Neural Architecture Search for Convolutional and Transformer Deep Neural Networks

BOYU CHEN

PhD candidate

Research Supervisor: Wanli Ouyang
Auxiliary Supervisor: Dong Xu

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Electrical and Information Engineering
The University of Sydney
Australia

31 December 2023

Statement of Originality

This thesis has not been submitted for any degree or other purposes. I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Boyu Chen | 28 February, 2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Wanli Ouyang | 28 February, 2023

Author Attribution

In this thesis, except the otherwise referenced, all research content is my original contribution. These are research outcomes I have published or submitted in the thesis.

Chapter 2 of this thesis is the published paper Chen et al. 2021a on the ICCV2021. In this paper I proposed a new method for the fast neural architecture search. I proposed the idea, conducted the experiments and drafted most of the paper. My co-authors helped me finish the whole paper and provided advice on the ideas and experiments.

Chapter 3 of this thesis is the published paper Chen et al. 2021b on the ICCV2021. In this paper I proposed a new vision transformer architecture through introducing convolutional layers and neural architecture search. I proposed the initial idea, conducted the experiments and drafted the main part of the paper. My co-authors help me complete the whole idea, design more experiments and finish the whole paper.

Chapter 4 of this thesis is under review at IEEE Transactions on Circuits and Systems for Video Technology. In this paper I propose a new technique for the vision transformer network and propose a new architecture based on neural architecture search. I proposed the idea, conducted the experiments and drafted part of the paper. My co-authors help me design experiments and review the final paper.

Boyu Chen | 28 February, 2023

Wanli Ouyang | 28 February, 2023

Abstract

Deep Neural Networks (DNN) have dominated computer vision tasks during the last decade. With multiple layers which can learn and recognize complex patterns and features from training data, DNNs demonstrate impressive proficiency across a variety of tasks. However, in recent years, the demand for highly customized deep neural networks has increased significantly, which has made manual design of architectures a tedious and time-consuming process. Even for human professionals, it's still a challenging task due to the expense of testing and verifying.

Neural architecture search (NAS), which aims to find the optimal network architecture automatically, has significantly improved network performance and shortened the process of designing the architectures in many computer vision tasks. In the past several years, Convolutional Neural Networks (CNN)-based architecture has been the mainstream architecture in computer vision tasks. Although many methods have improved the performance of CNNs through neural architecture search, the computation cost of these methods is not affordable by most researchers, which is about thousands of GPU hours. To this end, we propose a new indicator in the CNN neural architecture search process that significantly reduces the time required by model training and evaluation during the NAS in Chapter 2. Our method significantly accelerates the training and evaluation processes by over 10 times and 600,000 times, respectively, while maintaining consistent performance.

Recently, transformers without CNN-based backbones have been found to achieve impressive performance for image recognition. However, the simple network architecture following the designs in the Natural Language Processing (NLP) tasks cannot be the optimal choice in the computer vision tasks. So we propose the first neural architecture search method for the Vision Transformer models and improves the performance of the ViT models in Chapter 3. We introduce convolutional layers into the Vision Transformer networks and propose a new hierarchical NAS method to tackle the huge search space. Our searched model achieves a 4%

and 6% accuracy improvement compared to two popular baselines, DeiT-Tiny and ResNet18 (R18) respectively, at a comparable computational cost.

Finally, in Chapter 4, we introduce a new search space for ViT model and utilize the neural architecture search method to find the optimal architecture based on our new search space. We find the redundancy among transformer blocks and propose a new Attention-Sharing technique to improve the network capability. We also utilize neural architecture search techniques to decide whether each block should have the sharing attention or not and improve the network performance. Our searched models are capable of achieving up to a 6.6% increase in accuracy on the ImageNet classification task when compared with the baseline DeiT models.

To sum up, we propose methods for deep neural network architecture search, including CNN and ViT models. We intend to enhance the effectiveness of NAS methods and the performance of existing models. We hope the works will contribute to the field and make NAS methods affordable to all researchers.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Prof. Ouyang and Prof. Xu, for their tremendous advice, assistance, and encouragement throughout my PhD period. My academic career has been greatly shaped by their experience, knowledge, and persistent dedication to quality, which has allowed me to achieve this important milestone.

I would also like to thank my greatest partner and collaborator, Peixia Li. Without her support, there is no possibility for me to finish my PhD program. Besides, I really treasure the time we spent together on our projects and research.

I would like to extend my appreciation to my family for their unfailing love, understanding, and support during my PhD period. Their selflessness, support, and inspiration have served as a continual source of motivation for me.

Finally, I would like to express my gratitude to all the participants for donating so freely of their time and effort to my research project. This thesis would not be feasible without their significant contributions.

Thank you all for your invaluable support and encouragement.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Definition of Neural Architecture Search

The neural network is a type of machine learning model which has the ability to learn from data on complex tasks by adjusting their internal parameters through back-propagation algorithm. Recently, neural networks have achieved great progress in a wide range of applications, such as computer vision, natural language processing, and robotics. However, as the key part of neural networks, architecture is hard to design due to the extensive domain knowledge requirements and trial and error experimentation to determine the optimal architecture for the given task. Designing the architecture manually is both resource-intensive and time-consuming, and the results are often suboptimal.

Neural architecture search (NAS) is a research field that aims to design neural network architectures through search algorithms automatically. By utilizing search algorithms, NAS seeks to automate the process of network architecture design and explore the space of candidate network architectures. The goal of NAS is to find the optimal network architecture for a given task, as well as minimizing manual design and engineering needs. NAS has the great potential to reduce the amount of time and resources required to design neural networks significantly. Besides, NAS has the ability to improve the performance of these networks on a wide range of tasks.

In recent years, NAS has developed rapidly, with numerous methods and algorithms being proposed to tackle the challenge of network automating design. These methods can be roughly categorized into three types:

- Reinforcement learning-based methods: These methods use reinforcement learning to optimize an agent for network architecture generation. To get the reward for reinforcement learning, these methods need to train a mass of networks and evaluate their performance, which has tremendous computation cost.
- Evolutionary algorithms: These methods generate network architecture with the principles of natural selection. A population of networks is initialized, and each network is evaluated based on its performance. The search algorithm then selects the best-performing networks and combines their characteristics to create new networks. This process is repeated and the optimal architecture is the output of the final generation.
- Gradient-based methods: These methods use additional architecture parameters, which is a set of learnable parameters, to represent the architecture of the neural networks. They utilize gradient descent to optimize the architecture parameters and choose the optimal architectures according to the architecture parameters.

1.2 Challenges of Neural Architecture Search

Although progresses have been made in NAS, there are still many issues that should be resolved. Improving the efficacy of NAS methods is one of the major challenges. The search space of candidate network architectures is incredibly enormous, which existing NAS methods will be computationally expensive on. Enhancing the efficacy of these methodologies will be critical for applying them on the real-world applications.

The second is about the scalability of NAS methods. A typical NAS method will be applied on the fixed search space and specified constraint. When we try to scale the searched model with increasing channel numbers or depth of the model, the searched architecture under different constraints will be the suboptimal choice. The issue can be solved through searching a new architecture on a new scale, but the computation cost will be a burden for the application.

Another challenge to handle is developing more effective search algorithms. Currently, NAS techniques rely on a limited number of search algorithms, and the optimal algorithm for a given task remains uncertain. Consequently, the development of new search algorithms and their comparative assessment will become an important area of future study.

Finally, the last challenge for NAS is the issue of transferability. Similar to the scalability, networks searched for one task may not perform as well on another task and the generalization of NAS techniques to multiple tasks remains ambiguous. Thus, the development of NAS methods that can search networks with transferability across tasks will be an important field of future research.

In summary, NAS has the potential to revolutionize the field of deep learning by automating the design of neural network architectures and enabling more efficient and effective resource utilization. However, there are still many challenges to be addressed before NAS methods can be widely applied in real-world applications.

1.3 Literature of Neural Architecture Search

In this subsection, we present a literature review of neural architecture search. Considering the perspective of binary classification, the present neural architecture search algorithms may be broadly categorized into three distinct groups: reinforcement learning-based methods, evolutionary algorithms, and gradient-based methods. In this part, we will introduce several typical methods within each category.

Reinforcement Learning-based methods. In this method, the neural network architecture is treated as a parameter which should be optimized by a Reinforcement Learning agent. A mass of neural networks with different architectures are trained and evaluated to provide the training data for the agent and the agent utilizes this data to guide its search for the optimal architectures.

The concept of neural architecture search is proposed in (Zoph and Le 2016), which is treated as the first paper in this field. In this paper, the author utilizes a recurrent neural controller to

generate architectural hyperparameters of neural networks. With the generated architecture, the author trains generated model and gets the accuracy as the reward for training the RNN controller with reinforcement learning on a validation set. The process of 'generating-training-updating controller' is repeated until the controller converges. The searched model achieves better accuracy than every hand-designed deep network. However, this method needs to train 12,800 models to get enough data for the controller and it's so time-consuming that not anyone can afford the cost. Besides, this method can only search the network architecture on the small dataset which means there is not much practical value.

NASNet (Zoph et al. 2018) proposes a new search space for neural architecture search, which is named NASNet search space. In this search space, the searched architecture can be transferred to other datasets. In this paper, the neural architectures are searched based on the small dataset CIFAR-10 and then they scale the searched network architecture through stacking more copies of the searched cell to the large dataset, ImageNet. Besides, a new regularization technique called ScheduledDropPath is proposed for better neural network training, improving the performance of searched networks significantly. The searched architecture can also be applied on other computer vision tasks, such as object detection, and surpasses state-of-the-art human-designed models.

Evolutionary algorithms. For neural architecture search methods with evolutionary algorithms(EA), they utilize a random population of network architecture as the initialization of the evolution. Then they generate new architectures according to the performance of each network through crossover and mutation. The searched architecture is the final round evolution result.

AmoebaNet (Real et al. 2019) searched neural architecture based on the same search space on the NASNet. In this paper, the author proposes a new evolution algorithm called Regularized Evolution. Instead of measuring the architectures only through performance, an age property is introduced to favour the newer architecture. Compared with Reinforcement Learning based neural architecture search methods, the proposed Regularized Evolution method achieves a much faster convergence speed without any performance drop with a similar model size.

Furthermore, the searched AmoebaNet achieves the new state-of-the-art accuracy on ImageNet dataset when scaled up.

SPOS (Guo et al. 2020b) utilizes the weight-sharing one-shot method to estimate each architecture performance for Evolutionary Algorithms. A Single Path One-Shot method is proposed for better training of the weight-sharing supernet. The main idea is to alleviate the weight co-adaption issue in the weight-sharing models through a simplified supernet, which is only constructed by single-path architectures. The training follows uniform path sampling and the searching follows the standard Evolutionary Algorithms. The method achieves a much faster searching speed compared with AmoebaNet. Besides, it is flexible with complex search spaces and different search constraints and achieves start-of-the-art performance on the large dataset ImageNet.

Eproxy (Li et al. 2023) proposes an Extensible proxy (Eproxy), which leverages self-supervised learning and few-shot training techniques to minimize costs almost to zero. The core component of the Eproxy is a barrier layer, which is randomly initialized and frozen during the training phase. This layer introduces non-linear characteristics into the optimization process, enabling Eproxy to effectively evaluate different architectural performances in the preliminary stages. Besides, the authors introduce a Discrete Proxy Search (DPS) method. This method is designed to efficiently identify the most effective training settings for Eproxy, requiring only a limited number of benchmark tests on selected architectures for specific tasks.

Gradient-based methods. In gradient-based NAS, the architecture of the network is represented as a set of additional architecture parameters that are learned through a search process. The architecture parameters define the structure of the neural network and are learned through gradient descent on the training dataset. After training, the optimal architecture is selected through the magnitude of the architecture parameters.

DARTS (Liu et al. 2019c) proposes a differentiable architecture search method which reduces the searching cost greatly. This paper transfers the searching process over a discrete and non-differentiable search space to the continuous relaxation of the architecture representation.

In this way, the searching process can be very efficient using gradient descent. Besides, due to the time cost of calculating the gradient directly, this paper proposes an approximate architecture gradient method, which reduces the time cost further. The output of the cell is consisted of the weighted sum of all the operation in the search space, and the weight for the operation is optimized during the search process. After searching, the operation with maximal weight is kept as the searched architecture. This method achieves state-of-the-art accuracy performance while reducing searching costs from 2000 GPU days to 1.5 GPU days. Although this method has some drawbacks, such as the high GPU memory consumption, only being able to search on a small dataset, it is great work since the searching cost of the gradient-based method is little and the Neural Architecture Search method can be widely applied.

ProxylessNas (Cai et al. 2019) is proposed to directly search the architectures for the large-scale target task. Besides, this paper applies different loss function to search different neural architectures for different hardware with direct hardware metrics. The proxy means the DARTS need to search the architecture on the proxy tasks, such as searching on the small dataset or searching a small model and enhancing the small model after searching to get the full model. To reduce the GPU consumption, the ProxylessNas binarize the architecture parameters and only active one path at run-time. During the training process, these binarized parameters are trained based on BinaryConnect. Similar to DARTS, the path with maximal weight is kept as the searched architecture. Furthermore, to search the architecture for different hardware, this paper model the hardware metrics as a continuous function and optimize it with the architecture loss directly. This paper searches the network architecture directly on the large-scale dataset and achieves better performance than DARTS. Besides, they also search different models for different devices.

ShiftNAS (Zhang et al. 2023) investigates the performance gap between the training and testing stages and attributes this discrepancy to the use of uniform sampling, which is a common approach in supernet training. To address the problem of uniform sampling, the method utilizes a learnable sampling strategy to alleviate the inherent bias of uniform sampling. Furthermore, an LSTM-based architecture generator is proposed, providing the optimal subnet under the resource constraints. Both the sampling probability and the architecture generator

can be trained end-to-end in a gradient-based method. With ShiftNAS, one can directly obtain the optimal model architecture and parameters for a given computational complexity across various visual network models, including convolutional neural networks (CNNs) and vision transformers (ViTs).

1.4 Thesis Aims and Outline

The primary aim of this thesis is to investigate the issues in the field of neural architecture search and propose new methods to address these issues. We evaluate our methods on various computer vision tasks, including image classification, object detection, and instance&semantic segmentation. This thesis has five chapters. Chapter 1 gives a introduction to the neural architecture search. The following three Chapters, including Chapter 2, Chapter 3 and Chapter 4 propose three methods to improve efficiency and performance for neural architecture search methods. Finally, Chapter 5 gives the conclusion of these three methods and a future outlook.

- Chapter 1 **Introduction.** In this chapter, we present the definition and existing challenges of neural architecture search, a comprehensive literature review of neural architecture search, and the aims and outline of the thesis. This chapter establishes the foundational knowledge and context upon which the subsequent chapters build. The challenges and gaps highlighted in Chapter 1 act as a driving force behind the innovations proposed in subsequent chapters.
- Chapter 2 **BN-NAS: Neural Architecture Search with Batch Normalization.** In this chapter, we propose a new efficient indicator for neural architecture search with Evolutionary Algorithms. We utilize the widely-applied Batch Normalization to select the optimal architecture from candidate architectures and improve the NAS process efficiency. This chapter presents a direct response to the efficiency challenges as outlined in Chapter 1. Incorporating Batch Normalization into NAS alongside Evolutionary Algorithms, this chapter showcases significant advancements in tackling the specific issues previously identified.

- Chapter 3 **GLiT: Neural Architecture Search for Global and Local Image Transformer.** Recently, Vision Transformer is introduced into the field of computer vision and achieves comparable even better performance than traditional convolutional networks. This chapter presents a new network combining the convolutional layers and Vision Transformer with neural architecture search. We also propose a new search method to handle the increased search space due to the introduction of convolutional layers. Different from Chapter 2, which focuses on convolutional networks, this chapter advances the discussion by integrating convolutional layers with Vision Transformers in NAS. This approach addresses the expanded search space and increased complexity in NAS, challenges that originate from the foundational issues outlined in Chapter 1.

- Chapter 4 **PSViT: Better Vision Transformer via Token Pooling and Attention Sharing.** Similar to Chapter 3, this chapter also searches the Vision Transformer network architecture. Different from Chapter 3, which introduces the convolutional layer into Vision Transformer, we focus on the pure transformer architecture. We find the redundancy among transformer blocks and introduce the new Attention-Sharing into the search space for Vision Transformer. This chapter complements the work in Chapter 3 by exploring a novel perspective on the transformer model in NAS, thereby offering an alternative approach to addressing the challenges of redundancy and efficiency.

- Chapter 5 **Conclusion.** This chapter gives the summary of the three new proposed methods and have a future outlook in the field of neural architecture search. This chapter effectively brings the thesis full circle by connecting the conclusions and future directions back to the initial motivations and challenges introduced in Chapter 1. It not only highlights the ways in which the methods proposed in Chapters 2, 3, and 4 address the challenges and gaps identified in Chapter 1 but also demonstrates how these methods collectively contribute to the field of NAS.

CHAPTER 2

BN-NAS: Neural Architecture Search with Batch Normalization

Model training and evaluation are two main time-consuming processes during neural architecture search (NAS). Although weight-sharing based methods have been proposed to reduce the number of trained networks, these methods still need to train the supernet for hundreds of epochs and evaluate thousands of subnets to find the optimal network architecture. In this chapter, we present BN-NAS, neural architecture search with Batch Normalization, to accelerate neural architecture search. BN-NAS can significantly reduce the time required by model training and evaluation in NAS. Specifically, for fast evaluation, we propose a BN-based indicator for predicting subnet performance at a very early training stage. The BN-based indicator further facilitates us to improve the training efficiency by only training the BN parameters during the supernet training. This is based on our observation that training the whole supernet is not necessary while training only BN parameters accelerates network convergence for network architecture search. To activate the BN-based indicator in a broad range of operations and diverse NAS frameworks, we further present a pseudo-BN strategy with which a pseudo-BN layer is added at the end of those operations without BN layers. Extensive experiments show that our method can significantly shorten the time of training supernet by more than 10 times and shorten the time of evaluating subnets by more than 600,000 times without losing accuracy.

2.1 Motivations and Contributions

Deep neural networks have shown powerful representation capabilities in various computer vision tasks. A superior network architecture can bring a substantial performance gain.

Although previous works (Krizhevsky et al. 2012; Simonyan and Zisserman 2015; Szegedy et al. 2015; He et al. 2016; Huang et al. 2017) on network architecture designing greatly improve model performance, modifying network architecture artificially usually relies on prior knowledge and consumes a lot of manual try-and-error. Neural architecture search (NAS), which aims to find the optimal network architecture automatically, can liberate labor costs in network designing. It has significantly improved the network performance in many computer vision tasks, such as image classification (Zoph et al. 2018; Guo et al. 2020b; Li et al. 2020b; Ci et al. 2021; Chen et al. 2021b), object detection (Liang et al. 2020; Chen et al. 2019c; Liu et al. 2021), semantic segmentation (Chen et al. 2018b; Liu et al. 2019a), etc. However, a successful NAS method (Zoph et al. 2018; Real et al. 2019) usually means sampling and training a large number of architecture candidates from scratch, which takes up to thousands of GPU days. The huge searching budget makes NAS hard to be applied widely.

To overcome the above issue, one-shot methods (Pham et al. 2018; Guo et al. 2020b), have been proposed to reduce the computational cost based on the weight-sharing technique, reducing the search cost from thousands of GPU days to tens of GPU days. These methods construct a supernet that includes all candidate network architectures, which share a portion of parameters with each other. With the constructed supernet, one-shot NAS methods consist of three main stages: supernet training, subnet searching and subnet retraining. In each iteration of the supernet training stage, a subnet is randomly sampled from the supernet and trained by back-propagation. In the subnet searching stage, different subnets are sampled from the trained supernet and treated as the candidate architectures. The sampled subnets are evaluated on validation data, from which the top-k subnets with the highest accuracy on validation data are selected, like SPOS (Guo et al. 2020b). The selected subnets are then retrained from scratch in the subnet retraining stage. The primary benefit of one-shot methods is that the subnets can inherit the weights of supernet to reduce the computational burden significantly. However, the process of training supernet hundreds of epochs and evaluating thousands of subnets is still time-consuming, leading to tens of GPU days cost.

