Received 14 December 2023, accepted 27 December 2023, date of publication 8 January 2024, date of current version 23 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3350732

RESEARCH ARTICLE

Method for Expanding Search Space With Hybrid Operations in DynamicNAS

IKSOO SHIN[1,2], CHANGSIK CHO[1], AND SEON-TAE KIM[1,2], (Member, IEEE)
[1]Artificial Intelligence Research Laboratory, Future Computing Research Division, Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea
[2]Department of ICT, University of Science and Technology, Daejeon 34113, South Korea

Corresponding author: Seon-Tae Kim (stkim10@etri.re.kr)

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] (No. 2021-0-00766, Development of Integrated Development Framework that supports Automatic Neural Network Generation and Deployment optimized for Runtime Environment).

ABSTRACT Recently, a novel neural architecture search method, which is referred to as DynamicNAS (Dynamic Neural Architecture Search) in this paper, has shown great potential. Not only can various sizes of models be trained with a single training session through DynamicNAS, but the subnets trained by DynamicNAS show improved performance compared to the subnets trained by conventional methods. Although DynamicNAS has many strengths compared to conventional NAS, it has the drawback that different types of operations cannot be used simultaneously within a layer as a search space. In this paper, we present a method that allows DynamicNAS to use different types of operations in a layer as a search space, without undermining the benefits of DynamicNAS, such as one-time training and superior subnet performance. Our experiments show that common operation mixing methods, such as convex combination and set sampling, are inadequate for the problem, although they have a structure that is similar to the proposed method. The proposed method finds, from a supernet of hybrid operations, a superior architecture that cannot be found from a single-operation supernet.

INDEX TERMS Neural architecture search, NAS, weight-sharing NAS, DynamicNAS, convolutional neural network, vision transformer.

I. INTRODUCTION

The design of model architecture plays a pivotal role in the success of deep learning across various tasks, including image classification [1], speech recognition [2], and natural language processing [3]. Not only in these traditional fields, but also in practical domains such as point cloud [4] and coal mining [5], the impact of architecture design demonstrated recently. However, designing architectures for the domains is not an easy task and requires a time-consuming and laborious process, as each design's performance needs to be tested individually. Therefore, researchers have shifted their attention to Neural Architecture Search (NAS) to automate and improve the process of architecture design [6], [7], [8], [9], [10].

NAS has emerged as a powerful tool for discovering neural network architectures that were previously unknown to researchers [11], [12], [13]. Recent advances [14], [15], [16], [17] in the field of NAS have introduced a novel approach to building efficient neural networks. This method, referred to as weight-sharing NAS, has been successfully employed in models such as Once-For-All [14], AttentiveNAS [15], NASViT [16] and Autoformer [17]. However, it is important to note that there is a fundamental difference in the structural nature of these models compared to conventional weight-sharing NAS [7], [18], [19], [20], [21]. In weight-sharing NAS, also known as one-shot NAS, candidate operations are simultaneously employed in a single layer of a large network or supernet. The supernet consists of every subnet, with no sharing of parameters between the candidate operations. On the other hand, the novel NAS approach shares weight parameters between the different candidate operations, thereby increasing the level of weight sharing as compared with conventional weight-sharing NAS, which shares weight parameters only at the supernet level. Thus,

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

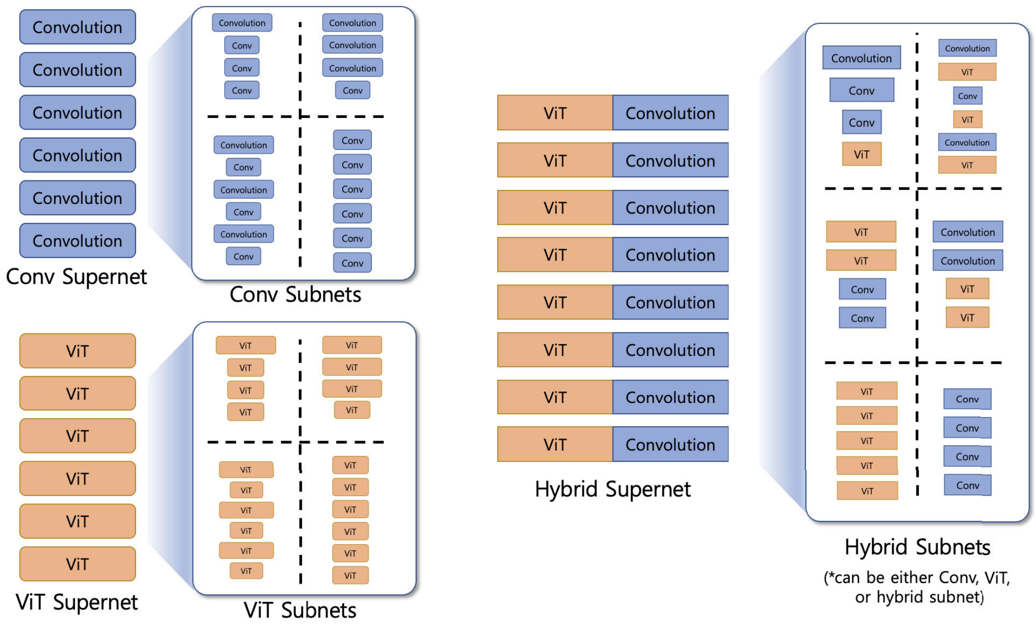

FIGURE 1. Illustration of the concept of this paper. Various scales of subnet can be sampled from a supernet trained by DynamicNAS. However, there is no option to select an operation due to the intrinsic nature of DynamicNAS, which forces one to select an operation manually. Our method gives this option to DynamicNAS. (Conv: Convolution, ViT: Vision Transformer).

we refer to the novel NAS approach as DynamicNAS in this paper.

From the previous works regarding DynamicNAS [14], [15], [16], [17], it is worth taking note that previous researches have not considered using different kinds of scalable operations as a search space within a layer. This stems from the structural nature of DynamicNAS, which shares weight parameters among candidate operations. As a result, its ability to explore a wide range of architectures is limited. In response to this, we propose a novel method to introduce more flexibility into the search space of DynamicNAS, allowing for the use of different kinds of operations within a layer. Figure1 illustrates the concept of the paper. In this paper, we demonstrate the effectiveness of the proposed method in expanding the search space of DynamicNAS compared to other naive methods and highlights the potential benefits of incorporating various types of operations within a layer.

The major contributions of our work can be summarized as follows:

- We propose a method that provides DynamicNAS an ability to choose an operation within a layer, while retaining the strengths of the DynamicNAS approach. Our approach resembles the one used in ProxylessNAS but differs in its practical implementation.
- In our method, we prevent interference that could occur between candidate operations and the impact of parameters for operation selection on candidate operations. This method can also be widely applied to other NAS methods.
- Our method does not require additional agents, which are typically used in NAS. Additionally, our method does not require additional training stages or epochs.
- In experiments with our method, we were able to find architectures that are superior to those extracted from a conventional single-operation supernet.
- We present experimental results to analyze the process of choosing preferred operations during supernet training and show that the process can dramatically change depending on the design of the search space. However, our method shows robustness to the change.

The contents of this paper are as follows. In the following section, we present the conventional works related to this study. In Section III, we briefly review the structure of DynamicNAS supernet. In Section IV, we introduce our method to address the problem presented above. In Section V, we present the result of experiments in which our method was used. In Section VI, we discuss the meaning of our experiments and future work. Finally, we conclude the presentation of our method in Section VII.

