2017, arXiv:1708.04552.

[38] M. Lindauer and F. Hutter, “Best practices for scientific research on neural architecture search,” *J. Mach. Learn. Res.*, vol. 21, no. 243, pp. 1–18, 2020.

[39] X. Dong, D. J. Kedziora, K. Musial, and B. Gabrys, “Automated deep learning: Neural architecture search is not the end,” 2021, arXiv:2112.09245.

[40] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, “Designing neural networks through neuroevolution,” *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, Jan. 2019.