In this chapter, we identify the parameters learned at the Batch Normalization (BN) layer as the key to significantly reduce the excessive time required by one-shot methods in both

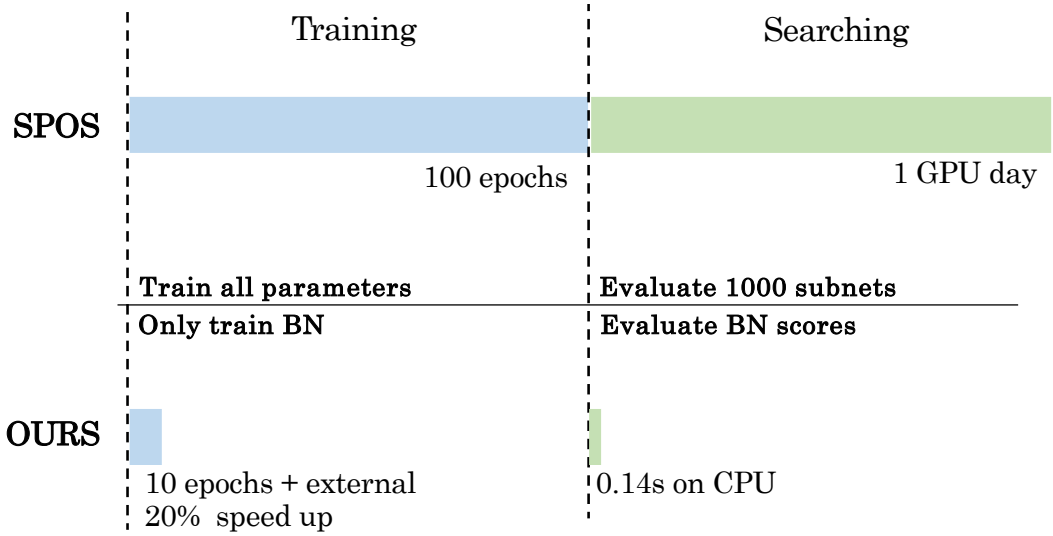


FIGURE 2.1: Designs and computational cost of SPOS and Our BN-NAS. Compared with SPOS, our proposed BN-NAS can accelerate the one-shot methods in two stages: training supernet more than ten times faster, and searching subnets more than 600,000 times faster. The key to the speed-up is the BN-based indicator, which saves the searching cost and facilitates the training only BN paramters with much fewer epochs. SPOS needs 11 GPU hours in total. Ours needs only 0.8 GPU hours.

supernet training and subnet searching stages. It is widely accepted that the BN parameter of a channel reflects the channel importance for network pruning approaches (Liu et al. 2017b; Kang and Han 2020). Channels with smaller BN parameters are considered as less important and pruning these channels will cause a small influence on network performance (Liu et al. 2017b). Considering BN parameters can reflect the channel importance, it is natural to accumulate the BN parameters from multiple channels to measure the importance of candidate operations and even the whole subnet. Based on this observation, we propose a novel BN-based indicator to measure the importance of operations as well as subnets. Unlike previous one-shot methods (Pham et al. 2018; Guo et al. 2020b) which evaluate thousands of subnets on validation data to get their performances, our BN-indicator is a more light-weight measurement of the subnet importance. Our BN-indicator can significantly reduce the searching cost from about 1 GPU day for SPOS (Guo et al. 2020b) to 0.14s on CPU for ours in the subnet searching stage, as shown in the column for 'searching' in Fig. 2.1.

The BN-indicator further motivates us to only train the BN parameters of supernet in the supernet training stage. In our BN-indicator, only parameters of BN layers matter subnet importance, so there is no need to ensure an optimal accuracy of the whole network. Instead, training BN parameters is enough if the properties of trained BN parameters can reflect subnet performance. To train a supernet, it is a general practice to train all parameters, $i.e.$, parameters of convolutional layers, fully connected layers, and BN layers. Yet training BN layers only is not groundless. Frankle $etal.$ (Frankle et al. 2020) find that networks only training BN parameters with other randomly initialized parameters fixed still have a certain capacity. During the supernet training stage, our BN-NAS only trains BN parameter but does not train the other parameters such as convolutional or fully-connected layers for two reasons: 1) the network can encode the knowledge from training data through training only a part of parameters, as found in (Frankle et al. 2020); 2) we focus on using BN parameters as the indicator for searching instead of network accuracy. We empirically find that training only BN parameters helps BN parameters become stable at earlier training epochs. Besides, there is an extra training speedup benefit from only training BN parameters. Based on the observations above, we propose a new BN-NAS. The BN-NAS train supernet with much fewer training epochs, and search subnets using the novel BN-based indicator for much faster speed.

One of the limitations of the proposed BN-indicator is that it requires the presence of Batch Normalization layers in the candidate operations, making it unsuitable for operations without BN layers, such as skip connections and pooling layers. To overcome this limitation, we propose a pseudo-BN strategy. For operations without BN layers, we manually add a pseudo-BN layer at the end, which is trained along with the normal BN layers during the supernet training process. This pseudo-BN layer is used to evaluate the importance of the operation during subnet searching. After the optimal subnet architecture has been searched, the pseudo-BN layers are removed for this operation during the subnet retraining. This strategy, combined with our BN-indicator, enables efficient computation in a wide range of operations and NAS frameworks.

To summarize, the main contributions are as follows:

- We propose a BN-based indicator for evaluating network architectures, which can significantly shorten the time required by one-shot NAS methods for searching candidate network architectures.
- We only train the BN parameters of the supernet and significantly reduce the number of epochs required for training the supernet, which is based on the use of BN-based indicator when evaluating network architectures. Training BN parameters only and reducing the training epochs could have adverse effects on the normal network architecture searching stage. However, with our BN-based indicator for searching, the adverse effect is overcome.
- We propose the pseudo-BN strategy to activate our BN-indicator in candidate operations without BN layers, making our BN-indicator more applicable to different operations and NAS frameworks.
- Extensive experiments on various datasets and downstream tasks demonstrate that our method can significantly improve the speed of NAS in the training stage (more than 10X, for example, from 100 to 10, with an external 20% speed up for SPOS as shown in Fig. 2.1) and searching stage (more than 600000X) without losing accuracy.

2.2 Related Works

2.2.1 Reinforcement Learning and Evolutionary Algorithm for NAS

NAS methods are proposed for automatic network architecture designing. Early methods utilize reinforcement learning (RL) (Zoph et al. 2018; Bello et al. 2017) or Evolutionary algorithm(EA) (Real et al. 2019) to generate network architecture samples. The generated network samples are evaluated on validation dataset and their accuracies are treated as rewards to guide the RL and EA to generate better architecture samples. Zhou $etal.$ (Zhou et al. 2020) propose an optimal proxy for the Economical Neural Architecture Search. However, the sampling and training processes are still time-consuming, making it difficult for NAS to be deployed on large-scale datasets such as ImageNet (Deng et al. 2009).

2.2.2 Weight-sharing NAS

To overcome the time-consuming problem of RL and EA, methods based on weight-sharing mechanism are proposed. These methods adopt the supernet constructed by all candidate subnets and divide the NAS process into three stages. Based on the difference in the training and searching stage, these methods can be divided into one-shot methods and differentiable methods.

One-Shot Methods. One-shot methods construct the supernet with candidate subnets directly and train the supernet based on sampled subnets for hundreds of epochs. After supernet training, thousands of subnets are sampled and evaluated on the validation set to find the optimal subnet architecture based on the validation accuracy. Since the search space is enormous, EA algorithm is adopted to generate subnets to be evaluated. Most one-shot methods focus on subnets sampling during the training. (Guo et al. 2020b) constructs the supernet and then trains the supernet through single-path random sampling. Based on (Guo et al. 2020b), (Chu et al. 2021) proposes a fair sampling method to alleviate supernet bias and improve the evaluation capacity. (You et al. 2020) proposes a sampling pool and samples subnets in the pool during the supernet training, improving the training efficiency. Unlike the above methods, we only train the BN parameters in supernet which are based on different sampling policies for much fewer epochs. Besides, we evaluate the subnets through our proposed BN-based indicator instead of evaluating subnets on validation set, accelerating the searching stage significantly.

Differentiable Methods. Different from one-shot methods, differentiable methods construct the supernet with additional architecture parameters. During the supernet training, the subnet sampling is controlled by architecture parameters which are trained alternatively with subnet parameters. After the supernet training, the optimal architecture is selected according to the magnitudes of the architecture parameters. (Liu et al. 2019c) treats the architecture parameters like the weight for the subnet output and updates the architecture parameters by back-propagation. (Cai et al. 2019) binarizes the architecture parameters to save the GPU memory usage during the supernet training. (Xie et al. 2019) introduces Gumbel random

variables to train the subnet and architecture parameters directly. However, these architecture parameters can introduce biases in the training process towards certain operations, particularly in skip connections. In contrast, our proposed method does not rely on external parameters and maintains fairness among all architectural candidates throughout the training process.

2.2.3 Efficient NAS Methods

Despite extensive research efforts aimed at reducing the computation cost of the Neural Architecture Search (NAS) process, the NAS process continues to pose a significant challenge and requires tens of GPU days for practical implementation. To address this issue, various methods have been proposed to improve the efficiency of NAS. For instance, the work presented in (Xiang et al. 2021) proposes a perturbation-based zero-cost operation selection technique for the Differentiable Neural Architecture Search (DARTS) method, with the goal of enhancing the speed of the search process. Similarly, the Zen-NAS method presented in (Lin et al. 2021) leverages a newly-proposed metric, the Zen-Score, as a gauge of subnet expressively, enabling the search for network architecture without the need for training network parameters. Our proposed method differs from these previous works in that it focuses on improving the efficiency of existing NAS methods and can increase their speed by more than 10 times without sacrificing performance. This distinguishes our approach from other NAS methods that either proposes completely new approaches or optimizes specific NAS methods like DARTS.

2.3 Method

2.3.1 Preliminary

Our approach can be seamlessly applied to the One-shot NAS method (Guo et al. 2020b; Chu et al. 2021) and differentiable NAS method (Liu et al. 2019c), focusing on the Batch Normalization Layer (Ioffe and Szegedy 2015) of different operations. A brief introduction of them is provided in this subsection.

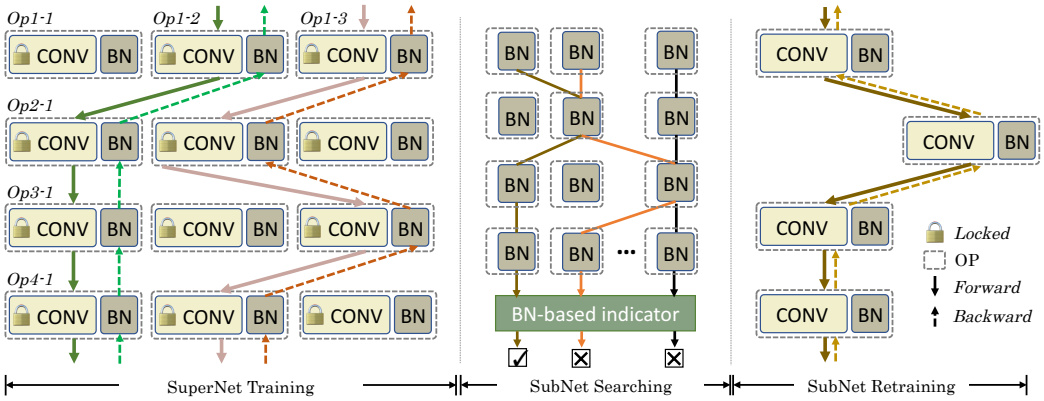


FIGURE 2.2: Overview of the proposed framework on one-shot method. We follow the three stages in one-shot methods. In supernet Training, we fix the convolution parameters and only train BN parameters for few epochs. In a iteration of supernet training, only a single path is sampled from the supernet for forward propagation, e.g. following the green solid arrows, and back-propagation, e.g. green dashed arrows. In Subnet Searching, we search the subnets (lines of the same color is a subnet) with proposed BN-based indicator. In Subnet Retraining, we train best subnet from scratch.

2.3.1.1 One Shot NAS

In One-Shot (e.g. SPOS) methods, a supernet $\mathcal{N}$ with weights $\mathcal{W}$ is constructed by all candidate operations forms the search space $\mathcal{A}$. The whole pipeline of these methods can be divided into three stages, $i.e.$ supernet Training, Subnet Searching, and Subnet Retraining.

Search Space. The supernet architecture is constructed by a series of candidate operations as shown in Fig. 2.3. A layer contains multiple (N) candidate operations. Every candidate operation in the layer follows the repeating structure of 'Conv-BN-ReLU' with different kernel sizes and expansion ratios (number of channels).

Supernet Training. The supernet $\mathcal{N}$ is trained by sampling a single-path architecture $a \in \mathcal{A}$ based on a sampling policy at each iteration. In the single path architecture search method, only a single candidate operation in each layer is activated. Then the weights of the sampled architecture, denoted by $\mathcal{W}_a$, are optimized by normal network training, $i.e.$ back-propagation.

Since the accuracy of subnet with weights inherited from the supernet should be highly predictive on the validation set, the supernet training often requires hundreds of epochs.

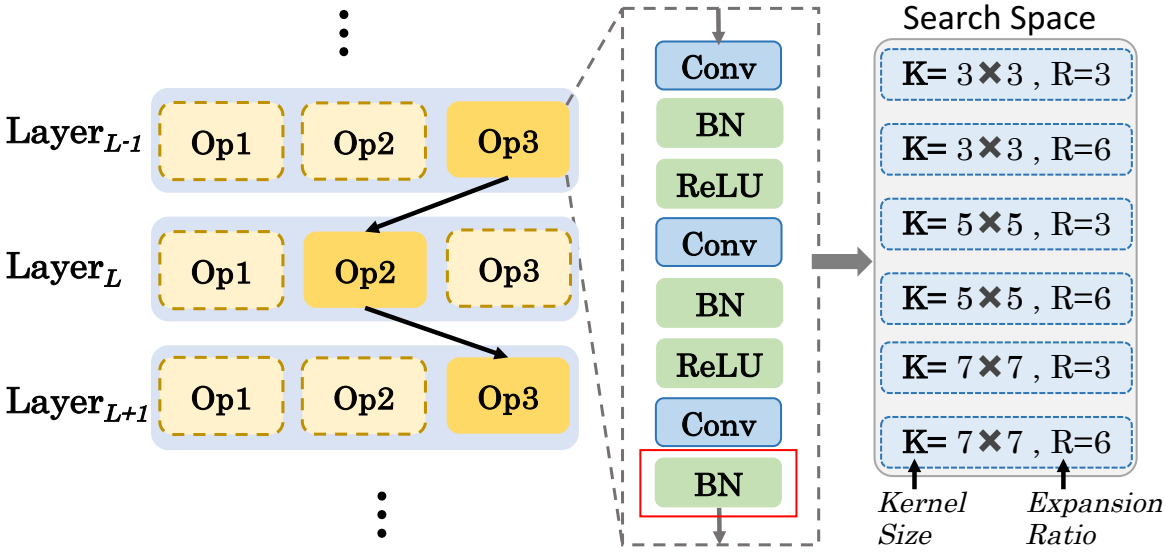


FIGURE 2.3: The single-path based search space. Only one operation is activated in each layer during network forwarding. We focus on the most popular search space for mobile setting network searching (Tan and Le 2019; Wu et al. 2019) as the shown in right. The candidate operation consists of a series of Conv-BN-ReLU and ends with Batch Normalization(BN). We search the optimal kernel size and expansion ratio of convolution layers in an 'Op'. There are totally 6 different candidates of kernel size and expansion ratio in each 'Op'.

Subnet Searching After training the supernet, the next step is to find the optimal architecture with the best performance. In SPOS, the accuracy on validation set is used for evaluating the subnet performance. The optimal subnet is selected according to the subnet accuracy on the validation set. To get a reliable searching result, thousands of subnets are needed to be evaluated.

Subnet Retraining In the retraining stage, the K subnets found at the subnet searching stage with the highest accuracy are retrained. They are then evaluated on the validation set and the subnet with the highest accuracy is chosen as the final optimal subnet.

2.3.1.2 Differentiable NAS

In this work, we also evaluate the performance of a widely-used method, Differentiable Architecture Search (DARTS) (Liu et al. 2019c), as a baseline. DARTS defines the final

network structure as a series of L cells, each of which is represented as a directed acyclic graph of N nodes. The task of DARTS is to determine both the optimal set of nodes and the interconnections among these nodes within each cell. Fig. 2.6 shows the search space and search process of one cell. We denote the search space as $\mathcal{A}$, which consists of different candidate functions $o(.)$. Each node j is connected with their previous node i with an edge E_{ij} which includes several operations ($o \in \mathcal{A}_{ij}$) weighted by architecture parameters α_o^{ij}. The input of node j is a set of all previous operations weighted by α:

$$x_j = \sum_{i<j} \sum_{o \in \mathcal{A}_{ij}} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{A}_{ij}} exp(\alpha_{o'}^{(i,j)})} o(x_i), \tag{2.1}$$

where $\alpha_{o'}^{(i,j)}$ denotes the architecture parameter of operation o' which enables the information flow from node i to node j. Every pair of nodes can be connected or not, so there is a special $zero$ operation in DARTS to indicate if two nodes are connected with each other.

Different from one-shot NAS, DARTS (Liu et al. 2019c) trains supernet parameters and architecture parameters alternately during the supernet training stage. The supernet parameters θ and the architecture parameters α are trained in a bilevel way:

$$\min_{\alpha} \mathcal{L}_{val}(\theta^*(\alpha), \alpha), \tag{2.2}$$

$$s.t.\ \ \theta^*(\alpha) = \arg\min_{\theta} \mathcal{L}_{train}(\theta, \alpha), \tag{2.3}$$

where $\mathcal{L}_{val}$ and $\mathcal{L}_{train}$ are the training loss on the validation and training dataset, respectively. After training, the operation with the highest architecture parameter will be selected as the optimal operation to construct the final subnet. The subnet retraining is the similar with SPOS (Guo et al. 2020b). For more details, we refer readers to DARTS (Liu et al. 2019c).

2.3.1.3 Batch Normalization Layer

Batch Normalization (BN) layer has been used in network pruning (Liu et al. 2017b; You et al. 2019; Kang and Han 2020), which is a good evaluation of channel importance. Given

the input x^{in} of BN layer, the output x^{out} is calculated through:

$$z = \frac{x^{\text{in}} - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}}, \tag{2.4}$$

$$x^{\text{out}} = \gamma \cdot z + \beta, \tag{2.5}$$

where ϵ is a small positive value for numerical stability, $\hat{\mu} \equiv \mathrm{E}\left[x^{\text{in}}\right]$ and $\hat{\sigma}^2 \equiv \mathrm{Var}\left[x^{\text{in}}\right]$ are means and variations calculated across mini-batches. The scaling parameter γ and bias parameter β are learnable parameters in BN layers to affine the normalized features z.

2.3.2 Algorithm Overview

Our method can speed up both one-shot NAS method and differentiable NAS method. To be clear, we first show how to apply our method to a one-shot NAS method (Guo et al. 2020b) and then describe its transformation to a differentiable NAS method (Liu et al. 2019c).

The pipeline of our proposed NAS method for SPOS (Guo et al. 2020b) is shown as Fig. 2.2. We follow the three stages in one-shot methods, $i.e.$ supernet training, subnet searching, and subnet retraining. In supernet training stage, the supernet containing all candidate operations are randomly initialized. Only BN layer parameters are updated through standard forward-backward training, while the other parameters of the supernet are fixed after initialization (Section 2.3.4). In subnet searching stage, subnets are sampled and evaluated based on our BN indicator (Section 2.3.3). In the subnet retraining stage, the best subnet chosen in the subnet searching stage is retrained. To make our BN indicator applicable on more operations and frameworks, we further propose a pseudo BN strategy (Section 2.3.5) with which the BN-indicator is competent to control network depth and attune to the differentiable NAS method (Liu et al. 2019c) (Section 2.3.6).

In the following, we start from the second stage (Subnet Searching). The order of the following description is consistent with the order of our exploration in this direction.

2.3.3 Subnet Searching with BN Indicator

Given the trained supernet, we need to evaluate the performance of sampled subnets in the Optimal Subnet Searching stage. We utilize BN parameters to evaluate the performance of candidate operations.

Change of Denotation for BN layer. Different from channel pruning, we focus on the operation-level outputs instead of channel-level. Take the c-th channel of activated operation o_l output (C channels in total) in layer l as the example, the denotation for the BN layer need to be changed correspondingly. Eqn. (2.4) can be rewritten as follows for the c-th channel of operation o_l:

$$z_c = \frac{x_c^{\text{in}} - \hat{\mu}_c}{\sqrt{\hat{\sigma}_c^2 + \epsilon}}, \tag{2.6}$$

$$x_c^{\text{out}} = \gamma_c \cdot z_c + \beta_c, \tag{2.7}$$

where symbol with subscript c represents the parameters in the c-th channel as the definition in Eqn. (2.4). Assume the normalized features in z follow the normal distribution $N(0, 1)$, a smaller scaling parameter γ means a smaller magnitude of BN layers output x^{out}. Since the output of a channel with smaller magnitudes contribute less for whole network (Liu et al. 2017b), we can treat the scaling parameter γ as the importance of the channel.

BN Indicator for An Operation. When we evaluate the n-th ($n = 1, \dots N$) candidate operation $o_{n,l}$ from the l-th layer ($l = 1, \dots L$), its BN indicator $S_{o_{n,l}}$ is calculated as follows:

$$S_{o_{n,l}} = \frac{1}{C} \sum_{c=1}^{C} \left| \gamma_c^{o_{n,l}} \right|, \tag{2.8}$$

where $\gamma_c^{o_{n,l}}$ is the learned parameters of c-th channel in chosen candidate operation $o_{n,l}$. A candidate operation has many CONV, BN, and RELU layers. During the forward-propagation of an operation, features are normalized several times and the final outputs are only determined by the last scaling parameters. Thus we utilize only the last BN layer of each building operation as shown by the red box in the right side of Fig. 2.3 to indicate the performance of the candidate operation. BN indicator needs the last layer of each operation to be a BN layer,

so it is not directly applicable to search for models (such as pre-activation ResNes) where BN layers are placed at the beginning of the ops. However, most existing NAS methods apply similar search space as ours, such as recently published works (Tan and Le 2019; Wu et al. 2019; You et al. 2020; Guo et al. 2020b; Chu et al. 2021; Wan et al. 2020). The BN-indicator can be directly applied to those methods to reduce the computation cost.