II. RELATED WORKS

This work is about how to expand the search space of DynamicNAS. The concept of DynamicNAS is based on SlimmableNet [22], which is a Convolutional Neural Network (CNN) that first applied a scalable width structure. The authors of Once-For-All [14] further developed this concept and increased the number of types of scalable structure, including depth, width, kernel size, and resolution, in CNN. While the concept of DynamicNAS was initially applied to CNNs, it has also been extended to other architectures such as Vision Transformer (ViT) [17]. The authors of NASViT [16] proposed a CNN-ViT hybrid network, but it is important to note that the choice of operation (CNN or ViT) for each layer in NASViT was determined manually by the authors.

Recent studies have focused on improving the performance of the final architecture of DynamicNAS. The subnet sampling method has been considerably addressed in AttentiveNAS [15] to achieve better results. Similarly, the authors of FocusFormer [23] have also concentrated on a method using a specialized architecture sampler instead of a uniform sampler to sample subnet architectures for each training step. On the other hand, the authors of PreNAS [24] proposed a different approach. They utilize a zero-cost proxy to reduce the search space before executing the main session and concentrate on training subents included in a smaller preferred search space. It is worth noting that PreNAS and our work take opposite directions. Where PreNAS aims to shrink the search space for better performance, our work expands the search space to explore a greater variety of architectures.

III. PRELIMINARIES

In this section, we briefly present the structure of a layer of DynamicNAS supernet, which will be used in the subsequent parts of this paper. A detailed explanation of the structure is presented in Appendix A with examples.

The lth layer of the DynamicNAS supernet can be represented as follows:

$$X_l = F_1(X_{l-1}) + F_2(X_{l-1}) + F_3(X_{l-1}) + \cdots$$

Here, X_l represents the output of the lth layer. $F_i(\cdot)$ represents the ith candidate operation of the lth layer. The structure of the layer can be changed based on the decision of whether to use each operation. There is a difference from weight-sharing NAS, where all operations can work independently. In DynamicNAS, only the first operation $F_1(\cdot)$ can work independently. If we want to use $F_{i(i\neq 1)}(\cdot)$, we must also use $F_{i-1}(\cdot)$ together with it. For example, let's suppose that $F_1(\cdot)$ is 3×3 convolution and $F_2(\cdot)$ is surrounding operations of 5×5, except the core part (i.e., 3×3) of 5×5 convolution. $F_2(\cdot)$ is considered a strange operation that is not commonly used when used standalone. However, when combined with $F_1(\cdot)$, the sum of $F_1(\cdot)$ and $F_2(\cdot)$ results in a 5×5 convolution, which is a commonly used operation in CNNs.

In this study, if an operation can be obtained by summing extra terms to another operation, we consider them homogeneous and they can be entangled through summation. However, if one operation cannot be obtained solely by summing additional terms to another operation, we consider them heterogeneous and they cannot be entangled through summation. For example, 3×3 convolution and 5×5 convolution are homogeneous as 5×5 convolution can be made only by summing extra terms to 3×3 convolution. On the other hand, 3×3 convolution and multilayer perceptron are heterogeneous as one operation cannot be made only by summing extra terms to the another operation. To summarize, DynamicNAS combines candidate operations by allowing them to share weight parameters. As a result, only homogeneous operations can be used in DynamicNAS as a search space.

IV. METHODOLOGY

This section describes our approach to using heterogeneous operations as a search space within a layer of the DynamicNAS supernet.

A. METHODOLOGY CONSIDERATIONS

Before we present our method, we emphasize the criteria we have considered while developing the method. We established two main criteria for our proposed method. The first one is that the method must maintain the benefit of DynamicNAS, namely, that it requires only one training stage over all stages until implementation. The second one is that it must be capable of identifying a better architecture than the single-operation supernet can identify. We utilize heterogeneous operations concurrently to expand the search space. Thus, we argue that at least it should find the same architecture as the architecture found in a single-operation supernet. If the proposed method does not meet both criteria, our approach may be redundant. For a test, we consider a Conv and a ViT block as heterogeneous candidate operations, which are commonly used in vision models.

A Conv block cannot be entangled with a ViT block, so we first considered applying the operation-mixing approaches of weight-sharing NAS, although the approaches do not entangle Conv and ViT blocks. In the previous section, we presented the concept of two primary weight-sharing NAS methods, namely the convex combination, and the set sampling methods, which have been widely used in recent NAS studies. The convex combination method, which combines a Conv block and a ViT block, can be represented as:

$$X_l = \alpha Conv(X_{l-1}) + \beta ViT(X_{l-1}), \tag{1}$$

where α and β are trainable parameters that control the contribution of the two blocks, and they are subject to the constraints $\alpha, \beta \in (0, 1)$ and $\alpha + \beta = 1$. Likewise, the set sampling method also combines a Conv block and a ViT block, but in a discrete manner:

$$X_l = \begin{cases} Conv(X_{l-1}) \\ ViT(X_{l-1}), \end{cases} \tag{2}$$

where each operation is randomly sampled following the uniform probability distribution for each step.

These methods were considered and tested as possible ways to address our problem. However, as expected, our experimental evaluation results showed that these methods may be inadequate for effectively exploring the architecture space while maintaining the advantages of DynamicNAS. We found that neither of these methods identified superior architectures that could outperform the single-operation supernet. This suggests that more sophisticated and efficient weight-sharing NAS methods may be needed to achieve better performance and generalization. The evaluation results of these methods will be presented in Section V.

B. PROPOSED METHOD

As a solution to give DynamicNAS an option to choose the operation, we propose a method that combines the advantages of two operation-mixing methods - convex combination and set sampling - to address their individual limitations. On the one hand, convex combination can update the importance of each operation during the training stage, as will be seen in Section V, it does not converge completely toward a preferred operation. On the other hand, the set sampling method restricts the number of operations for an inference step to one but does not update the importance of each operation. Consequently, the set sampling method shows poor performance. Thus, we present a unified solution that utilizes the strengths of both of these methods.

Our method, which combines both methods, has a structure such that:

$$X_l = S(\alpha + \alpha')Conv(X_{l-1}) + (1 - S)(\beta + \beta')ViT(X_{l-1}), \tag{3}$$

where S is a stochastic binary switch that selects between two candidate operations for each step. S is generated from a Bernoulli distribution with a parameter value of α. The trainable parameters α and β satisfy the constraint $\alpha + \beta = 1$. Meanwhile, the variables α' and β' are not trainable and are defined for each step, such that $\alpha + \alpha'$ and $\beta + \beta'$ are equal to 1. The sampling probability of each operation, that is, α and β, is included in the model structure to be updated during the searching stage with the weight parameters.

Eq. (3) can be rewritten as:

$$X_l = \begin{cases} (\alpha + \alpha')\mathrm{Conv}(X_{l-1}), & \text{if } S = 1 \\ (\beta + \beta')\mathrm{ViT}(X_{l-1}), & \text{if } S = 0, \end{cases}$$
$$\text{where } S \sim \mathrm{Bernoulli}(\alpha) \tag{4}$$

Eq. (4) shows that, more clearly, either Conv or ViT is selected as a candidate operation for each step based on a sample with a probability of α from the Bernoulli distribution. The output of (4) is either $Conv(X_{l-1})$ or $ViT(X_{l-1})$, which is the same as the output of the set sampling method. This is due to constraints $\alpha + \alpha' = 1$, $\beta + \beta' = 1$, Practically, our method works in the same manner as Algorithm 1 in the supernet training stage. The process from 10 to 14 of Algorithm 1 is the part especially added for our method.