BN-based Indicator for An Architecture. Assuming that there are L search layers in the supernet, we randomly sample the candidate operation $o_{a_l,l}$ from l-th layer to construct the subnet architecture $a = [o_{a_1,1}, ..., o_{a_l,l}, ..., o_{a_L,L}]$. The estimated BN score of the subnet $\mathcal{N}_a$ is calculated by

$$S_{\mathcal{N}_a} = \sum_{l=1}^{L} S_{o_{a_l,l}}. \tag{2.9}$$

Searching Architecture using BN-based Indicator. Through calculating the BN score of the subnet, we can estimate the subnet performance without evaluating it on the validation set and the searching stage can be formulated as

$$a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} S_{\mathcal{N}_a}, \tag{2.10}$$

$$s.t.\ \ FLOPs(a) < Constraint. \tag{2.11}$$

For searching an optimal subnet, we randomly sample subnets $\mathcal{N}_a$ under the FLOPs constraint and evaluate them based on our BN-based indicator. The optimal subnet is the one with the highest BN score $S_{\mathcal{N}_a}$. Accuracy on validation dataset is a common metric for evaluating subnets in most exsiting NAS methods, while BN-indicator is used in our BN-NAS for evaluating subnets.

2.3.4 Training Only BN Layer

Previous work (Frankle et al. 2020) shows that only training BN layers can still improve the expressive ability of DNN. Since only BN parameters, instead of subnet accuracies, are used during the subnet searching stage, we can only train BN layers instead of the whole

Algorithm 1 BN-based one-shot NAS

Inputs: supernet $\mathcal{N}$ representing Search space $\mathcal{A}$, Subnet sampling policy on the search space $P(\mathcal{A})$, Training epoch T, Sampling subnet number N_s for searching, Training set $\mathcal{T}_{train}$, FLOPs Constraint F
Output: Searched Model.

1) Training:
for $epoch \in 0, 1, \ldots, T$ **do**
  Train supernet through Sampling Policy $P(\mathcal{A})$ on training set $\mathcal{T}_{train}$.
end for
2) Searching:
Sample N_s subnets under Constraint F and evaluate them based on our BN-based indicator via Eqn. (2.8) and (2.9).
Choose the architecture a^* with highest score as the searching result, Eqn. (2.10).
3) Retraining:
Train the optimal architecture a^* from scratch on training set $\mathcal{T}_{train}$ and get the trained searched model M_{a^*}.

Return: M_{a^*}

supernet in the supernet training stage. Specifically, only BN parameters are updated during the back-propagation.

The time required for training supernet by our design is 8% ($10\% \times 80\% = 8\%$) of the time required by SPOS. The reduction in training time comes from two aspects, the fewer epochs (10%) and training BN only (further 80%).

1. Fewer epochs. The original SPOS method needs to train supernet for 100 epochs while our method needs only 10 epochs (equal to 10% of original training time).

2. Training BN only. When we train the supernet, we fix the parameters of all convolutional and fully-connected layers. Only the scaling and bias parameters of BN layers are trained through forward-backward propagation. Although the gradients of freezing parameters are calculated during backward propagation, these calculated gradients will not be stored or used for updating. Thus, it will be faster than training all parameters. Through only training the BN layers of the supernet, the time for training supernet can be saved by about 20% (equal to 80%).

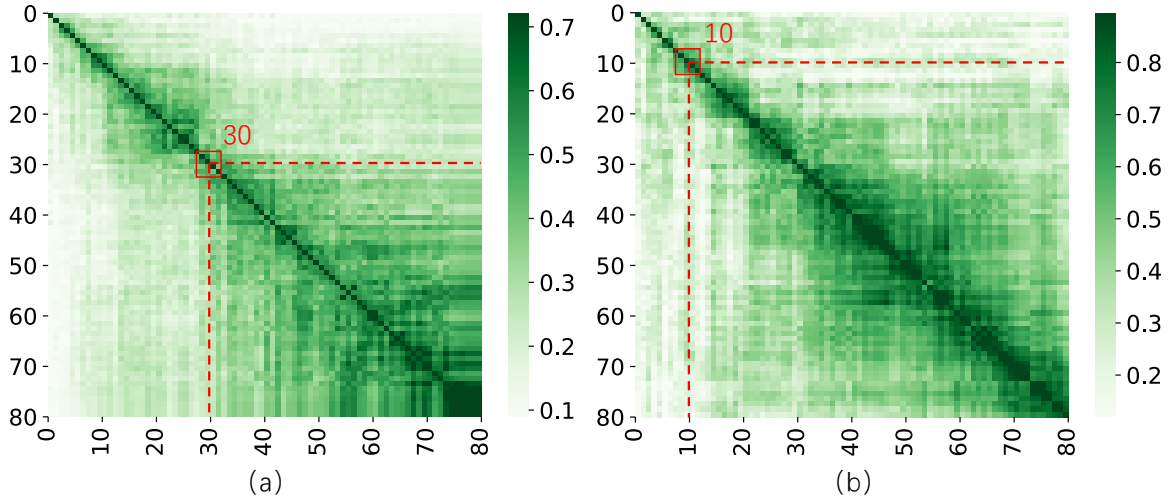


FIGURE 2.4: Early-bird Characteristic when training the supernet for all parameters (a) and training only BN parameters (b). For better visualization, we normalize the similarity comparison between 0 and 1. The i, j-th element in the figure means similarity between i-th and j-th epoch. Deeper color means higher similarity. We treat the inverted normalized L2-distance as the similarity of two masks from two epochs. Higher value (close to 1) indicates a higher similarity and is highlighted with a darker color. Comparing with training all parameters, training on BN will achieve a faster convergence of BN parameters.

2.3.4.1 Analysis on Early-bird Characteristics of BN-based Indicator.

If accuracy is used for evaluating network architectures, reducing the number of epochs or training only BN would have adverse effect at searching stage. Specifically, the rank from the subnets sampled from the supernet trained in this way would have low correlation with the retrained subnets, resulting in the subnet sampled from the under-trained supernet unreliable for evaluating the real performance. This adverse effect is observed by the experimental results in Section 2.4.6. On the other hand, the BN-based Indicator has the early-bird characteristics, which helps us to overcome the potential adverse effect.

Early-bird Characteristics when training all parameters. Inspired by the inspection of BN parameters for each channel in (You et al. 2019), we investigate the early-bird characteristics of our BN-based indicator. In this setting, all parameters are used. Given the trained supernet, we can evaluate sampled subnets through our proposed BN-based indicator. For every epoch during the supernet training, we define a local ranking vector among N candidate operations

in the same layer l according to the candidate operation score $S_{o_{1,l}}, S_{o_{2,l}}, \ldots, S_{o_{N,l}}$. We set the ranking of operation with the highest score as 1 and operation with the lowest score as rank N. By concatenating the L local rank vectors, we can map the trained supernet of an epoch to a ranking vector with size $N \cdot L$. For two ranking vectors from two different training epochs of a supernet, we calculate the L_2 distance of the two ranking vectors. We visualize the distances among different epochs and find that BN parameters in our supernet training show a similar characteristic as BN in network pruning (You et al. 2019). Fig. 2.4(a) shows the pairwise ranking vector distance matrices (80×80) of the supernet training. We can find from Fig. 2.4(a) that the similarity between the rank vector at the 30th epoch and the rank vector at the 80th epoch is high. And the rank vector tends to be stable after around the 30th epoch. This means we can get the optimal architecture information at around the 30th epoch. Therefore, the BN parameters at early training stages are already useful for indicating network performance.

Early-bird Characteristics when training BN only. When we only train BN layers parameters, Fig. 2.4(b) shows that the 'rank vector' tends to become stable much earlier (at about the 10th epoch) than that for training all parameters in Fig. 2.4(a) (becoming stable at about the 30th epoch). This means we can use BN-parameters at a very early training stage to find the optimal architecture. We conjecture the reason for faster convergence is that when freezing other parameters and training only BN parameters, the BN parameters try to fit the label with fixed parameters instead of changing parameters, which makes the BN parameters converge much more earlier. Thus, we can further shorten the training stage through training only BN parameters by one-tenth. For clarity, the whole pipeline of our proposed BN-NAS is shown in Algorithm 1.

Summary of the Early-bird Characteristics. From the results in Fig. 2.4, we find that: 1) BN-indicator helps the ranking to be stable and facilitates training using fewer epochs; 2) training BN only drives the early-bird characteristics to appear at earlier epochs and facilitates us to train the supernet using much fewer epochs.

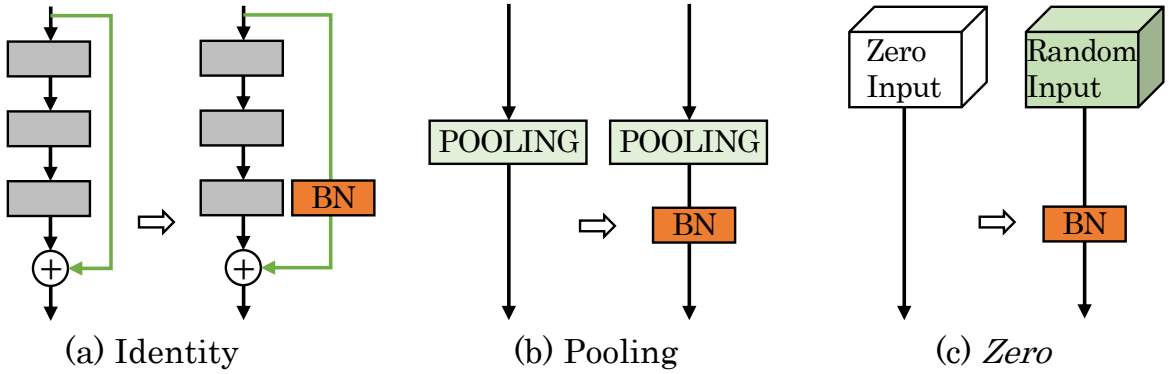


FIGURE 2.5: Illustrations of applying pseudo BN layers (shown in orange) to identity operation (a), pooling operation (b) and the specific $zero$ operation (c) in DARTS (Liu et al. 2019c).

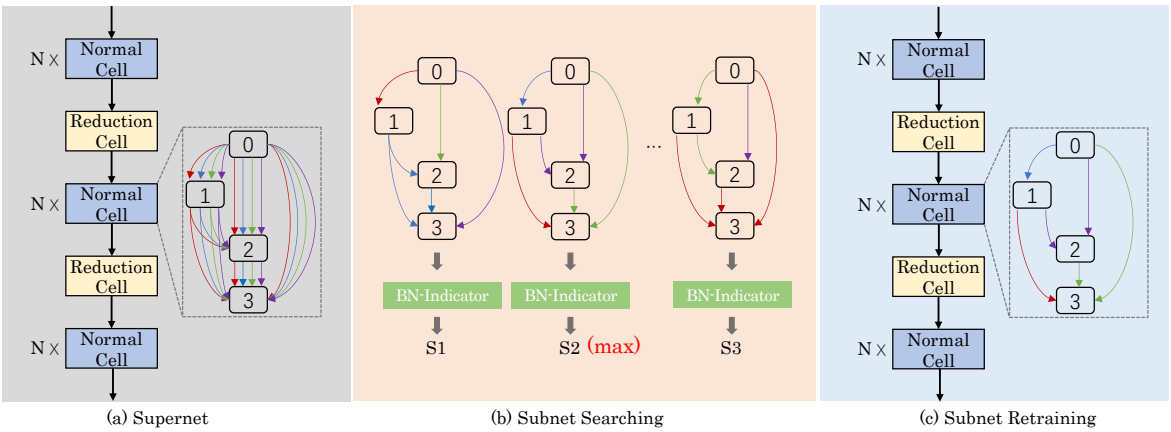
2.3.5 Pseudo BN Strategy

The above-mentioned BN indicator can only be applied to the operations with BN layers as shown in Fig. 2.3. The limitation hinders our BN indicator to be used in some general operations, like identity operation, pooling operation and $zero$ operation in DARTS (Liu et al. 2019c). To solve the above problem, we propose the pseudo BN strategy which adds a pseudo BN layer at the end of those operations without BN layers. Fig. 2.5 shows three examples of applying pseudo BN strategy to identity operation, pooling operation and $zero$ operation in DARTS (Liu et al. 2019c). For the first two operations, we directly add a BN layer at the end of the operations. For the last operation, we will show the details in Section 2.3.6.

TABLE 2.1: Search Space with and without identity operations. ORI_SPACE denotes the search space without identity operations. EXP_SPACE represents the search space with identity operations which is the same as that in (Hu et al. 2020).

	kernel size	expansion ratio	block number
ORI_SPACE	{3, 5, 7}	{3, 6}	17
EXP_SPACE	{3, 5, 7}	{3, 6}	0-21

Expanded Search Space. With the proposed pseudo BN strategy, we can add identity operations with pseudo BN layers to our search space. Here, we name the search space without identity operations as original search space (ORI_SPACE in Table 2.1) and the search spaceh with identity operations as expanded search space (EXP_SPACE in Table 2.1).



(a) Supernet (b) Subnet Searching (c) Subnet Retraining

FIGURE 2.6: (a) shows the supernet of our method on DARTS (Liu et al. 2019c). (b) shows the subnet searching process after the supernet training. We adopt Evolutionary Algorithm (EA) (Guo et al. 2020b) for subnet searching. We first randomly sample subnets with FLOPS constraints

For the original search space, we follow the search space in (Hu et al. 2020) , which is composed of MobileNetV2 blocks with kernel size {3,5,7}, expansion ratio {3,6}. Since our BN-based indicator without pseudo BN strategy acts on the BN layer, we do not involve identity operations. We follow the searched depth result of SPOS in (Hu et al. 2020) and search other operations during network architecture searching. Since the search space is shrunk, the result of SPOS has been improved. Our experiments are based on the improved version. For the FairNAS search space in (Hu et al. 2020), there are no identity operations.

In expanded search space, the candidate operations about kernel size and expansion ratio are the same as those in the original search space. We add an identity operation to each block for block skipping. Then, the block number of the whole network ranges from 0 to 21. Compared with original search space, there are candidate subnets with different depths in expanded search space.

Searching Process. During supernet training, the pseudo BN layers are trained together with normal BN layers. After training, we utilize BN parameters from either pseudo or normal BN layers to evaluate the importance of different operations and subnets. During subnet retraining, the pseudo BN layers are removed in the selected subnets. In summary, the pseudo BN layers only exist in the supernet training and subnet searching process.

2.3.6 BN Indicator for DARTS

Our BN indicator together with pseudo BN strategy can be seamlessly applied to all operations in the search space of DARTS, except for the specific $zero$ operation. For the $zero$ operation, DARTS (Liu et al. 2019c) sends zero inputs to the operation during supernet training. The importance of a $zero$ operation determine if the connection between two nodes is necessary. While, if we add a BN layer directly at the end of $zero$ operation, the BN layer will learn nothing because of the zero input. Therefore, we replace zero inputs to random inputs for the $zero$ operation. The value of the random inputs are so close to zero that their information can be ignored. The random inputs act similarly with the zero inputs, propagating no information to the $zero$ operation, but it enables the pseudo BN layer to learn something. The illustration of $zero$ operation is shown in Fig. 2.5.

Search Space. For experiments on DARTS (Liu et al. 2019c), we follow the search space for convolutional cells by adding the pseudo BN layers to pooling, identity and zero operations. The search space $\mathcal{A}$ includes the following candidate operations: separable convolutions with kernel size $\{3, 5\}$, dilated separable convolutions with kernel size $\{3, 5\}$, 3×3 average pooling with pseudo BN layers, 3×3 max pooling with pseudo BN layers, identity with pseudo BN layers, and $zero$ with pseudo BN layers. The final network is constructed with stacking multiple cells as shown in Fig. 2.6. There are two kinds of cells in the network, including the normal cell and the reduction cell. All normal cells share architecture with each other and the case is the same for reduction cells. Each cell consists of seven nodes. The output node in a cell is the channel-wise concatenation of all intermediate nodes. More details about the search space can be found in DARTS (Liu et al. 2019c).

With our BN indicator, the architecture parameters α^{ij} in Eqn. (2.1) becomes unnecessary, so we removed it from the supernet training and subnet searching process. For an intermediate node j after removing architecture parameters, the input is a set of all previous operations:

$$x_j = \sum_{i<j} \sum_{o \in \mathcal{A}_{ij}} o(x_i), \tag{2.12}$$

Algorithm 2 BN-based differentiable NAS

Inputs: supernet $\mathcal{N}$ representing Search space $\mathcal{A}$, Training epoch T, Training set $\mathcal{T}_{train}$, FLOPs Constraint F
Output: Searched Model.

0) Preparing:
Add pseudo BN layers to identity, pooling and $zero$ operations.
1) Training:
for $epoch \in 0, 1, \ldots, T$ **do**
Train supernet through Eqn. (2.13) on training set $\mathcal{T}_{train}$.
end for
2) Searching:
Sample N_s subnets under Constraint F and evaluate them based on our BN-based indicator via Eqn. (2.8) and (2.9).
Choose the architecture a^* with highest score as the searching result, Eqn. (2.10).
3) Retraining:
Train the optimal architecture a^* from scratch on training set $\mathcal{T}_{train}$ and get the trained searched model M_{a^*}.

Return: M_{a^*}

where $\mathcal{A}_{ij}$ denotes the search space from node i to node j. Compared with Eqn. (2.1), there is no operation weights controlled by architecture parameters α^{ij} in Eqn. (2.12).

Search Process. We train only BN parameters θ_{BN} of the supernet during the supernet training stage through:

$$\theta^*_{BN}(\alpha) = \arg\min_{\theta_{BN}} \mathcal{L}_{train}(\theta_{BN}, \alpha) \tag{2.13}$$

where $\mathcal{L}_{train}$ are the training loss on the training dataset. All other parameters are random initialized and fixed during training. There is no need to use a validation dataset for architecture parameter training in a bilevel way like Eqn. (2.2), so our method can simiplify the training pipeline of DARTS (Liu et al. 2019c). Besides, only training BN parameters can greatly reduce the time costs of supernet training.

After getting the trained supernet, we utilize the BN-indicator to calculate the importance score of each candidate operation through Eqn. (2.8). Simply selecting the operations with the highest importance score may make the selected subnets have higher FLOPs than the searched model in DARTS (Liu et al. 2019c). For fair comparisons, we utilize the searching method (shown in Section 2.3.3) in one-shot method to get optimal subnets with acceptable

TABLE 2.2: Comparison details between our method '-Ours' and the baseline models on ImageNet dataset (Deng et al. 2009). 'Epoch' is the supernet training epoch. 'Parameters' shows if 'ALL' parameters or only 'BN' parameters are trained during supernet training process. 'Val' means using validation dataset for subnet searching. 'Search Depth' shows if the depth of candidate subnets are different in the search space.

	Top1 ACC	FLOPs (M)	Epoch	Training Parameters	Subnet Search Cost	Total Search Cost	Search Data	Search Depth
SPOS	75.73	470	100	ALL	1 GPU Day	11 GPU Day	Val	
SPOS-Ours	75.67	470	10	BN	0.14s on CPU	0.8 GPU Day	None	
SPOS-RD	75.27	469	100×10	ALL	1×10 GPU Day	11×10 GPU Day	Val	✓
SPOS*	75.33	465	100	ALL	1 GPU Day	11 GPU Day	Val	✓
SPOS*-Ours	75.76	474	10	BN	0.14s on CPU	0.8 GPU Day	None	✓
FairNAS	74.07	325	150	ALL	1 GPU Day	15 GPU Day	Val	
FairNAS-Ours	74.12	326	15	BN	0.14s on CPU	0.8 GPU Day	None	

FLOPs. Then, the selected subnet is retrained with training data. More details can be found in Algorithm 2.

2.4 Experiments

In this work, we demonstrate the effectiveness of our proposed method on a variety of tasks, including image classification, object detection, instance segmentation, and semantic segmentation. Our evaluation encompasses comparison with both the baseline models (as detailed in Section 2.4.1) and state-of-the-art methods (as outlined in Section 2.4.2) on well-established datasets such as CIFAR (Krizhevsky, Hinton et al. 2009) and ImageNet (Deng et al. 2009). Moreover, we also evaluate the performance of our method on the NAS-Bench-201 dataset (Dong and Yang 2020) as described in Section 2.4.3. For the tasks of object detection and instance segmentation, we conduct experiments on the widely used COCO2017 dataset (Lin et al. 2014), which are documented in Section 2.4.4. In Section 2.4.5, we evaluate our method on the Cityscape (Cordts et al. 2016) and ADE20K (Zhou et al. 2019) datasets, specifically for the task of semantic segmentation. All experiments are tested on NVIDIA GTX 1080Ti GPU with the Pytorch framework.

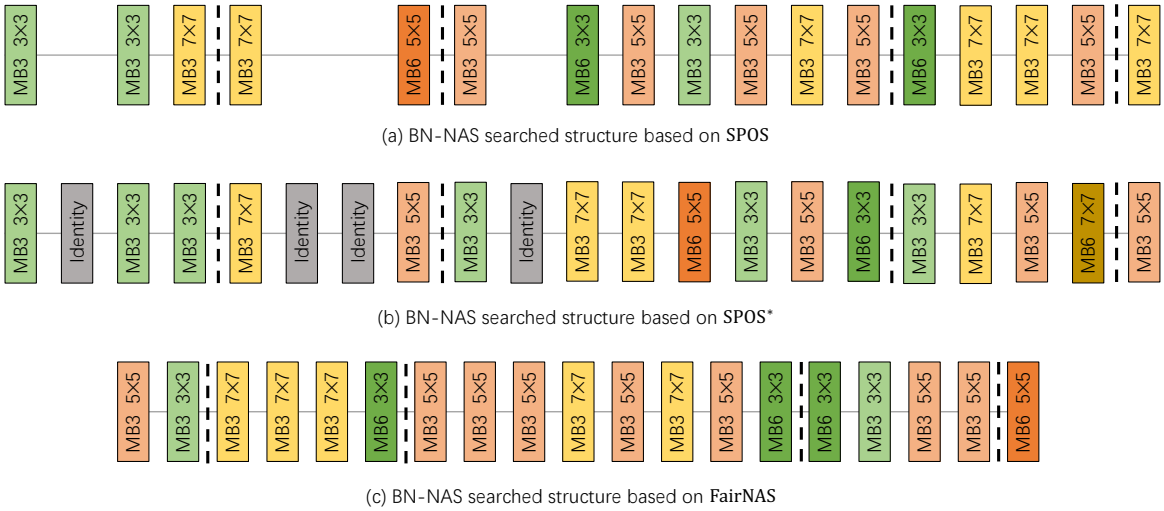


FIGURE 2.7: Our searched model architecture on ImageNet based on SPOS (a) and FairNAS (b). The module MBR $K \times K$ means the MobileNetV2 block with kernel size K and expansion ratio R.

2.4.1 Comparison with baseline models

Settings. We apply our method on three general NAS frameworks, including two basic one-shot NAS methods (SPOS (Guo et al. 2020b) and FairNAS (Chu et al. 2021)) and one differentiable NAS method (DARTS (Liu et al. 2019c)). We evaluate our method on ImageNet (Deng et al. 2009), CIFAR10 and CIFAR100 (Krizhevsky, Hinton et al. 2009). ImageNet dataset consists of 1.28M training samples and 50,000 validation samples. Since we do not need to evaluate our model on the validation set, we only utilize the training samples to train our supernet and the searched subnet. There are 50,000 training and 10,000 testing RGB images in CIFAR10 and CIFAR100, which are equally distributed over 10 and 100 classes. Each training image has a fixed resolution of 32×32.

In the experiments for the two one-shot methods, we maintain consistency in hyper-parameters across all experiments, with the exception of the number of training epochs, and only train the Batch Normalization layers. For network parameters training, we adopt mini-batch Nesterov SGD optimizer with a momentum of 0.9. We utilize the learning rate warm-up technique from 0.2 to 0.8 in the first five epochs and adopt cosine annealing learning rate decay from 0.8 to 0. We train the network with a batch size of 1024 and L2 regularization with weight of

TABLE 2.3: Comparison between DARTS and on CIFAR10, CIFAR100 and ImageNet datasets.

	Training Epoch	CIFAR Parameters	CIFAR10 Top-1	CIFAR100 Top-1	ImageNet Parameters	ImageNet Top-1
DARTS	50	3.4M	97.24	82.46	4.7M	73.3
DARTS-Ours	5	3.3M	97.22	82.36	4.7M	73.5

1e-4. Besides, the label smoothing is applied with a 0.1 smooth ratio. For baseline supernet training, we train 100 epochs and 150 epochs for SPOS and FairNAS. For our BN supernet training, we use one-tenth of baseline epochs, $i.e.$, 10 epochs and 15 epochs. For searched architecture retraining, we train the searched architecture from scratch for 240 epochs. For subnet searching, we follow the EA setting in (Guo et al. 2020b). The population size is 50 and max iterations is 20, sampling $N_s = 1000$ subnets under the FLOPs constraint in total.

In the experiments for the DARTS framework, we deviate from the original implementation by using Batch Normalization (BN) as our indicator, rather than the architecture parameters. We only train the BN parameters on the entire training dataset, without dividing the data into separate subsets for training different parameters. This approach avoids the approximation of architecture gradients in the DARTS framework. For the other parameters, we adhere to the settings used in the original DARTS implementation, including a batch size of 64, optimization with an SGD optimizer with a momentum of 0.9, an initial learning rate of 0.025 that is decreased to zero via a cosine decay schedule without restarts, and a weight decay of 3×10^{-4}. To maintain consistency with the experiments on the one-shot methods, we use one-tenth of the baseline number of epochs, $i.e.$, 5 epochs, in the supernet training for DARTS.

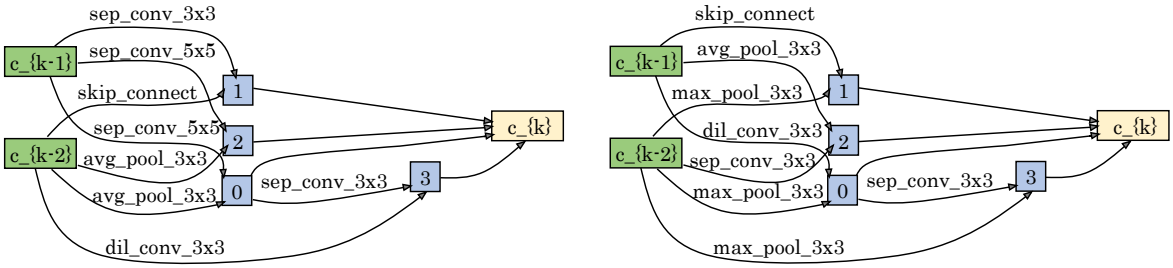
Comparison with One-shot Methods. We evaluate our proposed method on two one-shot NAS frameworks, SPOS (Guo et al. 2020b) and FairNAS (Chu et al. 2021), on ImageNet dataset (Deng et al. 2009) as shown in Table 2.2. In our evaluation on SPOS, we consider two variations of our method, denoted as 'SPOS-Ours' and 'SPOS*-Ours' in the following discussions. 'SPOS-Ours' refers to the architecture obtained through our BN-indicator based search process in the 'SPOS' search space, excluding the identity operations. On the other hand, 'SPOS*-Ours' extends 'SPOS-Ours' by incorporating identity operations

with pseudo BN layers into the search space, resulting in the same search space as that in SPOS. Accordingly, 'SPOS' and 'SPOS*' denote the baseline models without the utilization of our proposed BN-indicator. These models utilize the same search space as 'SPOS-Ours' and 'SPOS*-Ours', respectively. The architecture search results on SPOS and FairNAS are depicted in Fig. 2.4.

According to the results of 'SPOS' $vs.$ 'SPOS-Ours', 'SPOS*' $vs.$ 'SPOS*-Ours', and 'FairNAS' $vs.$ 'FairNAS-Ours', it can be seen that our method significantly shortens the process of Neural Architecture Search (NAS) in two stages: supernet training and subnet searching. During supernet training, benefited from the early-bird characteristic of the proposed BN-based indicator, we significantly reduce the training epochs of supernet, from 100 to 10 for SPOS and 150 to 15 for FairNAS. Besides, we only train BN parameters instead of all parameters, which has an additional 20% speedup for supernet training. During subnet searching, baseline methods adopt EA algorithm to sample 1000 subnets and evaluate each on the validation set. The cost of evaluating 1000 subnets is about 1 GPU day. Our method also samples 1000 subnets but utilize the BN-indicator for subnet evaluation, greatly reducing the evaluation cost from 1 GPU day to 0.14s on CPU.

For the search space without identity operation, 'SPOS-Ours' achieve similar performance with the baseline 'SPOS'. For search space with identity operation, our method 'SPOS*-Ours' even achieve better performance with baseline 'SPOS*', benefiting from our proposed Pseudo BN Strategy. Besides, our method can greatly reduce the search cost from 11 GPU days to 0.8 GPU days. The case is the same for 'FairNAS-Ours'. BN-Indicator helps to reduce the search cost from 16 GPU days to 0.8 GPU days. All above experiments demonstrates the efficiency of our method.

Moreover, our enhanced version, 'SPOS*-Ours', demonstrates comparable results to 'SPOS-Ours' despite having a larger search space. This demonstrates the robustness of our BN-indicator even in extensive search spaces, and confirms the viability of our pseudo BN approach in addressing the limitations of candidate structures. In contrast, 'SPOS*' achieves worse performance than 'SPOS' due to the larger search space.



(a) Normal cell learned based on ours.

(b) Reduction cell learned based on ours.

FIGURE 2.8: Our searched architectures on CIFAR10 based on DARTS, with (a) normal cell or (b) reduction cell.

It is worth mentioning that the depth and block allocation in 'SPOS' was obtained from the searched architecture in (Hu et al. 2020), meaning that an optimal network depth was already given in 'SPOS'. To ensure a fair comparison, we conducted 'SPOS-RD', which is similar to 'SPOS' but with a randomly assigned depth and block allocation. We searched 10 models with different random depth and block allocation, and the best performance among these models was reported as the result of 'SPOS-RD'. The comparison shows that 'SPOS-RD' achieves significantly worse performance compared to 'SPOS', highlighting the advantage of having an optimal depth and block allocation in 'SPOS'. While, our improved version 'SPOS*-Ours' eliminates the need for an optimal depth and block allocation, and is able to achieve similar (or even better) results by searching depth and blocks.

Overall, our method accelerates the one-shot NAS method about 14 times compared with baseline methods while the performance is still comparable. Additionally, the new pseudo BN strategy allows the BN-indicator to be applied in a wider range of scenarios, making 'SPOS*-Ours' a more versatile solution in Neural Architecture Search.

Comparison with DARTS. To evaluate the generalization ability of our method, we apply our BN-indicator and the pseudo BN strategy on a differentiable NAS method, DARTS (Liu et al. 2019c). The comparison results are shown in Table 2.3. Our method only shortens the supernet training time in DARTS, as the optimal architecture can be directly selected through architecture parameters in DARTS. For subnet searching, we adopt the same EA method as the one-shot methods based on our BN-indicator. Since EA with our BN-indicator only costs 0.14s on CPU, the searching cost of both DARTS and DARTS-Ours can be treated as zero.

During supernet training, we reduce the number of supernet training epochs from 50 to 5, resulting in a tenfold increase in speed as compared to the original DARTS method. This is a substantial improvement in terms of computational cost and time, making the method more feasible for practical applications. Additionally, in the original DARTS method, the supernet parameters and architecture parameters are optimized alternately, which is computationally expensive. By using our BN-indicator, we only need to optimize the BN parameters during the supernet training stage, which results in an additional 10% increase in speed.

In conclusion, the combination of our BN-indicator and pseudo BN strategy has demonstrated its ability to accelerate the DARTS method by approximately 11 times, without any drop in performance. This highlights the potential of our method to be adopted in various applications where computational efficiency is a crucial concern. The detailed architectures can be found in Fig.2.8.

2.4.2 Comparison with state-of-the-art methods

Settings. In this section, we adhere to the experimental setting described in Section 2.4.1. The performance of the models searched via the proposed methods in Section 2.4.1 is evaluated and reported. In addition, we integrate Squeeze-and-Excitation (SE) blocks (Hu et al. 2018b) into the searched models, following the SPOS (Single-Path One-Shot) framework, in order to enhance the model's capacity in capturing informative feature representations. The models with SE blocks are named as 'SPOS-Ours+SE' and 'SPOS*-Ours+SE'.

Results. Table 2.4 shows the comparison results of our method with state-of-the-art (SOTA) methods. Compared with the manual designed networks, like ResNet50 (He et al. 2016), MobileNetV2(1.4x) (Sandler et al. 2018) and ShuffleNetV2(2x) (Ma et al. 2018) and EfficientNet-B0(+SE) (Tan and Le 2019), all searched models based on our method achieves better performance with fewer FLOPs. Besides, we compared our method with NAS methods, including NASNet-A (Zoph et al. 2018), AmoebaNet-A (Real et al. 2019), SNAS(mild) (Xie et al. 2019), DARTS (Liu et al. 2019c), PDARTS (Chen et al. 2019b), CARS-G (Yang et al. 2020b), ProxylessNAS(GPU) (Cai et al. 2019), FBNet-C (Wu et al. 2019), FairNAS (Chu

TABLE 2.4: ImageNet classification results of our method and SOTA. Searching on a small dataset ($i.e.$ CIFAR (Krizhevsky, Hinton et al. 2009)) will significantly reduce the search cost. Transferring these methods directly to ImageNet may cause a massive increase in search cost. Some methods even cannot be applied to search on ImageNet due to computational cost. '+SE' denotes adding the Squeeze-and-Excitation strategy in (Hu et al. 2018b).

	Top1-ACC (%)	Params (M)	FLOPs (M)	Search Cost (GPU days)	Search Method	Search Dataset
ResNet50	75.3	25.6	4100	-	manual	-
MobileNetV2(1.4x)	74.7	6.9	585	-	manual	-
ShuffleNetV2(2x)	74.9	7.4	591	-	manual	-
EfficientNet-B0(+SE)	76.3	5.3	390	-	grid search	ImageNet
NASNet-A	74.0	5.3	564	2000	RL	CIFAR
AmoebaNet-A	74.5	5.1	555	3150	evolution	CIFAR
SNAS(mild)	72.7	4.3	522	1.5	gradient	CIFAR
DARTS	73.3	4.7	574	4	gradient	CIFAR
PDARTS	75.6	4.9	557	0.3	gradient	CIFAR
CARS-G	74.2	4.7	537	0.4	evolution	CIFAR
ProxylessNAS(GPU)	75.1	7.1	465	8.3	gradient	ImageNet
FBNet-C	74.9	5.5	375	9	gradient	ImageNet
FairNAS	74.07	4.2	325	16	evolution	ImageNet
SPOS	75.73	5.9	470	11	evolution	ImageNet
SPOS*	75.33	5.8	465	11	evolution	ImageNet
SPOS-Ours	75.67	5.4	470	**0.8**	evolution	ImageNet
SPOS-Ours+SE	76.78	7.6	473	**0.8**	evolution	ImageNet
SPOS*-Ours	75.76	5.6	474	**0.8**	evolution	ImageNet
SPOS*-Ours+SE	76.82	7.8	478	**0.8**	evolution	ImageNet
FairNAS-Ours	74.12	3.7	326	**0.8**	evolution	ImageNet
DARTS-Ours	74.18	4.7	585	**0.3**	evolution	CIFAR

et al. 2021) and SPOS (Guo et al. 2020b). Our method surpasses the existing SOTA NAS methods, regardless of the gradient-based method (Proxyless) or evolution-based (CARS-G) method. In terms of search cost, our method is comparable to the methods that perform architecture search on CIFAR and then transfer the architecture to ImageNet. Furthermore, our method requires significantly less search cost compared to the methods that directly perform architecture search on ImageNet. For instance, compared to EfficientNet-B0, which employs a grid search approach, our method requires less than one-tenth of the search cost. This is due to the fact that the grid search approach in EfficientNet-B0 requires training a large number of models on the full ImageNet dataset, which is computationally expensive compared to many evolution-based methods.

TABLE 2.5: Performance comparison on different datasets in the NAS-Bench-201 benchmark.

Methods	Search Time (GPU hrs)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		Val	Test	Val	Test	Val	Test
optimal	N/A	91.61	94.37	73.49	73.51	46.77	47.31
RSPS	2.6	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS	2.2N	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
SETN	7.9N	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90+4.07
GDAS	6.6N	90.00±0.21	93.51±0.31	71.14±0.27	70.61±0.26	41.70±1.26	41.84+0.90
Ours	0.18	89.64±0.41	93.15±0.22	69.87±0.33	70.22±0.17	43.27±2.25	43.37±1.72

2.4.3 Comparison on NAS-Benchmark

Settings. In order to rigorously evaluate the performance of our method, we conduct experiments on the NAS-Bench-201 (Dong and Yang 2020). NAS-Bench-201 is a widely recognized benchmark in the field of Neural Architecture Search, which comprises 15625 architectures within a cell-based search space. The performance of these architectures is evaluated on the CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets using the train-from-scratch protocol. In our experiments, we follow the experimental setup of DARTS from NAS-Bench-201, with the exception of reducing the training epochs from 50 to 5. To ensure the reliability of our results, we run the experiments 5 times with different random seeds. The performance is then reported as the mean and standard deviation of the results obtained from the 5 runs. This provides a robust evaluation of the performance of our method, and eliminates the impact of random fluctuations on the results. By conducting experiments on NAS-Bench-201 and reporting the results with mean and standard deviation, we aim to provide a comprehensive evaluation of our method and demonstrate its effectiveness in the field of Neural Architecture Search.

Results. We evaluate our method on NAS-Bench-201 in order to obtain the optimal architecture and compare its performance with several state-of-the-art methods, including RSPS (Li and Talwalkar 2020), DARTS (Liu et al. 2019c), SETN (Dong and Yang 2019b), and GDAS (Dong and Yang 2019c). The performance comparison is reported in Table 2.5 for the CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets. Additionally, we also report the search cost for each of the methods. The N in the search time refers to the cost of searching for N different models with varying sizes. Our method outperforms RSPS, DARTS, and SETN

on all three datasets, demonstrating its superiority in terms of accuracy. In comparison with GDAS, our method shows slightly inferior performance on the CIFAR-10 and CIFAR-100 datasets, but achieves 1.57% and 1.53% higher performance on the ImageNet-16-120 Val and Test datasets, respectively. Moreover, our method requires significantly less search cost compared to the other methods, with a search cost of only 0.18 GPU hours. This is due to the fact that our method trains the supernet only once and performs architecture search for different models with varying sizes at a nearly zero cost.

2.4.4 Detection and Instance Segmentation on COCO

Settings. In addition to image classification, we also aim to demonstrate the transferability of our BN-NAS method to object detection and instance segmentation tasks. To do so, we evaluate the performance of our models on the MSCOCO 2017 dataset, which consists of 118K training images and 5K validation images. To evaluate the performance of our BN-NAS model on object detection, we use the searched model SPOS*-Ours+SE from Section 2.4.2 as the backbone of the EfficientDet (Tan et al. 2020) model. The backbone is pretrained on the ImageNet dataset, similarly to other general backbones models. We follow the training protocol outlined in EfficientDet (Tan et al. 2020) and utilize the same detection head as used in (Tan et al. 2020). For the instance segmentation task, we adopt the Mask-RCNN-FPN framework (He et al. 2017) and simply replace the backbone with our searched model.

Results on object detection. The results of our object detection experiment are presented in Table 2.6. In this experiment, we conduct a comparison between our search model and a number of state-of-the-art models. All the above models are implemented within the same framework. These models include ShuffleNetV2, MobileNetV2, ResNet18, and EfficientNet. Our results indicate that, despite having a similar level of computation in terms of FLOPs as the model presented in EfficientDet-D0 (Tan et al. 2020), our searched model outperforms it by 1.82 mAP. Furthermore, when compared to other manually designed light networks, our searched model exhibits superior performance while also having significantly fewer FLOPs. This demonstrates the efficacy of our search method in discovering high-performance and computationally efficient object detection models. Additionally, it highlights the potential

of our searched model for transfer learning, as the model discovered through our method demonstrates good generalization ability to other tasks.

TABLE 2.6: Performance of our searched model and some SOTA light models on COCO dataset. Our methods achieve comparable performance with EfficientDet-D0 with much less search cost.

backbone	FLOPs (B)	mAP
ShuffleNetv2	14	27.6
MobileNetV2	8	31.7
ResNet18	21	32.2
EfficientDet-D0	2.5	33.46
Ours	2.7	35.28

Results on instance segmentation. We evaluate the performance of our models on the object detection and instance segmentation tasks, utilizing the Mask-RCNN-FPN framework. The results of these evaluations are presented in Table 2.7, where we compare our models against two benchmark models: EfficientNet and ResNet18. Our results demonstrate that our models outperform EfficientNet in terms of detection and instance segmentation accuracy, while having a comparable model size. Furthermore, we observe a substantial improvement in performance compared to ResNet18, despite having a much lighter model. This highlights the effectiveness of our proposed models in achieving a good balance between accuracy and model efficiency, and their potential for practical applications in real-world scenarios. These results contribute to the ongoing efforts in the field of computer vision towards developing more efficient and effective models for object detection and instance segmentation.

2.4.5 Semantic Segmentation on CityScapes and ADE20K

Settings. In addition to object detection and instance segmentation, we also evaluate our model on the semantic segmentation task using two well-established datasets: CityScapes (Cordts et al. 2016) and ADE20K(Zhou et al. 2019). The CityScapes dataset contains 5000 high-resolution images with pixel-level annotations for 19 semantic classes, including the void label. This dataset is widely used for evaluating semantic segmentation models and has been extensively used by the computer vision community. ADE20K is a large-scale semantic

TABLE 2.7: Performance comparison between different backbones on the object detection and instance segmentation tasks.

Backbone	Object Detection					
	AP	AP50	AP75	APs	APm	All
Ours	39.43	61.55	42.18	21.98	42.42	54.07
EfficientNet	37.73	60.23	40.12	20.90	40.84	51.60
ResNet18	33.46	54.48	35.49	18.72	36.37	43.61
Backbone	Instance Segmentation					
	AP	AP50	AP75	APs	APm	All
Ours	35.74	58.11	37.74	17.97	38.75	51.04
EfficientNet	34.72	56.81	36.72	17.43	37.69	49.54
ResNet18	30.68	51.35	32.22	14.98	33.41	42.39

segmentation dataset that includes 25K images with annotations for 150 semantic categories. To evaluate our model on this dataset, we utilize the LR-ASPP framework (Howard et al. 2019) in the MMSeg project (Contributors 2020). This framework is known for its ability to effectively handle the large number of categories present in the ADE20K dataset.