C. METHOD ANALYSIS

We demonstrate the effectiveness of our method through experiments and show that it outperforms both the convex combination and the set sampling methods individually in Section V. Prior to conducting the experiments, we conduct an analysis by comparing the operational output of a layer and the gradient of the loss function with respect to α to observe the differences between the proposed method, the convex combination, and the set sampling during the forward and backward processes.

In the forward process of our method, the output of each step, layer l, and its corresponding expected value can be described as follows:

Algorithm 1 Proposed Method

```
1:  Initialize weight parameters of supernet
2:  Initialize trainable variables $\{a_i, b_i\}_{i=1}^{L}$ with zeros
3:  for $epoch = 1, 2, \ldots, E$ do
4:      for $b = 1, 2, \ldots, B$ do
5:          $X_0 = Stem(input_b)$
6:          $l_b \sim U(L_{min}, L)$
7:          for $l = 1, 2, \ldots, l_b$ do
8:              The scale of $Conv_l \sim U$
9:              The scale of $ViT_l \sim U$
10:             $\alpha, \beta = softmax([a_l, b_l])$
11:             $S \sim Bernoulli(\alpha)$
12:             $\alpha' = 1 - \alpha.detach()$
13:             $\beta' = 1 - \beta.detach()$
14:             $X_l = S(\alpha + \alpha')Conv_l(X_{l-1}) + (1 - S)(\beta + \beta')ViT_l(X_{l-1})$
15:         end for
16:         $output = Head(X_{l_b})$
17:         $loss = criteria(output, label)$
18:         $loss.backward()$
19:         $optimizer.step()$
20:     end for
21: end for
```

$$X_l = SConv(X_{l-1}) + (1 - S)ViT(X_{l-1})$$
$$= \begin{cases} Conv(X_{l-1}), & \text{if } S = 1 \\ ViT(X_{l-1}), & \text{if } S = 0 \end{cases} \tag{5}$$
$$E[X_l] = \alpha Conv(X_{l-1}) + (1 - \alpha)ViT(X_{l-1}) \tag{6}$$

Eq. (4) is rewritten as (5) by substituting $\alpha + \alpha' = 1$, $\beta + \beta' = 1$. As mentioned earlier, (4) and (5) show that the forward process of our method works the same as the forward process of the set sampling method, which is that only one of the candidate operations is selected for each step. The distinction between the two methods lies in the variations in their respective sampling probability distributions. In our method, an operation is selected according to the probability distribution α for each step, which is different from the uniform distribution of the set sampling method. Thus, the expected value of X_l is equal to (6). This aligns with the expected value obtained from the convex combination method. In summary, each step of our method works the same as the steps of the set sampling method, and the expected value of our method works the same as the expected value of the convex combination method.

In the backward analysis, we investigate the impact of operation importance weight α during supernet training. To do this, we examine the gradient of the loss function $\mathcal{L}$ with respect to the parameter α in the backward phase of the optimization process. Specifically, we need to compute $\frac{\partial F(X_{l-1})}{\partial \alpha}$, which represents the effect of α on the output of an operation. There is no mechanism for updating the operation importance in the set sampling method, so we compare our method specifically with the convex combination method during the backward analysis. In contrast to the backward process of the convex combination method, the backward process of our method exhibits slight variations.

To derive $\frac{\partial F(X_{l-1})}{\partial \alpha}$, we begin with:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \frac{\partial \mathcal{L}}{\partial X_l}\frac{\partial X_l}{\partial \alpha} = \frac{\partial \mathcal{L}}{\partial F(X_{l-1})}\frac{\partial F(X_{l-1})}{\partial \alpha} \tag{7}$$

In our method, $\frac{\partial F(X_{l-1})}{\partial \alpha}$ can be represented as:

$$\begin{aligned}\frac{\partial}{\partial \alpha}F(X_{l-1}) &= \frac{\partial}{\partial \alpha}((\alpha + (1-\alpha))F(X_{l-1})) \\ &= \frac{\partial}{\partial \alpha}\alpha F(X_{l-1}) + \frac{\partial}{\partial \alpha}(1-\alpha)F(X_{l-1}) \\ &= \frac{\partial}{\partial \alpha}\alpha F(X_{l-1}) + \frac{\partial}{\partial \alpha}\alpha' F(X_{l-1}) \end{aligned} \tag{8}$$

Eq. (8) presents the principal idea behind our method. There's a difference between theoretical manner and practical manner. Theoretically, the output of (8) is 0, which implies that α will be unchanged during the training process. Practically, we defined $\alpha'(= 1-\alpha)$ as a non-trainable parameter to ensure that $\frac{\partial}{\partial \alpha}(1-\alpha)F(X_{l-1}) = 0$. By doing so, $\frac{\partial}{\partial \alpha}F(X_{l-1})$ becomes equal to $\frac{\partial}{\partial \alpha}\alpha F(X_{l-1})$ during the training process, which allows us to train α.

To further examine this, let $g = \alpha + \Delta$ $(g, \Delta \in \mathbb{R})$, where Δ is a constant variable. Then, we observe that:

$$\frac{\partial}{\partial g}gF(X_{l-1}) = \frac{\partial}{\partial \alpha}\alpha F(X_{l-1}) = \frac{\partial}{\partial \alpha}gF(X_{l-1}) \tag{9}$$

Eq. (9) shows that the gradient of $(\alpha+\Delta)F(X_{l-1})$ with respect to the operation importance parameter α is calculated as the same as the gradient of the output where α were multiplied by the output of a selected operation. However, it is important to note that in our method, the actual output is not affected by g itself, because Δ is defined such that $\alpha + \Delta = 1$. This is in contrast to a method where only α is multiplied by the output of a selected operation. In summary, by defining α' as a non-trainable parameter, we are able to make α trainable in our method and still maintain the desired output.

In addition to performing an analysis of the forward process, we examine the gradient of the loss function X_l with respect to the operation importance parameter α for each step and the expected value of the gradient. We compare the results obtained from the convex combination method with those of our proposed method. In our method, the gradient and the expected value of the gradient are shown such as:

$$\frac{\partial X_l}{\partial \alpha} = \begin{cases} \frac{\partial}{\partial \alpha}\alpha F(X_{l-1}), & \text{if } S = 1 \\ \frac{\partial}{\partial \alpha}(1-\alpha)G(X_{l-1}), & \text{if } S = 0 \end{cases} \tag{10}$$

$$E[\frac{\partial X_l}{\partial \alpha}] = \alpha\frac{\partial}{\partial \alpha}\alpha F(X_{l-1}) + (1-\alpha)\frac{\partial}{\partial \alpha}(1-\alpha)G(X_{l-1}) \tag{11}$$

$$= \alpha F(X_{l-1}) - (1-\alpha)G(X_{l-1}) \tag{12}$$

In the convex combination method, the gradient and its expected value are shown such as:

$$\frac{\partial X_l}{\partial \alpha} = \frac{\partial}{\partial \alpha}(\alpha F(X_{l-1}) + (1-\alpha)G(X_{l-1})) \tag{13}$$

$$= F(X_{l-1}) - G(X_{l-1}) \tag{14}$$

$$E[\frac{\partial X_l}{\partial \alpha}] = F(X_{l-1}) - G(X_{l-1}) \quad (= \frac{\partial X_l}{\partial \alpha}) \tag{15}$$

The key difference between the two methods lies in the expected value of the gradient. Our analysis of the conventional convex combination method showed that its expected value is given by $F(X_{l-1}) - G(X_{l-1})$. On the other hand, the expected value of the proposed method is given by $\alpha F(X_{l-1}) - (1-\alpha)G(X_{l-1})$. In the conventional method, the gradient is always affected by the value of $G(X_{l-1})$, regardless of the importance weight α. In contrast, in our proposed method, the effect of $G(X_{l-1})$ on the gradient decreases as the importance weight α increases.