Results. In order to assess the performance of our models on the semantic segmentation task, we conduct a comparison against two state-of-the-art light models, EfficientNet (Tan and Le 2019) and MobileNetV3 (Howard et al. 2019), on the CityScapes and ADE20K datasets. The results of these evaluations are summarized in Table 2.8. Our results indicate that our models outperform both EfficientNet and MobileNetV3 in terms of semantic segmentation accuracy on both datasets. This demonstrates the effectiveness of our models in handling the complex and challenging task of semantic segmentation, while maintaining a high level of computational efficiency.

TABLE 2.8: Performance comparison between different backbones on the CityScapes and ADE20K datasets.

Backbone	CityScapes			ADE20K		
	aAcc	mIoU	mAcc	aAcc	mIoU	mAcc
MobileNetV3	94.88	68.69	77.26	74.07	25.68	33.45
EfficientNet	94.36	66.23	74.85	72.83	23.72	32.07
Ours	95.21	71.16	79.11	74.46	26.45	34.33

2.4.6 Ablation Study

In this section, we design experiments to show the effectiveness of BN-indicator (in Section 2.4.6.1) and show the similar correlation relationship between the BN-indicator score and retrain accuracy compared with SPOS (Guo et al. 2020b) (in Section 2.4.6.2). Besides, we also evaluate the influence of different initialization methods on our method.

2.4.6.1 Indicators

Our BN-indicator is used for evaluating subnet during the searching process. Most existing NAS methods utilize the model accuracy on validation dataset to evaluate subnet, which is denoted as Acc-indicator here. Besides, we also randomly sampled five subnets from the supernet and choose the subnet with the highest accuracy as the random basline, as shown in red dotted line in Fig. 2.9.

Training all parameters for 100 epochs. We train the supernet for 100 epochs and search the subnet based on BN-indicator ('All/BN/100' in Fig. 2.9) and Acc-indicator ('All/Acc/100' in Fig. 2.9). The searched subnets with these two indicators perform similarly, showing that BN-indicator is on par with accuracy as the Acc-indicator when all parameters are trained for enough training epochs. However, both training the supernet for 100 epochs and evaluating model accuracy leads to much computation cost.

Training all parameters for 30 epochs. We reduce the training epochs of supernet from 100 to 30 and test the performance of two indicator under this setting, as shown in Fig. 2.9. With less training epochs, the performance of subnets from BN-indicator ('All/BN/30' in Fig. 2.9) perform better than those from Acc-indicator ('All/Acc/30' in Fig. 2.9). It shows that longer training epoch of supernet is essential for Acc-indicator but not important for BN-indicator. The searched model using Acc-indicator ('All/Acc/30' in Fig. 2.9) performs only better than random baseline (red dotted line in Fig. 2.9) by a small margin of 0.1%. The performance drop on the low correlation is caused by the incomplete training. The accuracy of subnets from supernet which only trained for 30 epochs cannot represent the retraining accuracy precisely, causing the searched model performs near random baseline. On the other

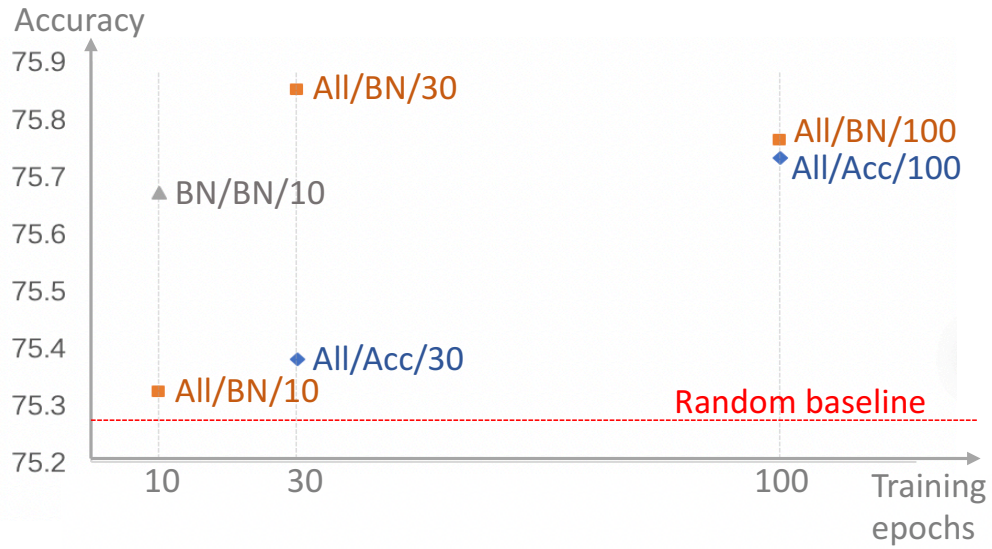


FIGURE 2.9: The accuracy of searched architectures with different training settings on ImageNet. 'All/BN/k' means training all parameters of the supernet for k epochs and using BN-indicator to find optimal subnets. In 'All/Acc/k', Acc-indicator is used instead of BN-indicator. In 'BN/BN/k', only BN parameters of the supernet is trained. Red dotted line shows accuracy of random baseline.

hand, the BN parameters of the supernet shows Early-bird characteristics. As shown in Fig. 2.4(a), it shows stronger correlation between the supernets trained for 30 epochs and 80 epochs, considering only their BN values. The Early-bard characteristics explains well why the proposed BN-indicator still keeps good performance for 30 epochs.

Training all parameters for 10 epochs. When further reducing the training epochs from 30 to 10, our BN-indicator cannot keep good performance, as shown in Fig. 2.9. The reason is that the BN parameters are not trained well at 10 epoch during the supernet training if all parameters are trained, as show in Fig. 2.4 (a). Training all parameters leads to inconsistent convolution parameters at different training epochs, hindering the BN parameters from converging faster. Inspired by this insight, we try to reduce the required training epoch of supernet by only training the BN parameters. As shown in Fig. 2.4 (b), the Early-bird characteristic appears even earlier at about 10 epoch when we only train BN parameters in the supernet. By training BN-only, our BN-NAS returns to its excellent performance during subnet searching, as shown by 'BN/BN/10' in Fig. 2.9.

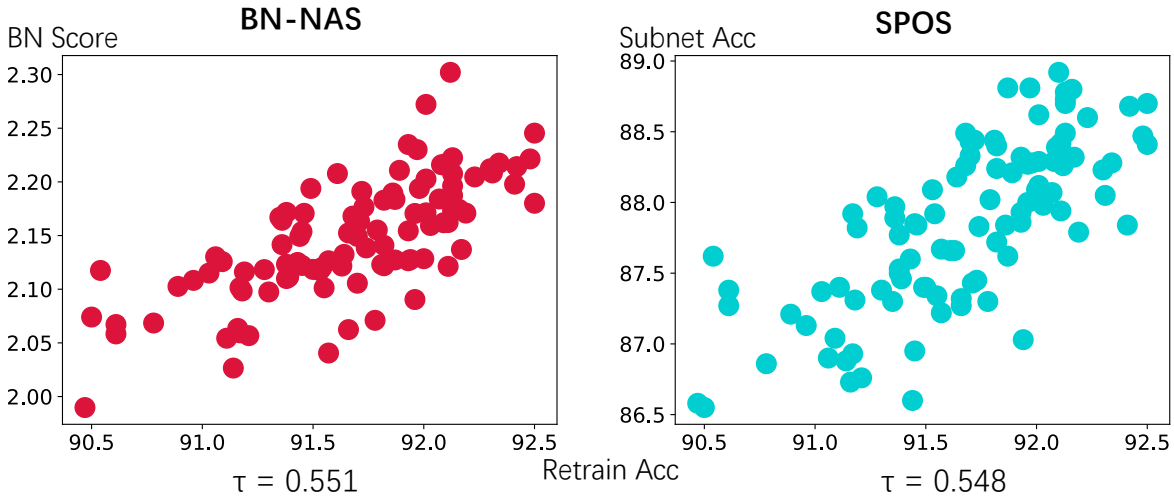


FIGURE 2.10: Model correlations for Ours and SPOS.

2.4.6.2 Correlation with retrain accuracy

For one-shot NAS methods, a well-known problem is the low performance consistency of different subnets. Indicator plays an important role to keep high performance consistency in one-shot NAS. To evaluate the effectiveness of our proposed BN-indicator, we conduct the correlation experiments on CIFAR10 dataset. We follow the search space in (Hu et al. 2020) and train the supernet for 600 epochs. Then we randomly sample 100 architectures and retrain them from scratch. We utilize Kendall Tau τ metric to show the correlation between the BN-score obtained by Eqn. (2.9) and the retrain accuracy of sampled models. We also show the correlation between validation accuracy and retrain accuracy of models which are sampled based on Acc-indicator (SPOS). As shown in Fig. 2.10, our method achieves similar Kendall Tau τ as SPOS, which means the proposed BN-indicator has a good indication ability as the Acc-indicator used in SPOS. Our experiments in achieving similar accuracy of the searched model also support this conclusion.

2.4.6.3 Initialization method

Since our proposed BN-NAS only trains BN parameters with other parameters fixed, network initialization methods may influence the whole NAS process. Targeting at the influence of different initialization methods on our BN-NAS, we investigate Kaiming's initialization

(Norm and Uniform) and another typical initialization of Xavier. From Table 2.9, we can see that the final accuracy is similar among three initialization methods. Such results demonstrate the robustness of proposed BN-NAS on different initialization methods.

TABLE 2.9: Top-1 accuracy comparison with different initialization.

	Top1(%)	FLOPs(M)
Kaiming(uniform)	75.62	473
Kaiming(norm)	75.67	470
Xavier(norm)	75.58	468

2.5 Summary

NAS has greatly boosted the SOTA methods with deep networks in computer vision. However, existing NAS methods are time-consuming. In this chapter, we propose a novel BN-based indicator to efficiently evaluate the performance of subnets selected from the supernet, drastically accelerating the searching process for NAS. Thanks to the Early-bird character, we can train the supernet by only training the BN layers, further reducing supernet training time. Our extensive experiments validate that the proposed BN-NAS can decrease the whole time consumption for one-shot NAS.

CHAPTER 3

GLiT: Neural Architecture Search for Global and Local Image Transformer

---

We introduce the first Neural Architecture Search (NAS) method to find a better transformer architecture for image recognition. Recently, transformers without CNN-based backbones are found to achieve impressive performance for image recognition. However, the transformer is designed for NLP tasks and thus could be sub-optimal when directly used for image recognition. In order to improve the visual representation ability for transformers, we propose a new search space and searching algorithm. Specifically, we introduce a locality module that models the local correlations in images explicitly with fewer computational cost. With the locality module, our search space is defined to let the search algorithm freely trade off between global and local information as well as optimizing the low-level design choice in each module. To tackle the problem caused by huge search space, a hierarchical neural architecture search method is proposed to search the optimal vision transformer from two levels separately with the evolutionary algorithm. Extensive experiments on the ImageNet dataset demonstrate that our method can find more discriminative and efficient transformer variants than the ResNet family (e.g., ResNet101) and the baseline ViT for image classification.

3.1 Motivations and Contributions

Convolutional Neural Networks (CNN) -based architecture (e.g., ResNet (He et al. 2016)) contributes to the great success of deep learning in computer vision tasks (Ren et al. 2015; Chen et al. 2018a; Li et al. 2019) for past several years. By stacking a set of CNN layers, CNN-based models can achieve larger receptive filed and perceive more contextual information on

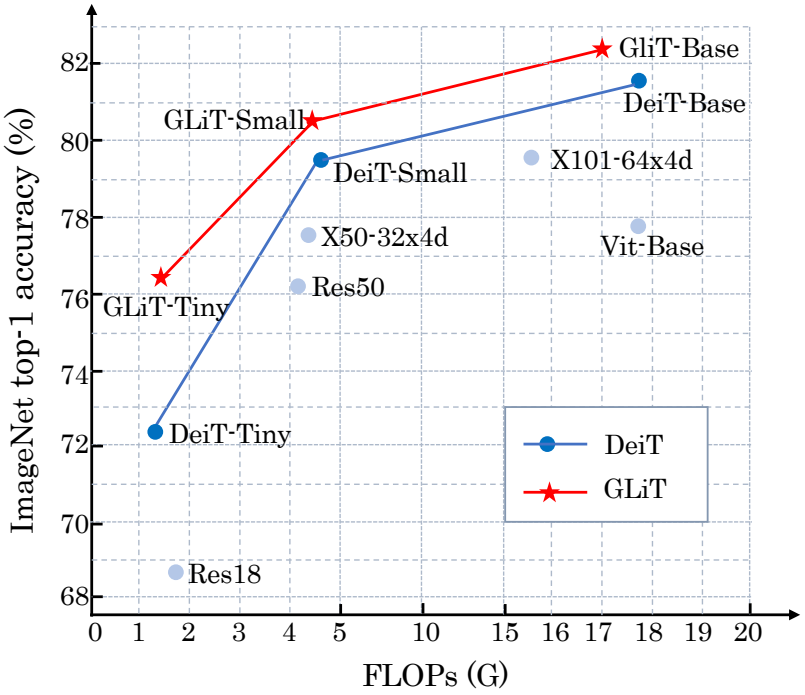


FIGURE 3.1: Top-1 accuracy (y-axis) and FLOPs (x-aixs) for different backbones on ImageNet. GLiT is our method.

scarifies of the efficiency. Driven by the great success of transformer (Vaswani et al. 2017) in Natural Language Processing(NLP) tasks, there are increasing interests in the computer vision community to develop more efficient architectures based on the transformer (Dosovitskiy et al. 2021; Touvron et al. 2021; Carion et al. 2020; Zhu et al. 2020b) which can manipulate the global correlations directly. Among these works, vision transformer (ViT) is a representative one (Dosovitskiy et al. 2021) as it does not rely on the CNN-based backbone to extract features and solely relies on self-attention modules in transformer to establish global correlations among all input image patches. While ViT achieves impressive performance, if extra training data is not used, ViT still has lower accuracy than the well-designed CNN models such as ResNet-101 (He et al. 2016). To further exploit the potential of transformer in image recognition tasks, DeiT (Touvron et al. 2021) uses teacher-student strategy for distilling knowledge to the transformer token. These two methods rely on the original transformer architecture but neglect potential gap between NLP tasks and image recognition tasks in architecture.

In this chapter, we argue that there are unignorable gaps between different kinds of data modalities (e.g., image and text), leading to the disparities between different tasks. Thus, directly applying the vanilla transformer architecture to other tasks may be sub-optimal. It is natural that there exists better transformer architectures for image recognition. However, hand-designing such an architecture is time consuming since there are too many influential factors to be considered. On one hand, Neural Architecture Search (NAS) has achieved great progress in computer vision tasks (Guo et al. 2020b; Liu et al. 2019c; Bello et al. 2017). It can automatically discover an optimal network architecture without manual try-and-error. On the other hand, computer vision community still has not investigated NAS for transformers.

Based on the above observations, we intend to discover a better transformer architecture by NAS for specific tasks, e.g., the image classification task in this work.

There are two key factors when designing NAS for transformer in vision tasks, a well-designed search space that contains candidates with good performance and an efficient searching algorithm to explore the search space.

A naïve search space would only contain the architecture parameters in the transformer architecture, such as the feature dimension for query and value, the number of attention heads in the Mutli-Head Attention (MHA) mechanism, and the number of MHA blocks. However, the search space does not consider two factors. First, the self-attention mechanism in transformer cost quadratic memory and computational burden w.r.t the number of input tokens (Zhu et al. 2020b) during the inference stage. Second, local recurrence in human visual system (Kietzmann et al. 2019; Larsson and Heeger 2006) is not realized in the transformers like ViT and DeiT. Inspiration from the local recurrence in human visual system leads to the success of convolutional layer, locally connected layers for computer vision tasks (LeCun, Bengio et al. 1995). Although theoretically possible, it is hard to model sparse local correlations by the vaninlla self-attention mechanism (e.g., a fixed-size neighbor tokens) in practice.

Considering the two factors mentioned above, we expand the search space of the vanilla transformer by introducing a locality module to the MHA. The locality module only operates

on the nearby tokens, requiring fewer parameters and computation. The locality module and the self-attention module are alternative, which is searched by NAS to decide which one is used. We rename the expanded MHA block as the global-local block as it can capture both global and local correlations among the input tokens. According to our experiments (Table 3.1), the flexibility of the transformer in capturing global and local information is an important factor for the final performance.

Introducing global-local block should be effective, but poses challenge to the searching algorithm. The NAS algorithm for our search space should 1) discover the optimal distribution of locality modules and self-attention modules in each global-local block, and 2) find the detailed settings of both locality modules and self-attention modules by searching the module parameters. Such a search space is extremely huge (10^{18} times of the possible choices in (Guo et al. 2020b) and 10^{12} times of the possible choices in (Liu et al. 2019c)), which makes it challenging for existing NAS methods like SPOS (Guo et al. 2020b) in geting an ideal result. To deal with the problem mentioned above, we propose a Hierarchical Neural Architecture Search method to find the optimal networks. Specifically, we first train a supernet that contains both locality modules and self-attention modules, and determine the high-level global and local sub-modules distribution with evolutionary algorithm. Then, the detailed architecture within each module are searched in a similar manner. Compared with traditional searching strategies, the proposed hierarchical searching can stabilize the searching process and improve the searching performance.

Fig. 3.1 shows that our searched Global Local image Transfomer (GLiT) achieves up to 4% absolute accuracy increase when compared with the state-of-the-art tranformer backbone DeiT on ImageNet.

To summarize, the main contributions are as follows:

- So far as we know, concurrent with (Chen et al. 2021d), we are the first to explore better transformer architecture by NAS for image classification. Our work finds a new transformer variant that achieves better performance than ResNet101 and ResNeXt101 using the same training setting without pre-training on extra data.

- We introduce locality modules to the search space of vision transformer model, which not only decreases the computation cost but also enables explicitly local correlation modeling.
- We propose a Hierarchical Neural Architecture Search strategy, which can handle the huge searching space in the vision transformer efficiently and improves the searching results.

3.2 Related work

Transformers in Vision. The vision community has witnessed bloom of interest and attention in combining transformers with CNN, including DETR (Carion et al. 2020) and Deformable DETR (Zhu et al. 2020b) for object detection, ViT (Dosovitskiy et al. 2021) and DeiT (Touvron et al. 2021) for image classification, and IPT (Chen et al. 2021c) for multi low-level tasks. Different from DETR (Carion et al. 2020) and Deformable DETR (Zhu et al. 2020b), our method does not rely on CNNs for feature extraction. Instead, the whole model is totally based on transformer architecture. Deformable DETR (Zhu et al. 2020b) introduces local mechanism to reduce computation by only attending to small set of key sampling points around a reference. The new local mechanism is not well optimized on GPUs, so training Deformable DETR still needs quadratic memory costs. Differently, our proposed locality module helps to reduce not only the computation but also the memory resources. It is more efficient than the local attention in Deformable DETR.

Global and Local Attention in NLP. Transformers based on self-attention technique were proposed in (Vaswani et al. 2017) to replace RNN for sequence learning on machine translation and become state-of-the-art since then. We are inspired by the use of global and local attention in (Gulati et al. 2020; Beltagy et al. 2020) for NLP. Longformer (Beltagy et al. 2020) splits the original global attention with mask global attention and masked local attention for long sequence learning. Our introduction of local attention is inspired by Conformer (Gulati et al. 2020), which combines convolutions with self-attention to build the global and local interactions in Automatic Speech Recognition (ASR) model. However, it is unknown

whether the Conformer for NLP is effective for image recognition. Different from Conformer and Longformer for NLP tasks, we introduce the convolution as the Local Attention in the transformers for the image classification task. Besides, our exploration on searching the distribution of global and local sub-modules in a network by NAS is not investigated in (Beltagy et al. 2020; Gulati et al. 2020).

Global and Local Attention in Vision. Similar as NLP field, global and local attention mechanism is also proved effective in computer vision tasks. SAN (Zhao et al. 2020) proposes pairwise and patchwise attention mechanism for image recognition. (Hu et al. 2018a; Wang et al. 2018) achieve the performance gain from both global and local information. SENet (Hu et al. 2018a) introduces the channel-wise attention in the local connected convolution network. (Wang et al. 2018) utilizes Non-local blocks to capture long-range dependencies in CNNs. Recently, BotNET (Srinivas et al. 2021) replaces the last residual blocks of ResNet through transformer blocks to extract global information. All the above methods manually design attention mechanism to CNNs, while our focus is to introduce the local attention to vision transformers and automatically search for the optimal global-local setting.

Neural Architecture Search. Recently, NAS methods make great progress on the vision tasks (Chen et al. 2021a; Ci et al. 2021; Liu et al. 2021; Zhou et al. 2020; Li et al. 2020b; Liang et al. 2020). Early NAS mothods apply RL (Bello et al. 2017; Zoph et al. 2018) or EA (Real et al. 2019) to train multiple models and update the controller to generate model architectures. To reduce the searching cost, weight-sharing methods are proposed. These methods construct and train the supernet which includes all architecture candidates. Darts (Liu et al. 2019c) proposes a differentiable method to jointly optimize the network parameters and architecture parameters. SPOS (Guo et al. 2020b) proposes a single-path one-shot method, which trains only one subnet from the supernet in each training iteration. After supernet training, the optimal architecture is found through Evolutionary Algorithm (EA). However, due to the memory restriction (Darts (Liu et al. 2019c)) or low correlation problems (SPOS (Guo et al. 2020b)), these two methods cannot handle our search space with too many candidate architectures. We propose the Hierarchical Neural Architecture Search to solve problem caused by huge search space.

NAS has been used to search an optimal architecture for NLP models. AutoTrans (Zhu et al. 2020a) designs a special parameter sharing mechanism for RL-based NAS to reduce the searching cost. (Tsai et al. 2020) proposes a sampling-based one-shot architecture search method to get a faster model. NAS-BERT (Xu et al. 2020) constructs a big supernet including multiple architectures and find optimal compressed models with different sizes in the supernet. Different from the above methods, we focus on NAS for transformer on image classification instead of NLP tasks. Concurrent with our work, (Chen et al. 2021d) proposes Weight Entanglement method to search the optimal architecture for original ViT model. Different from (Chen et al. 2021d), we introduce locality into vision transformer models and propose Hierarchical Neural Architecture Search to handle huge search space.

3.3 Method

We propose Global-Local (GL) transformer and search its optimal architecture. The *GLiT* consists of multiple global-local blocks (Sec. 3.3.1) which are constructed by introducing local sub-modules to the original global blocks, as shown in Fig. 3.2. Based on the global-local blocks, we design the *specific search space* for vision transformer (Sec. 3.3.2), as described in Table 3.2. Accordingly, the *hierarchical neural architecture search* method (Sec. 3.3.3) is proposed for better searching results, with which we first search the high-level global-local distribution and then detailed architecture for all modules, as shown in Fig 3.4.

Similary with other vision transformers (Dosovitskiy et al. 2021; Touvron et al. 2021), the GLiT receives a 1D sequence of token embeddings as input. To handle 2D images, we split each image into several patches and flatten each patch to a 1D token. The features of an image is represented by $F \in \mathbb{R}^{w \times h \times c}$, where c, w and h are the channel size, width and height of the image. We split the image features F into patches of size $m \times m$ and flatten each patch to a 1D variable. Then, $F \in \mathbb{R}^{w \times h \times c}$ is reshaped to $\tilde{F} \in \mathbb{R}^{m^2 \times \frac{cwh}{m^2}}$, which consists of m^2 input tokens. We combine the m^2 input tokens with a learnable class token (shown in green at the 'Input' of Fig. 3.2) and send all $m^2 + 1$ tokens to the GLiT. Finally, the output class token from the last block is sent to a classification head to get the final output.

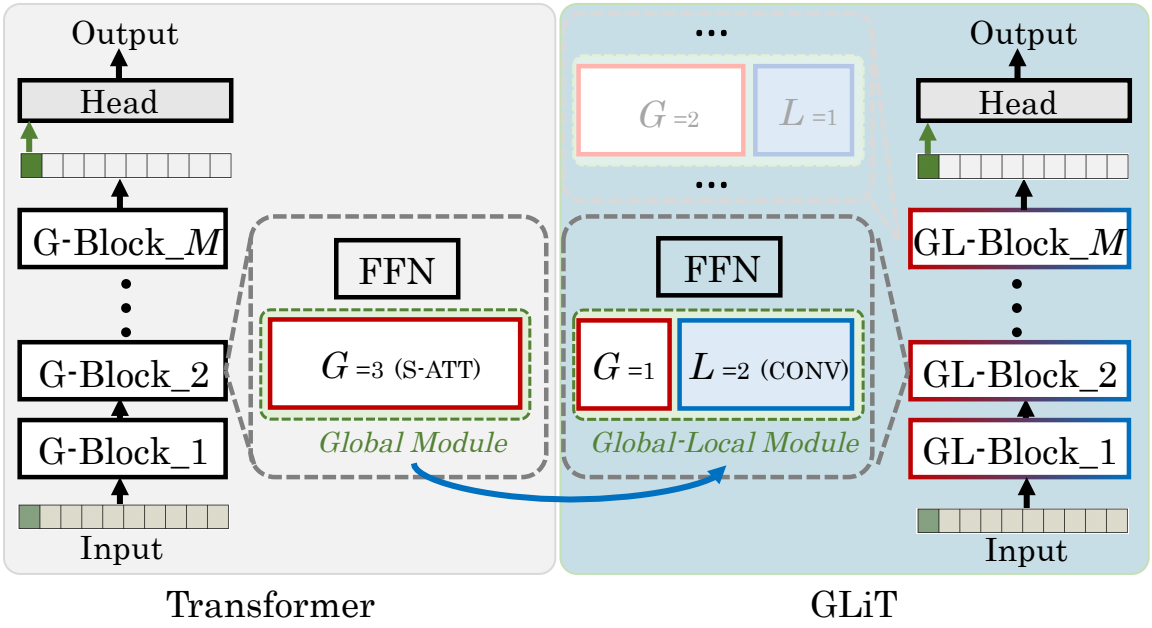


FIGURE 3.2: The construction of GLiT. 'S-ATT' is self-attention head, 'CONV' denotes convolution head, G and L are the numbers of self-attention and convolution heads. The original transformer consists of only global module and Feed Forward module, $i.e.$ the 'FFN' in the figure. We further introduce local sub-module to the global module and get the Global-Local module. GLiT is constructed by M GL blocks. The distribution of global and local sub-modules may be different in different GL blocks. For example, GL-Block_2 in this figure has $G = 1$ global sub-module and $L = 2$ local sub-modules.

3.3.1 Global-local block

There are two kinds of modules in the global-local block, including global-local module (in the green dotted box) and Feed Forward Module (FFN), as shown in the Fig 3.2.

3.3.1.1 Global-local module

Self-Attention as the Global Sub-module. All $m^2 + 1$ input tokens are linearly transformed to queries $Q \in \mathbb{R}^{(m^2+1)\times d_k}$, keys $K \in \mathbb{R}^{(m^2+1)\times d_k}$ and values $V \in \mathbb{R}^{(m^2+1)\times d_v}$, where d_k and d_v are the dimension of features for each token in the query (keys) and value. $d_k = d_v$ in the design of transformer. The global attention is calculated by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \tag{3.1}$$

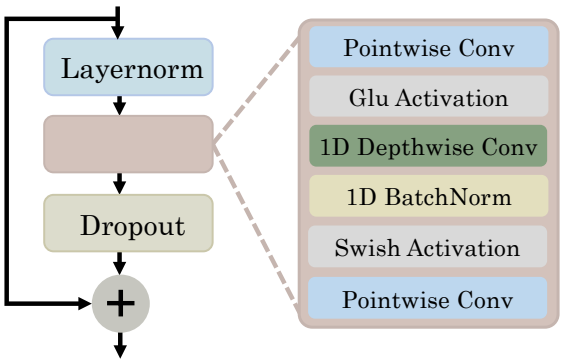


FIGURE 3.3: Convolution layers in the local sub-module.

This module calculates the attention results by considering the relationship among all input tokens, so we named this self-attention head as the global sub-module in this chapter.

The formulation in Eqn. (3.1) is further modified to Multi-Head Attention (MHA) mechanism, where the queries, keys and values are split into N parts along the dimensions, whose outputs are denoted as $head_0, head_1, ..., head_i, ..., head_N$,

$$head_i(Q_i, K_i, V_i) = softmax(\frac{Q_i K_i^T}{\sqrt{d_{head_i}}})V_i, \tag{3.2}$$

where Q_i, K_i and V_i are the i_{th} part of Q, K and V, d_{head_i} is the dimension of each head and is equal to $\frac{d_k}{N}$. The output values of N heads are concatenated and linearly projected to construct the final output.

Convolution heads as Local Sub-module. 1D convolution has been used in NLP tasks (Zhang et al. 2020; Gulati et al. 2020) to model local information. Inspired by the Conformer blocks in (Gulati et al. 2020), we apply 1D convolution to establish local connections, which is named local sub-module in the following description. As shown in Fig. 3.3, one convolution head consists of three convolutional layers, including two point-wise convolutional layers with one 1D depth-wise convolutional layer between them. Each convolutional layer is followed by normalization, activation (such as GLU (Dauphin et al. 2017)) and dropout layers. The first point-wise convolutional layer followed by Glu activation has an expansion ratio E to expand the feature dimension to E times. After that, the 1D depth-wise convolutional

layer with kernel size K does not change the feature dimension. Finally, the last point-wise convolutional layer projects the feature dimension back to the input dimension.

We utilize 1D convolution layer instead of 2D convolution layer in local sub-modules, as it is more suitable for 1D sequence of input tokens. Besides, the $m^2 + 1$ input tokens in GLiT can not be directly reshaped to a 2D array.

Constructing Multi-head Global-Local Module. Given global and local sub-modules, next question is how to combine them. We construct the global-local module by replacing several heads in the MHA with local sub-modules. For example, if there are $N = 3$ heads in the MHA, we can keep one MHA head ($head_0$) unchanged and replace two heads ($head_1$ and $head_2$) with our local sub-module. If all heads in MHA are global sub-modules, then the global-local block degenerates to the transformer block used in ViT (Dosovitskiy et al. 2021) and DeiT (Touvron et al. 2021). In the global-local module, queries, keys and values are only calculated for the heads implemented by global sub-module. For the heads implemented by local sub-module, inputs are directly sent to convolution layers.

Table 3.1 shows the experimental results of evaluating the GLiT with different ratios of the global and local sub-modules in the global-local block. Here, the total head number in each transformer block is 3. The first row with 3 self-attention heads and 0 Conv1d head is the baseline model corresponding the the ViT in (Dosovitskiy et al. 2021). In the 2nd, 3rd and 4th rows, we gradually replace self-attention heads (global sub-modules) with more Convolution heads (local sub-modules). As we can see, the ratio of global and local sub-modules has obvious influence on the performance. Simply replacing all self-attention heads by convolution heads will cause a huge performance drop due to the lack of global information. On the other hand, the network with 1 self-attention head and 2 convolution heads in every global-local module performs the best among all models, improving 1.8% Top-1 accuracy compared with the baseline model. The performance variation with different ratios between self-attention and convolution heads demonstrates that introducing local information brings more performance gains only with proper global-local ratio.

TABLE 3.1: Performance comparisons of different head distributions in DeiT-Tiny model (Touvron et al. 2021) on ImageNet dataset. All blocks utilize the same distribution of heads.

Self-attention head number	Conv1d head number	Acc (%)
3	0	72.20
2	1	72.89
1	2	73.98
0	3	71.02

3.3.1.2 Feed Forward Module

Apart from the global-local module, there is a Feed Forward Module (FFN) in each GL-block to further transform input features. The FFN consists of a Layer Normalization and two fully-connected layers with a Swish activation and dropout layer between them. Mathematically, the FFN $f(X)$ for its input $X \in \mathbb{R}^{m^2 \times d}$ can be represented as:

$$f(X) = \sigma(LN(X)W_1 + b_1)W_2 + b_2, \tag{3.3}$$

where $LN(\cdot)$ denotes Layer Normalization (Ba et al. 2016), $\sigma(.)$ is the Swish activation function, $W_1 \in \mathbb{R}^{d \times d_m}$ and $W_2 \in \mathbb{R}^{d_m \times d}$ are weights of fully-connected layers, $b_1 \in \mathbb{R}^{d_m}$ and $b_2 \in \mathbb{R}^d$ are the bias terms, d and d_m are respectively the feature dimension of input for the first and the second FC layers. We denote $d_z = \frac{d_m}{d}$ as the expansion ratio of FFN.

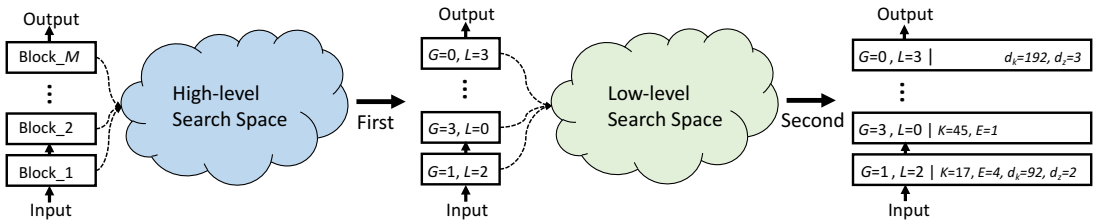


FIGURE 3.4: The framework of Hierarchical Neural Architecture Search. First, we find the optimal distribution of local (L) and global (G) sub-modules in the high-level search space. For example, $L = 1, G = 2$ means 1 local sub-module and 2 global sub-modules in the global-local module. Then, the detailed architecture for all sub-modules is searched in the low-level search space (detailed in Table 3.2).

3.3.2 Search space of the global-local block

The search space of proposed global-local block includes the high-level global-local sub-module distribution and low-level detailed architecture of each sub-module. At the high-level, we aim to search the distribution of convolution and self-attention heads over all global-local blocks. At the low-level, we search the detailed architecture of all sub-modules. Table 3.2 summarizes the high-level and low-level search space implemented in this chapter.

High-level global-local distribution. Suppose there are N_m heads in the mth transformer block $m \in \{1, \ldots, M\}$, we can keep $G \in \{0, \ldots, N_m\}$ self-attention heads unchanged and replace L self-attention heads with convolution heads ($L = N_m - G$), so there are $N_m + 1$ different variations of the global-local distribution in the mth block. The candidate high-level designs for the mth block is $\mathcal{N}_m = [(0, N_m), (1, N_m - 1), ..., (j, N_m - j), ..., (N_m, 0)]$, where $(j, N_m - j)$ denotes $G = j$ self-attention heads and $L = N_m - j$ convolution heads in the global-local module. The high-level search space contains the candidate high-level designs for all M blocks , $i.e.$ $\mathcal{N} = \mathcal{N}_0 \times \mathcal{N}_1 \times ... \times \mathcal{N}_m \times ... \times \mathcal{N}_M$, where $\times$ denotes Cartesian product.

Low-level Detailed architecture. The search space of detailed architecture focuses on four items: the feature dimension d_k of queries (keys, values) in self-attention heads, the expansion ratio d_z of FFN, the expansion ratio E of the first point-wise convolution layer and the kernel size K of the 1D depth-wise convolutional layer in the convolution heads. Table 3.2 lists all the possible choices for d_k, d_z, E, and K. Suppose the total candidate numbers of d_k, d_z, E, K are V_1,V_2,V_3,V_4, we can get the search operation sets $\mathcal{D}_k = [d_k^1, d_k^2, ..., d_k^{V_1}]$, $\mathcal{D}_z = [d_z^1, d_z^2, ..., d_z^{V_2}]$, $\mathcal{E} = [E_p^1, E_p^2, ..., E_p^{V_3}]$ and $\mathcal{K} = [K_d^1, K_d^2, ..., K_d^{V_4}]$. Random selecting one operation from each set can form a candidate global-local block, so there are totally $V_1V_2V_3V_4$ candidates of one global-local block on the low-level.

It should be noted that all convolution heads in one block share the same architecture, similarly for self-attention heads. For block $m \in \{1, \ldots, M\}$, the inner search operation sets for convolution, self-attention and FFN are $\mathcal{E}_m \times \mathcal{K}_m$, $\mathcal{D}_{km}$ and $\mathcal{D}_{zm}$, where $\times$ denotes

Cartesian product. The overall search space for block m is $\mathcal{S}_m = \mathcal{D}_{km} \times \mathcal{D}_{zm} \times \mathcal{E}_m \times \mathcal{K}_m$ and the final search space in the low-level is $\mathcal{S} = \mathcal{S}_0 \times \mathcal{S}_1 \times ... \times \mathcal{S}_m \times ... \times \mathcal{S}_M$.

TABLE 3.2: Search Space for GLiT. 'Local' is the local sub-module, 'Global' is the global sub-module and 'FFN' is the Feed Forward Module. (G, L) denotes the number of global and local sub-modules in each block. K is the kernel size of local sub-module, E is the expansion ratio of local sub-module, d_k is the feature dimension of global sub-module, d_z is the expansion ratio in FFN.

High-Level	(G,L)		(0,3), (1,2), (2,1), (3,0)
Low-Level	Local	K	17, 31, 45
		E	1, 2, 4
	Global	d_k	96, 192, 384
	FFN	d_z	2, 4, 6

Search Space Size. Considering both the high-level distribution and detailed architectures on the low-level, the candidate number of each block is about $(N+1)V_1V_2V_3V_4$. In our search space, different blocks have different high-level distributions and detailed architectures. If a transformer has M blocks, the final search space contains $((N+1)V_1V_2V_3V_4)^M$ candidate networks, which is an extremely huge search space. For our implementation in Table 3.2 for $M = 12$ blocks, $((N+1)V_1V_2V_3V_4)^M \approx 1.3 \times 10^{30}$, which is about 10^{12} times the search space of DARTS (Liu et al. 2019c) and 10^{18} times the search space of SPOS (Guo et al. 2020b). The main-stream fast Neural Architecture Search methods, such as differential (Liu et al. 2019c) and one-shot (Guo et al. 2020b) method, can not work well on this huge search space. For DARTS, the parameters of all candidate networks are trained for each iteration, leading to an unacceptable memory requirements. One-shot NAS method does not have the above problem, because they only select one candidate network during each training iteration. However, the correlation between the retrained subnet and the subnet sampled from the supernet is lower under the huge search space, so the architectures searched using supernet become unreliable. To solve the searching problem on the huge search space, we propose the Hierarchical Neural Architecture Search method to get the optimal network architecture with suitable memory requirements.

3.3.3 Hierarchical Neural Architecture Search.

The Hierarchical Neural Architecture Search method consists of two main stages, as shown in Fig. 3.4. First, we search the optimal distribution $\mathcal{N}^*$ of the global and local sub-modules in each block. Then, we fixed the distribution $\mathcal{N}^*$ and search the detailed architecture $\mathcal{S}^*$ of both the global and local sub-modules.

First Stage. At the first stage, we fixed the low-level detailed architecture parameters d_k, d_z, E and K for global and local sub-modules. The one-shot NAS method SPOS in (Guo et al. 2020b) is applied to search the optimal distribution of global and local sub-modules from the search space $\mathcal{N}$. There are three main steps when the SPOS is applied: supernet training, subnet searching and subnet retraining. In each iteration of supernet training, we 1) randomly sample indices $[j_0, j_1, ..., j_M]$; 2) use these indices to sample a subnet from the supernet, where the number of global and local sub-modules in the M blocks is $[(j_0, N_0 - j_0), (j_1, N_1 - j_1)..., (j_m, N_m - j_m), ..., (j_M, N_M - j_M)]$; and 3) train the subnet. After supernet training, we utilize Evolutionary Algorithm (Guo et al. 2020b) at the subnet searching step to find the top-5 optimal architectures according to validation accuracy. Finally, at the subnet retraining step, we retrain the five networks and choose the architecture with the highest validation accuracy as the output model, where the distribution of global and local sub-modules is $\mathcal{N}^* = [(j_0^*, N_0 - j_0^*), (j_1^*, N_1 - j_1^*)..., (j_m^*, N_m - j_m^*), ..., (j_M^*, N_M - j_M^*)]$.

Second Stage. After obtaining the optimal the distribution of global-local modules in all blocks at the first stage, we fix this distribution and search the detailed architecture of all modules. Similar to the first stage, we adopt SPOS (Guo et al. 2020b) to find the optimal architecture $\mathcal{S}^*$ in the search space. The main difference is changed search space and correspondingly the random index of a block is an array with four elements, instead of a single number j_m. The random index of block m is $(j_m^1, j_m^2, j_m^3, j_m^4)$, which corresponds to the index of $(\mathcal{D}_{km}, \mathcal{D}_{zm}, \mathcal{E}_m, \mathcal{K}_m)$ respectively.

The proposed search method has two main advantages compared with existing NAS methods (Guo et al. 2020b; Liu et al. 2019c). First, the proposed method divides the huge search space into two smaller search spaces. As mention above, the size of original search space

is $((N+1)V_1V_2V_3V_4)^M$. With our proposed search method, the total size of two smaller search space is $(N+1)^M + (V_1V_2V_3V_4)^M$, which is reduced to less than 10^{-7} times the original search space for our implementation in Table 3.2. Second, the size of low-level search space $(V_1V_2V_3V_4)^M$ can be further reduced with a fixed global-local distribution. As shown in Fig. 3.5, after the first searching stage, most blocks in the searched architecture include either global or local sub-modules and only two blocks have both global and local sub-modules. For most blocks, the size of low-level search space is V_1V_2 or V_3V_4, instead of $V_1V_2V_3V_4$. To fix the size of search space for each block, we reduce the search space for the blocks with both global and local sub-modules, by only searching d_z and E for these blocks. With the hierarchical search method, the final search space falls into the effective search space range of existing NAS methods. The significantly reduced search space makes it easier for SPOS in obtaining better model.

3.4 Experiments

We evaluate our GLiT on image classification task. In Section 3.4.2, we compare our searched transformer architectures with DeiT (Touvron et al. 2021), which is a recently published algorithm on vision transformer. In Section 3.4.3, we design more experiments to show the necessity of our search space and search method. All experiments are tested on NVIDIA GTX 1080Ti GPU with the Pytorch framework.