These findings suggest that our proposed method has a distinct advantage over the conventional method in that it allows us to control the influence of each operation on the gradient of the final output, depending on the value of α. This feature may be particularly useful when one operation dominates another, as we can adjust the value of α to ensure that both operations contribute equally to the final output.

Overall, our findings provide new insight into the behavior of NAS when a convex combination method is used, and they highlight the potential benefits of our proposed method to improve its performance.

D. COMPARISON WITH PREVIOUS WORK

We compared our proposed method with ProxylessNAS [21], which has a mechanism similar to ours. Although the overall approach of ProxylessNAS is similar to that of ours, there are some implementation differences that are worth noting. Basically, ProxylessNAS is a weight-sharing NAS method, whereas our method is based on DynamicNAS. Specifically, ProxylessNAS uses operation importance parameters, similar to our α, and binary gates, similar to our S, to evaluate the importance of each operation. ProxylessNAS selects one operation at a time from among the candidate operations of a supernet to update at each step, like ours. By doing this, the authors of ProxylessNAS intended to reduce the memory requirement of a supernet. However, the operation importance parameters in ProxylessNAS are not included in the model architecture. Therefore, the operation importance parameters cannot be updated during supernet training. Instead, they derive the gradient with respect to operation importance parameters through the following process [25]:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \alpha_i} &= \sum_{j=1}^{|\mathcal{O}|}\frac{\partial \mathcal{L}}{\partial p_j}\frac{\partial p_j}{\partial \alpha_i} \\ &\approx \sum_{j=1}^{|\mathcal{O}|}\frac{\partial \mathcal{L}}{\partial g_j}\frac{\partial p_j}{\partial \alpha_i}\end{aligned}$$

$$= \sum_{j=1}^{|\mathcal{O}|} \frac{\partial \mathcal{L}}{\partial g_j} \frac{\partial \frac{e^{\alpha_j}}{\sum_k e^{\alpha_k}}}{\partial \alpha_i}$$

$$= \sum_{j=1}^{|\mathcal{O}|} \frac{\partial \mathcal{L}}{\partial g_j} \frac{\sum_k e^{\alpha_k}(\mathbf{1}_{i=j} e^{\alpha_j}) - e^{\alpha_j} e^{\alpha_i}}{(\sum_k e^{\alpha_k})^2}$$

$$= \sum_{j=1}^{|\mathcal{O}|} \frac{\partial \mathcal{L}}{\partial g_j} p_j(\mathbf{1}_{i=j} - p_i)$$

where $|\mathcal{O}|$, g, and α represent the number of candidate operations in a layer, the binary gate, and the operation importance parameter, respectively.

To update the operation importance parameters, the authors of ProxylessNAS implemented an additional backward module in the backward process. Unlike the operation importance parameters of ProxylessNAS, the operation importance parameters of our method are included in the model structure, so there is no need to implement an additional backward module. ProxylessNAS and our proposed method have similarities, but the practical approach of our method is more straightforward as we do not need an additional backward module. In addition, our method simultaneously trains the weight parameters and the operation importance parameters, which is another difference between our method and ProxylessNAS.

V. EXPERIMENTS

In this section, we demonstrate the effectiveness of our method numerically. Specifically, we present the implementation details and performance test results on ImageNet [26]. We also analyze how the operation importance parameters converge and show that the convergence process of the operation importance parameters can change considerably depending on the search space.

A. IMPLEMENTATION DETAILS

The entire process for our experiments is identical to the Autoformer process [17]. As in [17], a two-stage search is used, which consists of supernet training and evolutionary search. The hyperparameters for supernet training and evolutionary search are also the same as those used for Autoformer.

1) MODEL ARCHITECTURE SPACE

As a baseline model, Autoformer-T supernet [17] is used for our experiments. To make a hybrid operation such as (3), a scalable Conv block of AttentiveNAS [15] is added to each layer of Autoformer-T. The search spaces for each operation are summarized in Table 1. When Conv is added to each layer, reshaping modules are included before and after Conv to fit the shape of the input of ViT into Conv. The shape of the input and output of ViT is (B, S, D), where B means batch size, S means the length of the sequence, and D means embedding dimension. The shape of the input and output of Conv is (B, H, W, C), where B means batch size, H means height, W means width, and C means channel. Thus, the shape (B, S, D) changes to $(B, \sqrt{S}, \sqrt{S}, D)$ before Conv and changes to (B, S, D) after Conv.

TABLE 1. Search space of Conv/ViT blocks.

	ViT Search Space			
ViT	Embed Dim	Q-K-V Dim	MLP Ratio	Head Num
Range	(192,240,24)*	(192,256,64)	(3.5, 4, 0.5)	(3, 4, 1)
	Conv Search Space			
Conv	Kernel	Channel	Expansion	-
Range	(3, 5, 2)	(192,240,24)	(3, 6, 3)	-
Depth	(12, 14, 1)			

*(Lowest, Highest, Step)

TABLE 2. Principal hyperparameters used in experiments.

Parameter Name	Value	Parameter Name	Value
Learning Rate(Max)	1e-3	Optimizer	AdamW
Learning Rate(Min)	1e-5	Repeated Aug	X
Learning Rate Scheduler	cosine	Weight Decay	0.05
Batch Size	1024	Label Smoothing	0.1
Total Epochs	500	Dropout Path	0.1
Warmup Epochs	20	Warmup LR(Min)	1e-6

It is important to note that batch normalization [27] is commonly incorporated into Conv operations to train neural networks. Batch normalization plays a crucial role in enhancing model performance. In DynamicNAS, the number of channels, and kernel sizes change at each step, which results in different statistical values for batch normalization at each step. This variability can adversely affect model performance. Some methods, such as those in [15] and [22], have been proposed to mitigate this. Our experiments employed the methodology proposed in [22]. Unlike [15], this approach does not require an additional statistical value training process for batch normalization, but still delivers desirable outcomes.

2) SUPERNET TRAINING

Supernet training works in the same manner as the process presented in Algorithm 1. As in [17], the size of the operation is chosen according to the uniform distribution. Only operation selection process is additionally added to the conventional process for each step. The principal hyperparameter for supernet training is summarized in Table 2.

3) EVOLUTIONARY SEARCH

The implementation of evolutionary search follows the same protocol as in [17], and [28]. The only difference is that the preferred operation for each layer is determined by the operation importance parameters. If the operation importance parameter of a layer is greater than 0.5, Conv is used for the layer. If the operation importance parameter of a layer is less than 0.5, ViT is used for the layer. The population size for the evolutionary searching is 50. The number of generations is set to 20. At each generation, we select the top 10 architectures. The mutation probabilities p_d and p_m are set to 0.2 and 0.4.

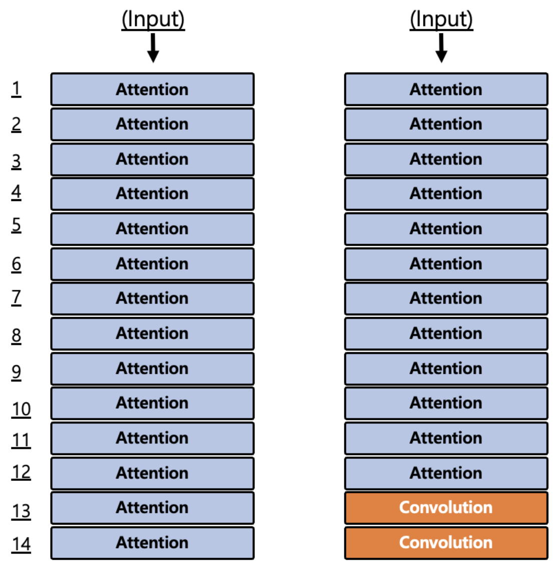

FIGURE 2. Autoformer-T model structure vs. the model structure found by our method.

4) PERFORMANCE TEST

To verify the performance of our algorithm, we test the performance of the models in each model size segment of the Autoformer-T supernet. In a supernet trained with DynamicNAS, multiple subnets of different sizes can be extracted. To evaluate the overall performance of the supernet trained by our method, we divided the subnets sampled from supernet into size intervals and used the performance of the model with the highest performance in each segment as the representative performance. The segments are divided into 2M (M stands for 1e6) intervals based on the number of parameters, and the model that performs the best is selected within the 0M-6M, 6M-8M, 8M-10M, and 10M-12M segments. All models are tested using PyTorch 1.8.1 on 4 Nvidia Tesla A100 GPUs.

5) DATASET

We use the ImageNet2012 dataset [26] for the experiments. ImageNet2012 is a benchmark dataset for image classification. It consists of a training set of about 1.2M and a validation set of 50,000 color images of 1,000 objects. The images vary in size, so we resize them to 224×224.

B. PERFORMANCE ON IMAGENET

Finally, we present the results of our method. The final model structure found by our method is presented in Figure 2. All layers except for the last two are determined to use ViT. The last two layers are determined to use Conv. A detailed analysis of the convergence process of the operation importance parameters is presented in the following subsection. The performances of each model are presented in Table 3. The models found by our method show superior performance across all ranges of model sizes. The minimum improvement was 0.09% and the maximum improvement was 0.28%.

TABLE 3. Evaluation of our method and classical operation-mixing methods on ImageNet.

	Model Size (Param)			
Method	~6M[†]	~8M	~10M	~12M
Autoformer(Acc)	74.9%(74.7%)[‡]	76.61%	76.75%	76.80%
Autoformer(Param)	5.96M(5.7M)	7.94M	9.38M	10.04M
Sample (Acc)	70.42%	72.40%	73.09%	73.07%
Sample (Param)	5.97M	7.65M	9.98M	11.90M
Convex (Acc)	60.03%	62.82%	63.32%	64.47%
Convex (Param)	6.00M	7.96M	9.96M	12.00M
Proposed (Acc)	**75.12%**	**76.70%**	**76.96%**	**77.08%**
Proposed (Param)	5.98M	7.72M	9.00M	11.02M

[†]M stands for 1e6

[‡]Result from our experiment (Result from original paper [17])

Although our method found an improved architecture compared with the conventional single-operation supernet, it does not necessarily imply that our approach can find the best architecture in the given search space. However, the results demonstrate that our method is capable of effectively leveraging the expanded search space. The subnets sampled from a supernet trained using the set sampling exhibit a performance degradation of around 4%. The set sampling method provides equal training opportunities to Conv and ViT in the training stage. A preferred operation is chosen during the searching stage. Consequently, the performance of every subnet was affected. We consider that the root cause of the performance degradation is due to providing equal training opportunities to Conv and ViT. In the case of the convex combination, a regularization term was used in addition to the loss function of the supernet to force the operation importance parameters to converge toward a preferred operation:

$$-\lambda \sum_{i=1}^{l}(\alpha_i - 0.5)^2,$$

where λ was set to 1e-3. Without the regularization term, the operation importance parameters failed to converge toward a preferred operation. It leads to the necessity of using both operations in conjunction. Although it was possible to select one operation based on the final operation importance parameters without the regularization term, then the subnets showed an accuracy lower than 10%. When the regularization term was used to encourage the convergence of operation importance parameters toward a single operation, the performance of the subnets still experienced a noticeable decrease even with this approach, as can be seen in Table 3. Despite the expanded search space, the convex combination and set sampling methods demonstrated limitations in effectively utilizing it.

Some competitive vision transformer models, which have a similar size to ours, are also compared in Table 4. DeiT [29], ConViT [30], TNT [31], and FocusFormer [23] are pure vision transformer architecture. LVT [32] is a hybrid architecture built of Conv and ViT blocks. Our model demonstrates an overall increased size but competitive performance compared to other models.

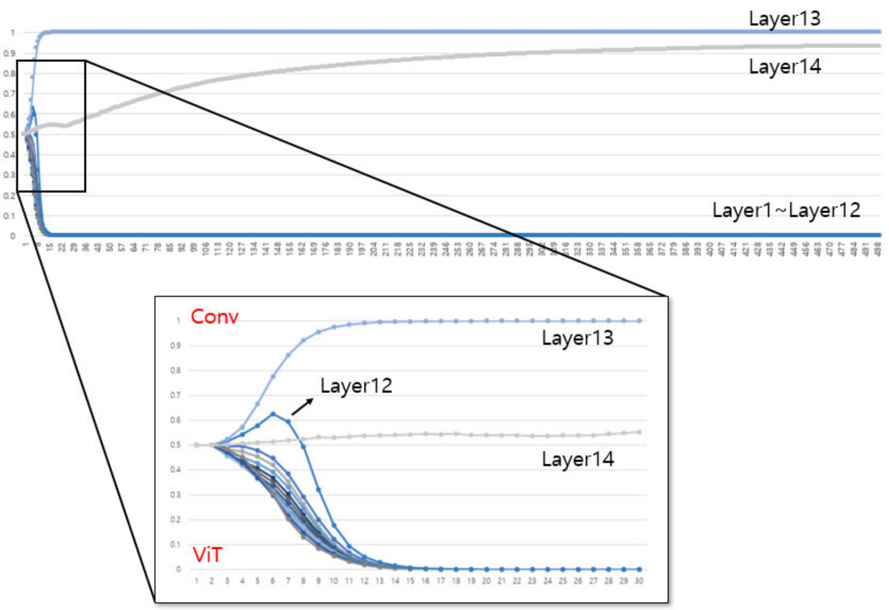

FIGURE 3. Graph of the convergence process of the operation importance parameters by epoch using our method.

TABLE 4. Performance comparison: proposed model vs. vision transformers with similar model sizes.

Model	Top-1(%)	Top-5(%)	#Params(M)
DeiT-Ti (2021)	72.2	91.1	5.7
ConViT (2021)	73.1	91.7	6.0
TNT-Ti (2021)	73.9	91.9	5.5
Autoformer-T (2021)	74.7	92.6	5.7
LVT† (2022)	74.8	92.6	5.5
FocusFormer (2022)	75.1	92.8	6.2
Proposed†	**75.1**	**92.7**	**6.0**

†hybrid architecture built of Conv and ViT blocks

C. OPERATION IMPORTANCE PARAMETER CONVERGENCE ANALYSIS

We also analyzed how the operation importance parameters converge. Table 6 and Figure 3 present the convergence process of the operation importance parameters by epochs. If the graph goes to 1, the preferred operation is Conv, and if the graph goes to 0, the preferred operation is ViT. We can see that the preferred operations of all layers except the 14th layer are determined before the 20th epoch. The operation importance parameter of the 14th layer converges to 1 in the latter part of the training. We focused on the behavior of the 12th layer, which came close to 1 and then changed direction after a few epochs.