3.4.1 Implementation Details

Dataset. All experiments are conducted on ImageNet, which consists of 1.28M images in $train$ set and 50,000 validation images in $test$ set. We split 50K samples from $train$ set to construct val set. The val set is used for subnet evaluation during the searching process.

Hyper-parameters. We adopt mini-batch Nesterov SGD optimizer with a momentum of 0.9 during the supernet training. We utilize the learning rate 0.2 and adopt cosine annealing learning rate decay from 0.2 to 0. We train the network with a batch size of 1024 and L2 regularization with weight of 1e-4 for 100 epochs. Besides, the label smoothing is applied

with a 0.1 smooth ratio. For subnet searching, we follow the EA setting in (Guo et al. 2020b), which samples $N_s = 1000$ subnets under the FLOPs constraint in total. For the searched model retraining, we follow the training settings in DeiT (Touvron et al. 2021).

3.4.2 Overall Results on ImageNet

We compare the searched transformer with two CNNs (ResNet and ResNeXt) and the state-of-the-art vision transformer DeiT (Touvron et al. 2021). Table 3.3 shows the results under different computational budgets. The results for the existing models, such as R18 (Resnet-18) and R50 (Resnet-50), in Table 3.3 are copied from the results reported in (Touvron et al. 2021). We also use the training setting in (Touvron et al. 2021) and report the results for R18, R50, X50-32x4d (Resnext-50), and X101-64x4d (Resnext-101), with ^{s} followed by these models in Table 3.3 for denoting the same training configurations. Our models achieve better accuracy than all compared networks under similar FLOPS restrictions. For example, our searched model with 1.3G FLOPS restriction achieves 76.3% accuracy score, which is higher than both DeiT-Tiny and ResNet18 (R18) by more than 4 points and 6 points respectively. Our searched models achieves obvious improvement in accuracy from the symphony our two designs: local information and architecture search. The local information brought by Conv1d without proper distribution has limited improvement according to Table 3.1. However, the searched global and local information distribution shows much better performance according to our ablation study and the detailed architecture search will further improve the performance of our GLiT model.

3.4.3 Ablation study

In this section, we conduct experiments to demonstrate the necessity of our searching space and effectiveness of Hierarchical Neural Architecture Searching method. All ablation studies are based on our GLiT-Tiny model on ImageNet using the same training setting as before.

Searching Space. The proposed searching space includes two levels, the global-local distribution and the detailed architecture of all modules. To verify the effectiveness of both

TABLE 3.3: Classification accuracy of different models on ImageNet. 'Acc' denotes the Top-1 accuracy and '^{s}' denotes the models trained using the training configurations in DeiT (Touvron et al. 2021). R18 denotes Resnet-18, X50 denotes Resnext-50.

Models	Params(M)	FLOPS(G)	Acc(%)
R18	11.7	1.8	69.8
R18^s	11.7	1.8	68.5
DeiT-Tiny^s	5.7	1.3	72.2
GLiT-Tiny^s	**7.2**	**1.4**	**76.3**
R50	25.6	4.1	76.1
R50^s	25.6	4.1	78.5
X50-32x4d	25.0	4.3	77.6
X50-32x4d^s	25.0	4.3	79.1
DeiT-Small^s	22.1	4.6	79.6
GLiT-Small^s	**24.6**	**4.4**	**80.5**
X101-64x4d	83.5	15.6	79.6
X101-64x4d^s	83.5	15.6	81.5
Vit-Base	86.6	17.6	77.9
DeiT-Base^s	86.6	17.6	81.8
GLiT-Base^s	**96.1**	**17.0**	**82.3**

levels, we investigate our model with the model searched only on global-local distribution ('Only distribution' in Table 3.4), and the baseline model DeiT-Tiny with human design. The model with only global and local distribution searched performs much better than the baseline model DeiT-Tiny without NAS. It also outperforms the best human-designed architecture with global-local information (73.98% in Table 3.1), improving the accuracy by about 1.5%. The performance gains come from the optimal transformer architecture with proper global-local information distribution searched by our method. After considering the detailed architecture of all modules, the final model ('Ours') further improves the accuracy about 1%, which is significant on ImageNet. Besides, we also compare our search space with that in NLP-NAS (Tsai et al. 2020). The search space in (Tsai et al. 2020) includes the expansion ratio (query, key and value) and MLP ratio. The searched model from the NLP-NAS search space achieves a 73.4% top-1 accuracy on ImageNet, which is 1.2% higher than DeiT but 2.9% lower than ours. Directly using the search space in (Tsai et al. 2020) limits performance improvements. Our search space and search method are more effective, which are our two main contributions. The experimental results in Table 3.4 demonstrate all components in our searching space is essential for an excellent vision transformer.

TABLE 3.4: Performance comparisons of models from different searching spaces on ImageNet. 'DeiT-Tiny' is the baseline model which totally relies on hand-designing. 'Only distribution' is the model searched only on global-local distribution. 'Ours' is the model searched from the complete searching space.

Method	Flops(G)	Acc
DeiT-Tiny	1.3	72.2
Only distribution	1.4	75.4
Ours	1.4	76.3
NLP-NAS	1.4	73.4

Searching Method. We compare the proposed searching method with the baseline NAS method (SPOS) and random search baseline. All searching methods are used in our proposed search space. For fair comparisons, we train the supernet of SPOS for 200 epochs, which is the same as the training epochs in ours. After supernet training, we select 5 architectures and retrain them for SPOS and ours. For the random baseline, we randomly sample and retrain 5 networks. The architecture with the highest retraining accuracy are chosen as the final model. In Table 3.5, our method achieve much better performance than the SPOS method and random search baseline. The performance improvement is from our hierarchical searching method, which reduce the searching space effectively in all stages. For SPOS method, since the supernet is constructed by the huge search space, the optimization of the supernet is different, even with the double training epochs compared with our supernet. Due to the insufficient training, the subnet with lower computation will converge faster and the searched result on the validation set tends to be a model with lower computation as shown in Table 3.5. Not only the model in Table 3.5, but also the other four models in the top-5 models identified by SPOS have small flops of 1.0G, 0.9G, 0.9G, and 0.9G. As a result, the searching result with vanilla SPOS method has performance even worse than random search.

TABLE 3.5: Performance comparisons between SPOS, Random searching baseline, and our searching method on ImageNet.

Method	Flops(G)	Acc
Ours	1.4	76.3
SPOS	0.9	72.7
Random search	1.3	73.3

Methods to introduce locality. We choose Conv1D to introduce the locality into Vision Transformer. However, there are other methods such as restricting self-attention in a local area or using Conv2D. We evaluate these two methods and compare them with ours. For restricting self-attention in a local area, we set the candidate local area sizes as (14, 28, 49). We keep the other settings (including the hierarchical search) fixed for fair comparison. In Table 3.6, the model with local self-attention has only 72.4% top-1 accuracy on ImageNet (much lower than ours 76.3%), possibly due to the lack of communication among different local areas. However, Conv1D in our GLiT model can solve this issue. To use Conv2D in our network, we remove the CLS token and add a global average pooling at the end of the network for classification. The candidate kernel sizes are set as $(3 \times 3, 5 \times 5, 7 \times 7)$. The searched model with Conv2D achieves 76.4% accuracy, which is similar with ours. For fair comparison with our baseline model ViT and DeiT, which utilize the CLS token, we adopt Conv1D in our final models.

TABLE 3.6: Performance comparisons between Self-attention in a local area, Conv2D and our searching method on ImageNet.

Method	Flops(G)	Acc
Local-area	1.4	72.4
Conv2d	1.4	76.4
Conv1d	1.4	76.3

3.4.4 Discussion.

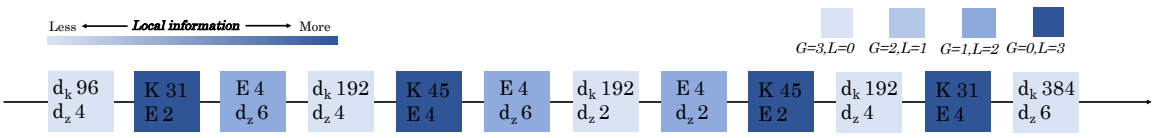


FIGURE 3.5: The architecture of GLiT-Tiny in Table 3.3. Each box represents a global-local block. The darker the color denotes the more local sub-modules in the block. L is the number of local sub-modules in each block. G is the number of global sub-modules.

Searched architecture. Fig. 3.5 shows the searched architecture of GLiT-Tiny (Table 3.3). There are only 25% blocks consists of both global and local sub-modules. Most blocks

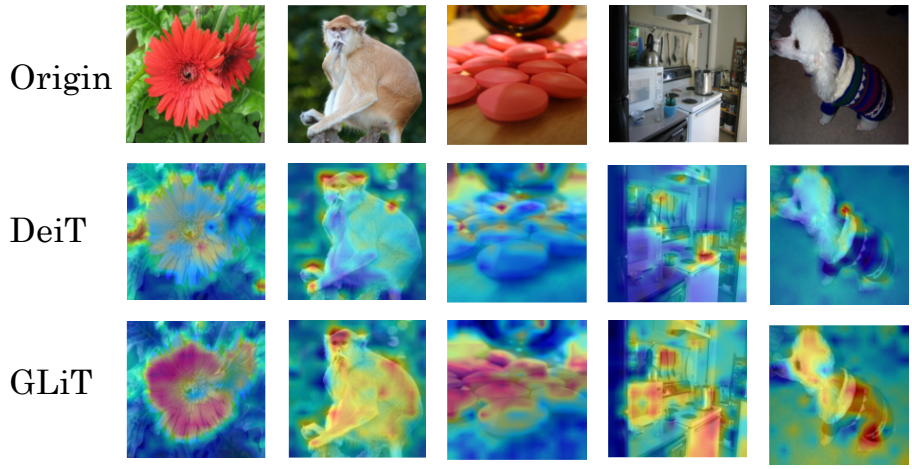


FIGURE 3.6: Visualization of features for DeiT (Touvron et al. 2021) (second row) and our GLiT (third row). Images in the first row are from ImageNet.

contains either global or local sub-modules. Sequential connection between global and local sub-modules may be more necessary than parallel connection. There is no 1D convolution layer with kernel size 17 in the searched architecture. 17 is the smallest value of kernel size in the search space. It shows that too small kernel size is not suitable for locality modules in vision transformers. The searched architecture has a trend of all-local, to local-global mixture, and then back to all-local blocks. This helps local and global features interact through the transformer blocks. This architecture looks like a mechanism similar to the stacked hourglass in (Newell et al. 2016), which has stacks local-global CNNs, where 'local' corresponds to CNN with high-resolution features and 3×3 convolution has smaller receptive fields, while 'global' corresponds to CNN features with lower resolution and a 3×3 convolution looks at more global region of the same image.

Visualization. In Fig. 3.6, we show the visualization of learned features of both DeiT (the second row) and our GLiT (the last row). We calculate the heat maps by reshaping the output tokens to the same size as input images and averaging the reshaped tokens along channels. By reaching a good combination of local and global features, our GLiT focuses on more object regions than DeiT.

3.5 Summary

In this chapter, we exploit better architectures for vision transformers through carefully designing the searching space with local information and the hierarchical searching method. Transformer is applied to vision community not long ago. Its architecture is not well exploited for image recognition. Our method provides a feasible and automatic network design strategy. In addition to showing better performance compared with existing vision transformers, this work will inspire more researches on finding optimal transformer architecture for computer vision tasks.

CHAPTER 4

PSViT: Better Vision Transformer via Token Pooling and Attention Sharing

---

In this chapter, we observe two levels of redundancies when applying vision transformers (ViT) for image recognition. First, fixing the number of tokens through the whole network produces redundant features at the spatial level. Second, the attention maps among different transformer layers are redundant. Based on the observations above, we propose a PSViT: a ViT with token **P**ooling and attention **S**haring to reduce the redundancy, effectively enhancing the feature representation ability, and achieving a better speed-accuracy trade-off. Specifically, in our PSViT, **token pooling** can be defined as the operation that decreases the number of tokens at the spatial level. Besides, **attention sharing** will be built between the neighboring transformer layers for reusing the attention maps having a strong correlation among adjacent layers. Then, a compact set of the possible combinations for different token pooling and attention sharing mechanisms are constructed. Based on the proposed compact set, the number of tokens in each layer and the choices of layers sharing attention can be treated as hyper-parameters that are learned from data automatically. Experimental results show that the proposed scheme can achieve up to 6.6% accuracy improvement in ImageNet classification compared with the DeiT in (Touvron et al. 2021).

4.1 Motivations and Contributions

Deep neural network (DNN) has been the core of the recent popular artificial intelligence (AI) wave. It greatly enhances the state-of-the-art (SOTA) for many tasks in natural language processing (NLP) and computer vision (CV), which are two major application fields of AI.

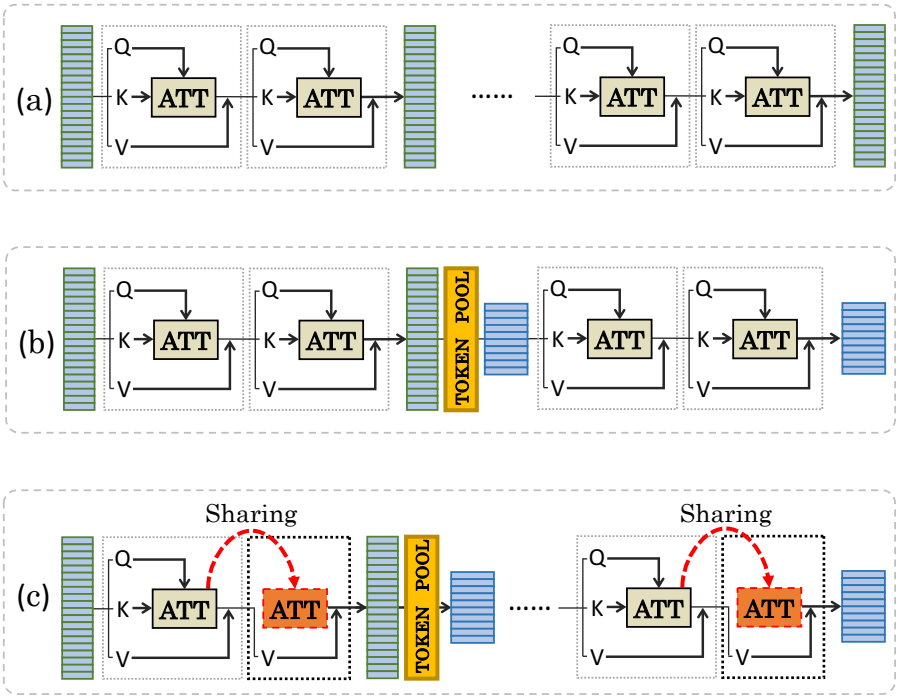


FIGURE 4.1: Token pooling and attention sharing. (a) The original ViT. (b) Token pooling (TOKEN POOL in the figure) for ViT, where the number of tokens (input length) in the next layer can be smaller than the number of tokens in the current layer, and the token dimension per token can increase. (c) Attention map (ATT in the figure) sharing for ViT, where the attention at the previous transformer layer is copied to the current layer.

Among them, transformer (Vaswani et al. 2017; Devlin et al. 2018; Li et al. 2020a; Shi et al. 2021) has found its almost ubiquitous applications in the field of NLP due to its great advantage of long-range capture and parallelism capability compared to the previous prevalent recurrent neural network (RNN).

Recently, transformer is reevaluated by researchers in the field of CV and applied to solve some typical problems such as image generation (Parmar et al. 2018) and image classification (Dosovitskiy et al. 2021; Touvron et al. 2021). The flexible attention mechanism of the transformer in the whole pipeline intrigued the emergence of many following works such as object detection (Carion et al. 2020), image segmentation (Wang et al. 2021a), and video instance segmentation (Wang et al. 2021b). The inspiring achievements of the transformer in the above pioneering works are turning the transformer into a capable alternative to the prevalent convolutional neural network (CNN) in CV field.

In this chapter, we find two redundancies in most vision transformers. Our first observation is that the fixed number of tokens observed by the model produces the redundant features at the spatial level. As shown in traditional CNNs (Zeiler and Fergus 2014), deep neural networks tend to encode low-level information in shallow layers and high-level information such as semantic features in deep layers. A fixed number of tokens may be an improper design for this purpose, since the fixed number of tokens may not be enough for low-level information encoding and too redundant for high-level information encoding.

To better capture the semantic features as well as reducing the computation, it is a common practice to downsample feature resolution in CNN as the layers get deeper. As a counterpart in a transformer, the number of tokens should have more flexibility to adjust itself based on different levels. To achieve this, we propose the attention pooling mechanism for adapting token numbers at different layers.

The second form of redundancy resides in the similarity of the attention maps between adjacent layers. In ViT models, relations among all tokens are built in the transformer encoder according to the input feature embeddings. However, since feature embeddings between the neighboring layers change smoothly, the learned attention between neighboring layers may be similar as Fig. 4.2 shows. The strong similarity between attention at adjacent layers motivates us to share the attention among adjacent layers. On the other hand, the attention takes 43% computation and 33% the number of parameters in the transformer layer. Sharing attention saves the computation required for obtaining attention.

Specifically, the number of tokens in each transformer layer and the setting for attention sharing will affect the performance of transformer. While it is intuitive for tuning token number and attention sharing to be effective, it is challenging to reach an optimal design for vision tasks. Instead of relying on heuristic human design for getting the optimal design, we can resort to a more principled way by leveraging the recent success of Automated Machine Learning (AutoML) (He et al. 2021). Specifically, we first construct a set of possible choices, called search space in AutoML, which consists of the possible design combinations for token pooling and attention sharing. For token pooling, we search its location in the network.

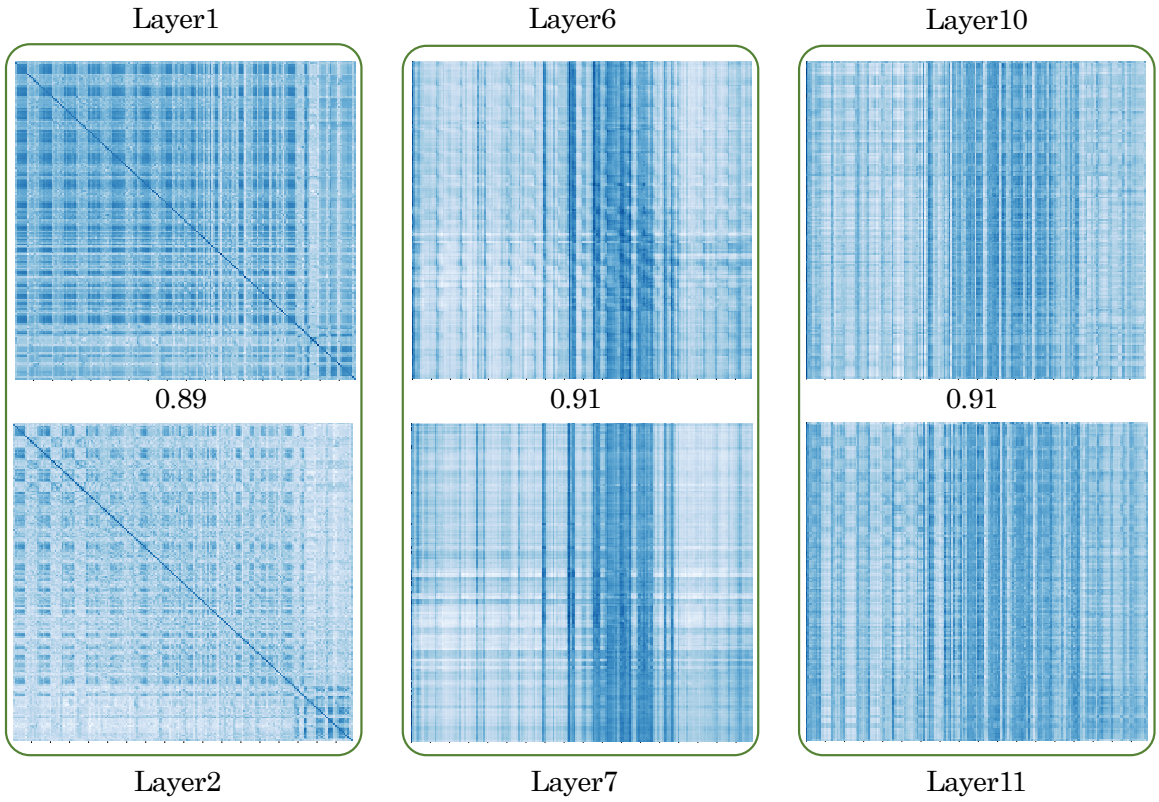


FIGURE 4.2: Attention maps from 6 different transformer layers. The attention maps of layers 1 and 2 have high normalized correlation, $i.e.$ 0.89. Similarly for layers 6, 7 and layers 10, 11 with the normalized correlation being 0.91.

Searching the locations of token pooling is equal to setting the optimal numbers of layers at different stages (we refer to the layers with the same number of tokens as a stage).

The choice of whether a layer shares the attention with its neighbour forms a discrete optimization problem. Our objective is to find the optimal configuration of attention sharing which minimizes the computational cost as well as maximizing the performance. In this work, we apply Neural Architecture Search to solve the configuration.

We treat these key factors in the network design as hyper-parameters and learn these parameters from data with the AutoML method. The two differences between the original ViT model and our proposed PSViT are shown as Fig. 4.1.