We regarded the behavior of the 12th layer as a phenomenon that needs to be addressed. The candidate operation, which will eventually become the preferred operation, loses the opportunity to train when the operation importance parameter is poorly converged. Although this phenomenon should be thoroughly checked, we naively assumed that it happened because the 12th layer may be the last layer or may be the middle layer of the supernet depending on the choice of depth. To address this issue, we add a Conv-ViT block to

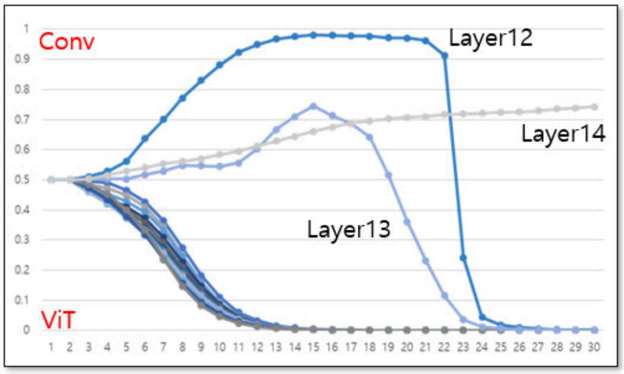

FIGURE 4. Graph of the convergence process of the operation importance parameters by epoch using our method after the last layer is fixed.

the Head layer of the supernet. We remove a block from the layers of the supernet to maintain the total number of layers. By doing this, we ensure that the layers used as the search space are always used as the middle layer.

Figure 5 and Table 5 show the result of the modification (the one referred to as Proposed Mod1 in Figure 6 and Table 5). The result shows that our assumption was wrong. The convergence process of the operation importance parameters of the 12th and 13th layers became more unstable. Nevertheless, the performance was slightly improved, even though the convergence process became more unstable. However, we observed that our method exhibited a novel characteristic, which is that the operation importance parameter could converge toward one side, even after it almost converged toward the other side. This may be another strength of our method.

In terms of convergence stability, we stabilized the convergence by utilizing a fixed last layer as described above,

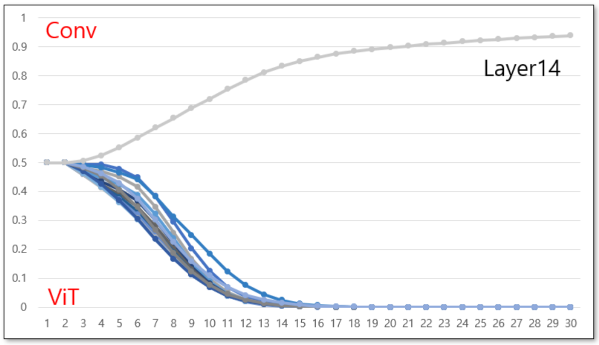

FIGURE 5. Graph of the the convergence process of the operation importance parameters by epoch using our method after the last layer is fixed and the kernel size of Conv is fixed to 5 × 5.

TABLE 5. Performance analysis of modified methods.

Method	Model Size (Param)			
	∼6M†	∼8M	∼10M	∼12M
Autoformer(Acc)	74.9%	76.61%	76.75%	76.80%
Autoformer(Param)	5.96M	7.94M	9.38M	10.04M
Proposed (Acc)	75.12%	76.70%	76.96%	77.08%
Proposed (Param)	5.98M	7.72M	9.00M	11.02M
Proposed Mod1 (Acc)	75.13%	76.89%	77.32%	77.35%
Proposed Mod1 (Param)	5.99M	7.97M	9.78M	10.95M
Proposed Mod2 (Acc)	**75.33%**	**77.00%**	**77.59%**	**77.63%**
Proposed Mod2 (Param)	5.98M	7.99M	9.85M	10.86M

†M stands for 1e6

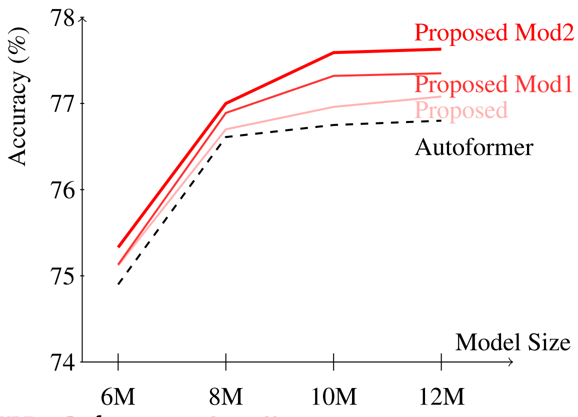

FIGURE 6. Performance on ImageNet.

and consequently by giving Conv a kernel size of only 5 × 5. Figure 5 shows the result of the modification (the one referred to as Proposed Mod2 in Figure 6 and Table 5). Every operation importance parameter converges to ViT except the one in the Head layer. The operation importance parameters for the 12th and 13th layers converge quickly toward ViT as well as the operation importance parameters for the 1st∼11th layers. In addition, there is another notable phenomenon. In comparison to the operation importance parameter of the 14th layer of Figure 3, the operation importance parameter of the 14th layer of the modified method converges to Conv more rapidly. The impact of the modifications can be observed in Figure 6 and Table 5. The performance of all model segments improves after the modifications are used.

TABLE 6. Ablation study.

Method	Model Size (Param)			
	∼6M†	∼8M	∼10M	∼12M
w/ Bias (Acc)	**75.12%**	**76.70%**	**76.96%**	**77.08%**
w/ Bias (Param)	5.98M	7.72M	9.00M	11.02M
w/o Bias (Acc)	71.45%	72.90%	73.12%	73.10%
w/o Bias (Param)	5.98M	7.77M	9.09M	10.03M

†M stands for 1e6

We could have conducted additional experiments, but we decided that continuing further would extend the scope of the study and so have left it for future research. Our observations suggest that the convergence process can vary significantly depending on the architecture of the supernet. Nevertheless, it is interesting to note that the final architecture consistently converged to the architecture that utilizes Conv in the last layer, regardless of variations in the supernet. This seems to be an interesting phenomenon.

D. ABLATION STUDY

In the ablation study, we tested our method without α' and β' to observe the ablation effect. In our method, α' and β' were used to make α and β equivalent to 1 so that the output of the candidate operations could be transferred to the next layer as it was. There is no difference between the proposed method with α' and β' and without them, considering the gradient of the loss function with respect to the operation importance parameters. The experimental result of the proposed method without α' and β' is presented in Table 6. We can see that the performance of subnets is degraded by approximately 4% without α' and β'. From this observation, we can conclude that removing the effect of the operation importance parameters in the forward process significantly affects the final result.

VI. DISCUSSION

A. OUR RESULT VS. EARLY CONVOLUTION

The final model architecture found by our method assigns Conv in the last layers. This seems contradictory to the observations in [33]. In [33], the authors argue that convolution in the early stage of ViT can provide better performance. However, it is noteworthy that they changed the patch embedding layer to a module consisting of convolutions. In our method, we used patch embedding, as it was, as a stem layer. Therefore, it is difficult to judge if our result refutes the claims of [33].

B. THE DESIGN OF CANDIDATE OPERATIONS

In our experiments, to prove the effectiveness of our method, we used the Conv, and ViT operations of previous works. It is not considered to redesign the internal structure of candidate operations. The size of an operation or the balance between candidate operations may be important for the performance. We will consider these topics in our future work.

VII. CONCLUSION

In this paper, we proposed a method that enables DynamicNAS to use different types of operations within a layer as a search space, while preserving the advantages of DynamicNAS, such as one-time training, and superior subnet performance. Through experiments, we demonstrated the effectiveness of our method and showed that it outperformed the convex combination and the set sampling methods individually. Furthermore, we observed that the convergence process of the operation importance parameters can vary significantly, depending on the design of the search space, but the final architecture remains robust to the variations. Our results provide new insight into the behavior of NAS using the convex combination method and highlight the advantages of our proposed method in improving subnet performance.

For the future work, we are considering the addition of more candidate operations, such as MLP-mixer [34]. We are also considering the automatic internal design of each operation, as mentioned earlier, for another future study. The automatic internal design would be related to the concept of Searching the Search Space (SSS), which is concerned by some previous works [35].

APPENDIX A COMPARISON BETWEEN WEIGHT-SHARING NAS AND DYNAMICNAS

In this appendix, we will present a more detailed analysis of the concept of DynamicNAS and compare it with weight-sharing NAS using examples.

A. WEIGHT-SHARING NAS

In a supernet of weight-sharing NAS, one layer of the supernet consists of an element-wise summation of the results of candidate operations. For example, let the set $\mathcal{O}$ of candidate operations in a given layer, denoted as l, comprises 3×3, 5×5 and 7×7 convolutions ($Conv_3$, $Conv_5$, $Conv_7$), all of which are commonly used in CNNs:

$$\mathcal{O}_l = \{Conv_3, Conv_5, Conv_7\}$$

For the sake of simplicity in mathematical notation, we assume that the convolutions have only one input channel and one output channel. Then, the output of a weight-sharing NAS supernet layer can be represented as follows:

$$X_l = Conv_3(X_{l-1}) + Conv_5(X_{l-1}) + Conv_7(X_{l-1}), \quad (16)$$

where

$$Conv_3(X_{l-1}) = [[o^{wh}]_{w=1}^{W}]_{h=1}^{H}, \quad o^{wh} = \sum_{i=1}^{3\times3} w_i^3 x_i^{wh}$$
$$Conv_5(X_{l-1}) = [[o^{wh}]_{w=1}^{W}]_{h=1}^{H}, \quad o^{wh} = \sum_{i=1}^{5\times5} w_i^5 x_i^{wh}$$
$$Conv_7(X_{l-1}) = [[o^{wh}]_{w=1}^{W}]_{h=1}^{H}, \quad o^{wh} = \sum_{i=1}^{7\times7} w_i^7 x_i^{wh}$$

In Eq. (16), X_l is the output feature map of the convolutions at layer l. $Conv_n(X_{l-1})$ denotes that a convolution operation having a kernel size of n applied to the input feature map X_{l-1}. o^{wh} is the output value at position (w, h) in the output feature map, which is calculated as the sum of the element-wise products of the kernel weights w_i^n and the corresponding input feature map values x_i^{wh}. W and H denote the width and height of the output feature map, respectively. Where i iterates over the $n \times n$ kernel size. Eq. (16) finally combines three convolution operations of different kernel sizes into one.

To apply the optimization method with the operation importance parameters to (16), it can be reformulated as the following:

$$X_l = \alpha Conv_3(X_{l-1}) + \beta Conv_5(X_{l-1}) + \gamma Conv_7(X_{l-1}) \quad (17)$$

where α, β, and γ, which are operation importance parameters, have different types of values according to the optimization method. When metaheuristic optimization techniques such as reinforcement learning and evolutionary algorithm are used, α, β and γ have the following values for each step of the training:

$$\{\alpha, \beta, \gamma\} = \{1, 0, 0\}, \{0, 1, 0\} \text{ or } \{0, 0, 1\}$$

We call the method, which has the above optimization structure, the set sampling method in the remainder of this paper. The sampling probability of each set is differentiated by the specific optimization method. When a first-order optimization algorithm like gradient descent is used, the values of α, β, and γ typically take on the following values:

$$\alpha, \beta, \gamma \in (0, 1), \quad \alpha + \beta + \gamma = 1$$

where α, β, and γ are trainable parameter. We call the method, which has the above optimization structure, the convex combination method in the remainder of this paper. DARTS [7] first formulated an architecture search in a differentiable manner and introduced the method. Their method updates the operation importance parameters with weight parameters during the searching stage. In addition to that, there is a modification that makes each parameter have its own probability distribution:

$$\alpha, \beta, \gamma \in (0, 1)$$

where α, β, and γ are also trainable parameters. This modification was first proposed in FairDARTS [20]. Removing the restriction $\alpha + \beta + \gamma = 1$ allows for the selection of multiple operations as the final operation.

When a supernet composed of small subnets is used, weight-sharing NAS significantly reduces the computational cost and time required to search for optimal network architectures. Before weight-sharing NAS was widely used, every small subnet was designed, trained, and tested from scratch, that is, trained, and tested through a trial and error process [6], [9], [36], [37]. It usually required considerable computational resources, such as GPUs and large amounts of memory. This process is expensive and time-consuming, especially when a large number of candidate architectures are tested.

B. DYNAMICNAS

Upon examining the commonalities between [15], [16], [17], we discover that they employ weight parameter sharing to combine candidate operations. Let us revisit the scenario in which the set of candidate operations in a layer l includes 3×3, 5×5, and 7×7 convolutions. Compared with (16) of weight-sharing NAS, the output of a DynamicNAS supernet layer can be represented as follows:

$$X_l = Conv_3(X_{l-1}) + Conv'_5(X_{l-1}) + Conv'_7(X_{l-1}), \quad (18)$$

where

$$Conv_3(X_{l-1}) = [[o^{wh}]_{w=1}^{W}]_{h=1}^{H}, \quad o^{wh} = \sum_{i=1}^{3\times3} w_i x_i^{wh}$$

$$Conv'_5(X_{l-1}) = [[o^{wh}]_{w=1}^{W}]_{h=1}^{H}, \quad o^{wh} = \sum_{i=3\times3+1}^{5\times5} w_i x_i^{wh}$$

$$Conv'_7(X_{l-1}) = [[o^{wh}]_{w=1}^{W}]_{h=1}^{H}, \quad o^{wh} = \sum_{i=5\times5+1}^{7\times7} w_i x_i^{wh}$$

It is important to note, in Eq. (18), that the starting indexes for summation in $Conv'_5(X_{l-1})$ and $Conv'_7(X_{l-1})$ differ from those in $Conv_5(X_{l-1})$ and $Conv_7(X_{l-1})$. Moreover, although each candidate operation of (16) uses its own weight(w_i^n), the weights of the candidate operations of (18) are sampled from the same set(w_i). $Conv_3(X_{l-1})$, which is the smallest operation, is the same as $Conv_3(X_{l-1})$ of (16). $Conv'_5(X_{l-1})$ cannot be used as a standalone operation. It must be used with $Conv_3(X_{l-1})$ to function as a complete operation. $Conv_5(X_{l-1})$ of (16) can be obtained from $Conv_3(X_{l-1}) + Conv'_5(X_{l-1})$. Similarly, $Conv'_7(X_{l-1})$ cannot be used as a standalone operation. $Conv_7(X_{l-1})$ can be obtained from $Conv_3(X_{l-1}) + Conv'_5(X_{l-1}) + Conv'_7(X_{l-1})$. Eq. (18) can be reformulated to apply the optimization method with the operation importance parameters in the same way as (17):

$$X_l = \alpha Conv_3(X_{l-1}) + \beta Conv'_5(X_{l-1}) + \gamma Conv'_7(X_{l-1}), \quad (19)$$

$$\text{where} \quad \{\alpha, \beta, \gamma\} = \{1, 0, 0\}, \{1, 1, 0\} \text{ or } \{1, 1, 1\}$$

$Conv'_5(X_{l-1})$ and $Conv'_7(X_{l-1})$ cannot be used as a standalone operation, so the sets α, β, and γ are different from the sets of the weight-sharing NAS. In practical implementation, only $Conv_7(X_{l-1})$ needs to be declared, encompassing $Conv_3(X_{l-1})$, $Conv'_5(X_{l-1})$, and $Conv'_7(X_{l-1})$. The weight parameters w_i for $Conv_5(X_{l-1})$ are extracted by isolating the core parameters of $Conv_7(X_{l-1})$, excluding the surrounding ones. Similarly, the weight parameters for $Conv_3(X_{l-1})$ are derived from $Conv_5(X_{l-1})$.

DynamicNAS can also be extended to other aspects of CNN, such as the number of channels and layers (also known as the width and depth of CNN) [14], [15], [38]. Additionally, DynamicNAS can be applied to ViT architectures. For example, in the Autoformer [17] model, DynamicNAS is used to optimize the dimension of the representation vector, the number of heads, the expansion ratio, and the number of layers.

A layer of a supernet can be modeled as a function that takes the output of the previous layer as the input; we can represent 3 layers of a supernet as:

$$X_l = F_l(F_{l-1}(X_{l-2})), \quad (20)$$

where F_n denotes the operation of layer n. While (20) contradicts the structure of DynamicNAS, whose operations are entangled through summation, it is possible to give the operations a structure that aligns with DynamicNAS by adopting the residual connection technique, in which $F(x) = x + f(x)$ [1]. Then Eq. (20) can be reformulated as:

$$X_l = X_{l-2} + f_{l-1}(X_{l-2}) + f_l(X_{l-2} + f_{l-1}(X_{l-2})) \quad (21)$$

If the layers from 1 to $l-1$ are included, the output becomes $X_{l-2} + f_{l-1}(X_{l-2})$. If the layers from 1 to $l-2$ are included, the output becomes X_{l-2}. Thus, the use of the residual connection enables the entanglement of operations between layers, resulting in a more complex and flexible supernet structure.

The entanglement of operations in DynamicNAS leads to a notable reduction in the number of parameters in the supernet, facilitating the search for various subnets across a wide range of configurations. DynamicNAS incurs low memory cost compared with weight-sharing NAS. In addition to that, an important observation is that every subnet of a trained supernet of DynamicNAS can be used immediately after the supernet training stage without additional training or fine-tuning [14], [16], [17]. This is a characteristic of DynamicNAS. There has been no theoretical analysis of the advantages of DynamicNAS so far; this remains for future work. After supernet training is complete, an optimization technique is often employed to explore the final subnet in the searching stage. Various architectures may be suitable for a particular environment. The optimization technique determines the best subnet for the environment. Such optimization commonly involves the use of an evolutionary algorithm [13], [16], [17].

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[2] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: Towards end-to-end speech synthesis," 2017, *arXiv:1703.10135*.
[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 5598–6008.
[4] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, Oct. 2019.
[5] Z. Xing, S. Zhao, W. Guo, F. Meng, X. Guo, S. Wang, and H. He, "Coal resources under carbon peak: Segmentation of massive laser point clouds for coal mining in underground dusty environments using integrated graph deep learning model," *Energy*, vol. 285, Dec. 2023, Art. no. 128771.

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

[7] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.

[8] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 19–34.

[9] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.

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

[11] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[12] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–34, May 2022.

[13] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, "Neural architecture search for transformers: A survey," *IEEE Access*, vol. 10, pp. 108374–108412, 2022.

[14] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2019, *arXiv:1908.09791*.

[15] D. Wang, M. Li, C. Gong, and V. Chandra, "AttentiveNAS: Improving neural architecture search via attentive sampling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 6414–6423.

[16] C. Gong, D. Wang, M. Li, X. Chen, Z. Yan, Y. Tian, and V. Chandra, "NASViT: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training," in *Proc. Int. Conf. Learn. Represent.*, 2021.

[17] M. Chen, H. Peng, J. Fu, and H. Ling, "AutoFormer: Searching transformers for visual recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 12250–12260.

[18] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang, and Q. Tian, "Weight-sharing neural architecture search: A battle to shrink the optimization gap," *ACM Comput. Surv.*, vol. 54, no. 9, pp. 1–37, Dec. 2022.

[19] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.

[20] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 465–480.

[21] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.

[22] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," 2018, *arXiv:1812.08928*.

[23] J. Liu, J. Cai, and B. Zhuang, "FocusFormer: Focusing on what we need via architecture sampler," 2022, *arXiv:2208.10861*.

[24] H. Wang, C. Ge, H. Chen, and X. Sun, "PreNAS: Preferred one-shot learning towards efficient neural architecture search," 2023, *arXiv:2304.14636*.

[25] L. Weng. (Aug. 2020). *Neural Architecture Search*. lilianweng.github.io. [Online]. Available: https://lilianweng.github.io/posts/2020-08-06-nas/

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

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

[28] Z. Guo, "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Oct. 2020, pp. 544–560.

[29] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10347–10357.

[30] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun, "ConViT: Improving vision transformers with soft convolutional inductive biases," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 2286–2296.

[31] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 15908–15919.

[32] C. Yang, Y. Wang, J. Zhang, H. Zhang, Z. Wei, Z. Lin, and A. Yuille, "Lite vision transformer with enhanced self-attention," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 11988–11998.

[33] T. Xiao, P. Dollar, M. Singh, E. Mintun, T. Darrell, and R. Girshick, "Early convolutions help transformers see better," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 30392–30400.

[34] I. Tolstikhin, "MLP-mixer: An all-MLP architecture for vision," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 24261–24272.

[35] M. Chen, K. Wu, B. Ni, H. Peng, B. Liu, J. Fu, H. Chao, and H. Ling, "Searching the search space of vision transformer," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 8714–8726.

[36] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[37] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[38] D. Wang, C. Gong, M. Li, Q. Liu, and V. Chandra, "AlphaNet: Improved training of supernets with alpha-divergence," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10760–10771.

IKSOO SHIN received the B.S. degree from Kyungpook National University, South Korea, in 2012. He is currently pursuing the Ph.D. degree with the University of Science and Technology, South Korea.

His research interests include AutoML, neural architecture search, and the out-of-distribution detection.

CHANGSIK CHO received the B.S. and M.S. degrees from Kyungpook National University, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree from the Department of Computer Science, Chungnam National University, South Korea, in 2011.

In January 1995, he joined the Electronics and Telecommunications Research Institute (ETRI), South Korea, where he is currently a Principal Researcher. His research interests include AutoML, MLOps, and no-code neural network development tool.

SEON-TAE KIM (Member, IEEE) received the B.S. degree from the Department of Electrical and Electronics Engineering, KAIST, in 1997, the M.S. degree from Seoul National University, in 2000, and the Ph.D. degree from Korea University, South Korea, in 2012.

He was a Principal Researcher with the Electronics and Telecommunication Research Institute (ETRI), South Korea, in February 2000. Since 2021, he has also been a Professor with the Department of Artificial Intelligence, University of Science and Technology. His research interests include image processing, visual representation, lightweight OS, and sensor networks.