The contributions can be summarized as follows:

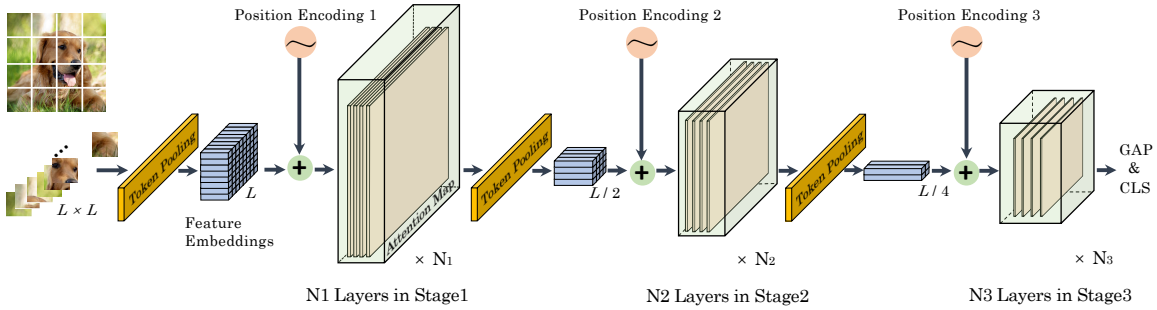


FIGURE 4.3: Pipeline of our proposed PSViT model. There are three stages in the model, which are separated from the proposed token pooling layers.

- We present a PSViT scheme with token pooling and attention sharing to devise a better vision transformer that can effectively enhance the feature representation ability and control the computation allocation flexibly
- We propose a compact set of design choices for token pooling and attention sharing, based on which the optimal design choice is learned by leveraging off-the-shelf AutoML methods.
- Comprehensive experimental results show that the proposed scheme has up to 6.6% accuracy improvement in ImageNet classification compared with the DeiT in (Touvron et al. 2021).

4.2 Related Work

Transformers for Vision. Transformer (Vaswani et al. 2017) was originally proposed to overcome the limited attention range and unsuitability of RNN in NLP. It then quickly gained popularity in the whole field of NLP. In the past few years, a new trend of applying transformer in the field of CV is witnessed. And transformer first found its application for low-level vision problem such as image generation (Parmar et al. 2018) and image super resolution (Yang et al. 2020a). Parmar *et al.* (Parmar et al. 2018) advanced image transformer to generate images through constraining the self-attention mechanism only to local neighborhoods and achieved promising results. Yang *et al.* (Yang et al. 2020a) came up with a new texture transformer network for image super-resolution, where the low resolution and reference

images are formulated as queries and keys in a transformer and obtained new SOTA for image super resolution.

Later, transformer was extended to high-level vision problems such as object detection (Carion et al. 2020; Zhu et al. 2020b), image classification (Dosovitskiy et al. 2021; Chen et al. 2020) and so on. To overcome the problem of some hand-designed components such as non-maximum suppression and anchor generation in the typical object detection, a complete end-to-end scheme (DETR) (Carion et al. 2020) based on transformer was proposed with impressive results. However, DETR still suffers from low convergence. As such, Zhu *et al*. (Zhu et al. 2020b) introduces Deformable DETR, in which attention modules only consider a small set of key sampling points, leading to much faster training speed and better performance especially for small objects. Dosovitskiy *et al*. (Dosovitskiy et al. 2021) explicitly defined the concept of Vision Transformer (ViT), and directly applied transformer to image classification with on-par-with or even better performance. Chen *et al*. (Chen et al. 2020) utilized a sequence transformer to auto-regressively predict pixels with no need of 2D input structure to learn useful representations for images. The notable results reported by them demonstrate the attractive potential of transformers for unsupervised representation learning.

Different from all the previous transformer based works in the field of CV, we concentrate on the token pooling and attention sharing that are not touched before. In addition, we apply the AutoML method to achieve this goal, which makes our work distinct from the manual design of these previous works.

Automated Machine Learning (AutoML). Designing a better and efficient DNN is worthwhile for all the tasks in CV. Previously, most researchers and engineers in the field of DNN try to design better models based on heuristics and experiences. Such a process tends to be very tedious. Fortunately, in the past few years, a novel methodology of designing a better DNN model, that is, AutoML, arises quickly. Currently, the AutoML methods for neural network architecture design can be divided into three categories, reinforcement learning (RL) based methods (Zoph and Le 2016; Cai et al. 2018; Tian et al. 2020), neuro-evolutionary based methods (Elsken et al. 2018; Guo et al. 2020b; Real et al. 2020) and differentiable methods (Liu et al. 2019c; Xie et al. 2019; Wan et al. 2020).

RL based methods (Zoph and Le 2016; Cai et al. 2018) and neuro-evolutionary methods (Liu et al. 2017a; Real et al. 2020) are good at searching discrete architectures, but generally requires huge computational cost in searching. To overcome the great computational burdens of RL and evolutionary methods during the search, differentiable search methods (Liu et al. 2019c; Xie et al. 2019; Wan et al. 2020) relaxed the discrete search space and applied SGD for searching. Then, it received a lot of attention due to its great advantage of searching efficiency (Wan et al. 2020), and it has been widely used in many computer vision problems such as segmentation (Liu et al. 2019b; Chen et al. 2019a) , object detection (Chen et al. 2019c; Ghiasi et al. 2019; Guo et al. 2020a), model compression (Dong and Yang 2019a; Wu et al. 2018; Shaopeng et al. 2020) and so on.

Inspired by the outstanding performance of AutoML for both CV and NLP, we also utilize AutoML for better ViT models with the aim of boosting its performance effectively. Specifically, the evolution based method of SPOS (Guo et al. 2020b) is taken as our hyper-parameters learning method due to its fast efficiency of the single path in the supernet searching and training. Nevertheless, AutoML has not been investigated for Transformer in image recognition and our search space of token pooling and attention sharing has not been studied in CV.

4.3 Method

The whole framework of our proposed PSViT model is shown as 4.3. There are three stages in the model, which are separated from the proposed token pooling layers. The token pooling layers help to reduce the spatial size (from L to $L/4$ in the Figure) and increase the dimension of feature embedding. Each stage has several layers (including the layer with independent or sharing attention). The attention maps from different layers in the same stage have the same size. We apply Global Average Pooling (GAP) to the last features and send the generated vector to a classifier (CLS) for the final classification.

First, we give a brief introduction about transformer in Section 4.3.1. Then, we present two essential mechanisms of our PSViT, including token pooling and attention sharing, in Section 4.3.2. Finally, we introduce enhanced PSViT with AutoML method in Section 4.3.3.

4.3.1 Background about Transformer

Transformer (Vaswani et al. 2017) is built with stacked self-attention layers and feed-forward neural networks.

Self-attention Layer. The self-attention layer utilizes the relevance or interaction of one token to other tokens. The input of self-attention layer is transformed into three different vectors, $i.e.$ the query vector q, the key vector k and the value vector v, and their corresponding packing form from different inputs are Q, K, V respectively, we can calculate their attention among different inputs by the following equation:

$$Att(Q, K, V) = Softmax(\frac{Q \cdot (K^T)}{\sqrt{d_h}}) \cdot V, \tag{4.1}$$

where d_h is the number of features per token and $\cdot$ denotes matrix multiplication. The equation above means that the self-attention can be obtained by first computing the dot product of the query with all the keys, $i.e.$ $Q \cdot (K^T)$. Then, they are normalized with the softmax function to get attention scores. Finally, the attention scores are multiplied by the value vector. As such, the output of each token becomes the weighted sum of all the tokens in a sequence/image. To boost the performance of vanilla self-attention, multi-head attention is further proposed by applying multiple attention functions to the input.

Vision Transformer. Dosovitskiy *et al*. (Dosovitskiy et al. 2021) proposed ViT which directly applied the above transformer from NLP to the image classification task. They divided an image into a sequence of flattening 2D tokens that are treated as the inputs to transformer. A trainable linear projection was adopted for token embeddings with the aim of yielding constant widths for different layers in a transformer. Then, another learnable embedding is further applied to the sequence of embedding tokens. In addition, 1D positional encoding is also attached to the token embeddings to maintain the positional information. When ViT is trained on large datasets, it can achieve a very impressive classification accuracy of 88.36%.

Building upon this work, Touvron *et al*. (Touvron et al. 2021) improved the ViT training speed by introducing a novel teacher-student token distillation strategy specific to transformers

TABLE 4.1: Preliminary experiments on ImageNet about Token Dimension Settings for Each Stage. 'Token Dimensions' denotes the channel number of tokens for the three stages. Similarly for 'Token Numbers' and 'Layer Numbers'.

Model	DeiT-Tiny	Token Dimension1	Token Dimension2
w/o Pooling	no Pooling	2 Poolings	2 Poolings
Token Dimensions	[192, 192, 192]	[192, 192, 192]	[192, 256, 384]
Token Numbers	[197, 197, 197]	[197, 99, 50]	[197, 99, 50]
Layer Numbers	[4, 4, 4]	[4, 8, 20]	[4, 4, 4]
FLOPS(G)	1.3	1.3	1.3
Top1 Acc	72.2%	75.0%	76.3 %

(DeiT), yielding competitive results on ImageNet without external data as in ViT. We choose DeiT as our baseline model in this work for its great training efficiency.

4.3.2 Token Pooling and Sharing for Transformer

Based on the above review and problem analysis for the current ViT, we propose two mechanisms named token pooling and attention sharing to overcome the above three issues for the goal of a better ViT.

4.3.2.1 Token Pooling

The token pooling mechanism adjusts the number of tokens for each stage as Fig. 4.3 shows. When the network depth increases, we decrease the number of tokens for removing spatial redundancy and increase the feature dimension for accommodating more high-level features that should be different from each other. There are two design options for the token pooling. The first one is following the network design in (Dosovitskiy et al. 2021), treating the image patches as 1D tokens and utilizing the additional CLS token for the classification task. The other one is removing the CLS token and keeping the image patches in a 2D array, which is the same as the pooling strategy in most networks for computer vision tasks, such as ResNet (He et al. 2016).

For the first design, similar to (Zhou et al. 2021), we achieve token pooling by convolution and maxpooling for the convenience of implementation. Different from (Zhou et al. 2021) that

only decreases the number of tokens, our goal is to enhance the feature representation ability. Therefore, the feature dimension is not changed in (Zhou et al. 2021) but will be changed in ours. Specifically, we first utilize 1D convolution with a small kernel size to change the feature dimension ($i.e.$, dimension of each token) and then decrease the number of tokens via 1D maxpooling. We name our network with the above pooling strategy as PSViT-1D. In the second strategy, we adopt a 2D convolutional layer with stride 2 for token pooling, which is widely applied in many convolutional networks. Here, we name the network with the second pooling strategy as PSViT-2D.

Simply adding a token pooling layer after each layer will decrease the model's representation ability expeditiously. Motivated by the principle of deep networks in CV, such as VGGNet, ResNet and MobileNet, we add a pooling layer after several layers and name the layers with the same token numbers as a stage.

Investigation on Two Choices of Token dimension. To increase the representation ability of the ViT while maintaining its computational overhead in terms of FLOPS at the same time, we may have two choices to modify the network structure: increasing token dimension while decreasing the token number or keeping the tokens dimension and increasing the transformer layer number. Based on these two choices, we conducted some preliminary experiments with the 1D pooling strategy in Table 4.1, which shows that increasing the feature dimensions while decreasing the number of tokens is a better choice. Specifically, in the experiments, the baseline model fixes the feature size while the 'Dimension1' scheme in Table 4.1 fixes the token dimensions and decreases the token numbers through the stages. To balance the total computation, several encoder layers are added. The 'Dimension2' scheme in Table 4.1 increases the token dimensions while decreasing the token numbers. These two schemes design the architectures following the principle that the computation is uniformly distributed across the stages (Liang et al. 2020). We can find in Table 4.1, these two token dimension designs perform much better than the DeiT-Tiny model which fixes the feature sizes through the whole network. The method decreasing the token number while increasing the feature dimension ('Dimension2') achieves the best performance.

Analysis on the Design Choices. In CV tasks, there are two characteristics. *First*, as discussed in Section 4.1, high-level features have redundancy at spatial level. Therefore, it is reasonable to reduce the redundancy at the spatial level by token pooling. This is supported by the results in Table 4.1, where two token pooling design performs much better than the DeiT-Tiny model without token pooling. *Second*, low-level features could be few but high-level features should be more. Different layers in a deep network encode information of different levels. Low-level features, e.g. edges and textures, at shallow layers can be few and can be shared for representing high-level features. In contrast, high-level features, e.g. attributes or objects of different viewpoints, at deeper layers are more difficult to be shared. Therefore, most of the CNN designs, such as VGGNet (Simonyan and Zisserman 2015), ResNet (He et al. 2016) and MobileNet (Sandler et al. 2018), follow the rule that deeper layers have higher feature dimensions. This is also validated by the results in Table 4.1, where increasing the feature dimension at the deeper layer in 'Dimension2' performs better than fixing the feature dimension in 'Dimension1'.

4.3.2.2 Attention Sharing

As shown in Fig. 4.2, the attention maps in continuous multi-attention layers are similar. As one of the reasons, identity mapping is used in the transformer, making a transformer layer function as the residual. In this case, the main features, though changed by the residual, will not drastically change. Therefore, the features in adjacent layers will be similar to each other, leading to the attention maps among features similar to each other. This similarity leads to redundancy. Hence, it is beneficial to apply some sharing mechanisms so as to reduce redundancy and, at the same time, decrease the computation.

Specifically, we achieve the goal of attention sharing through reusing the attention calculation process between adjacent layers as shown in Fig. 4.1(c). If a layer reuses the attention scores from its preceding layer, then we do not calculate the attention scores as well as the Q and K in Eqn. (4.1). The attention calculated at the preceding layer is used as the input attention of this layer and the whole process of attention calculation in Eqn. (4.1) is simplified to the dot product between V and input attention scores.

Attention sharing can help to remove the redundancy of the attention map among adjacent transformer layers. On the other hand, some adjacent layers might have very different features and sharing their attention maps becomes not so effective. Taking this into consideration, we should provide the flexibility so that the whole ViT still has the choice of using the original multi-head attention module without sharing attention maps. As such, we take the sharing attention module as an optional choice to the original independent multi-head attention module in designing the overall transformer architecture.

4.3.3 AutoML Enhanced Transformer

Recently, AutoML has greatly boosted the SOTA for many models in CV and NLP. In this section, we also turn to AutoML for the goal of a better ViT.

4.3.3.1 Search Space for PSViT

As mentioned before, token pooling refers to the operation that decreases the number of tokens at the spatial level and increases the channel number of features per token, and such an operation may greatly affect the feature representation ability. Hence, we specifically consider its different forms in ViT. The design choices of each stage mainly include three factors: the number of tokens N_t, the token dimension N_f, and the number of layers for each stage N_b. To get an optimal choice for these three factors, we construct a search space that has multiple choices for them. Each element in the search space would lead to a new candidate network architecture for the transformer.

In each layer, there are S_t possible choices for the number of tokens, S_f possible choices for the token dimension, and S_s possible choices for using the attention maps for each map. For L layers, the search space considering only the token pooling would contain $(S_t \cdot S_f \cdot S_s)^L$ candidate designs. For example, when $S_t = 4$, $S_f = 4$, $S_s = 4$, and $L = 36$, then there are about $1.1 * 10^{65}$ choices, about $9.6 * 10^{52}$ times the size of SPOS method (Guo et al. 2020b). In comparison, SPOS has 4 choices in each layer, while this search space has 64 in each layer. This search space is too large and has too many choices in each layer. It costs too much time

when searching algorithms are used for obtaining the best architecture from this search space. Besides, too many choices in each layer may make the existing fast searching algorithm not effective, as found in (Ci et al. 2021). Therefore, we need to reduce the search space.

Our designs of the search space and reasons are as follows:

- According to the analysis in Section 4.3.2.1, we increase the token dimension but decrease the token numbers as the depth increases. This helps to reduce the search space by removing the network architectures that do not fit this rule.
- Following the mature design of CNN, we constrain that multiple layers within a stage have the same number of tokens and the same token dimension. To reduce the search space and computation required by the searching algorithm, we only use three stages of token pooling.
- For the number of layers at each stage, we only consider the limited number of layers to further reduce the search space and computation.
- We provide flexibility to apply attention sharing or not for different layers.

Our supernet design for Representing the Search Space. If we consider the number of layers and the setting for attention map sharing in each stage as hyper-parameters for a network, then a natural choice for learning them is the RL or EA based approaches, which are very slow in searching. Therefore, we employ weight sharing based methods (Cai et al. 2019; Bender et al. 2018) as our searching algorithm, which requires defining a supernet to subsume all candidate architectures in the search space. A possible design of supernet is shown in Fig. 4.4. There are three stages in the supernet. A token pooling layer is placed between stages to change the token number and features dimension. Each stage has six cells, where a cell has three choices of paths, a basic transformer layer, two sharing layers with the attention map of the first transformer layer being copied to the second layer, and identity mapping. A candidate architecture could choose one path out of three choices for each cell independently. With the inclusion of identity mapping, the candidate network can have 0 (all cells choosing the identity layer) to 36 (all cells choosing the sharing layers) transformer layers. Such supernet design provides more feasibility to the transformer architecture. For example, a candidate

could only select basic layers for all layers, which is equivalent to the transformer enhanced by two token pooling layers. As another example, a candidate architecture could select the sharing layers path for all the first 6 cells at the first stage and select identity for the remaining cells, which has 12 transformer layers with every two layers sharing attention map but without token pooling.

4.3.3.2 AutoML for the Searching

With the search space and supernet defined in section 4.3.3.1, we find the optimal network architecture for the transformer using the SPOS (Guo et al. 2020b), which is briefly introduced below.

Training the supernet. The built supernet in section 4.3.3.1 includes all the candidate networks. For each cell of the supernet, there are multiple choices. By activating only one choice in each cell, a candidate network can be constructed. During training, SPOS only selects one candidate network by uniform sampling for each training iteration and updates the parameters of the selected candidate network in the supernet. The candidate network will inherit the trained parameters from the supernet and keep updating these parameters. Once the supernet is trained, all the candidate networks can inherit the weights from it without being trained from scratch for searching.

Searching from the trained supernet. After the supernet is trained, the best candidate network architecture can be obtained by further applying the evolutionary method to all the candidate networks in the supernet. More details could be found in (Guo et al. 2020b).

4.4 Experiments

In this section, we elaborate on the dataset, implementation details, and experimental results to evaluate the performance of the proposed PSViT.

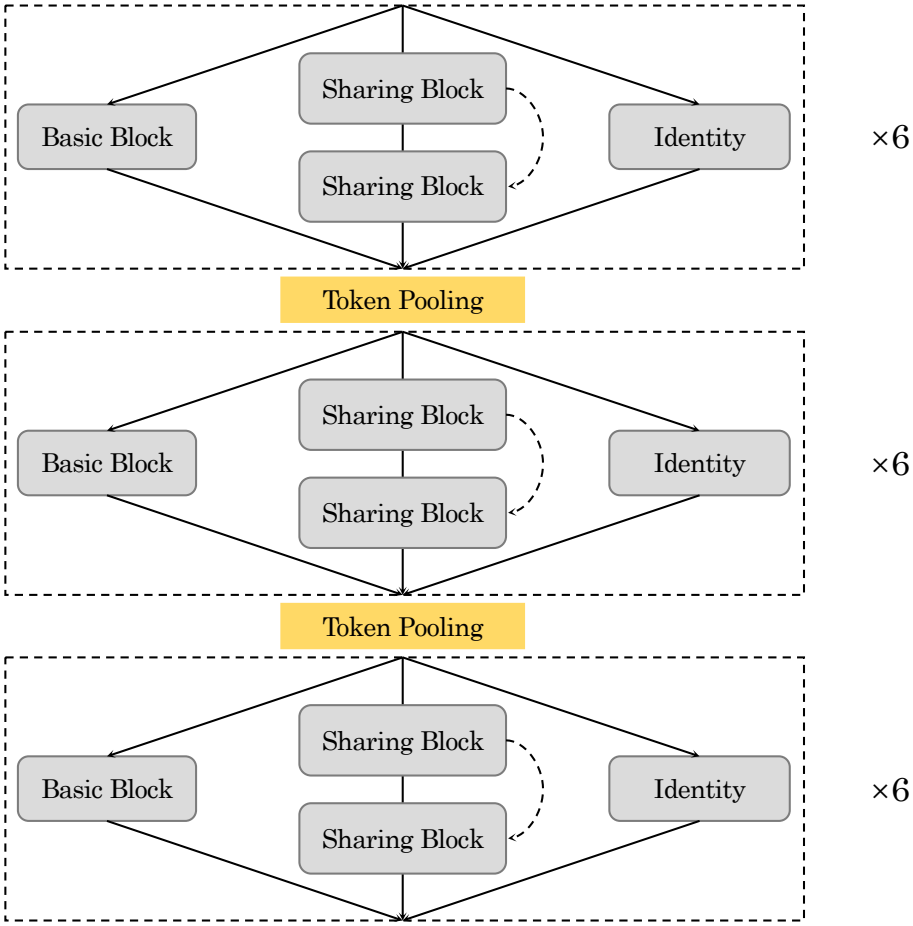


FIGURE 4.4: The architecture of our supernet. There are three stages, each stage containing 6 cells. A cell has three choices of paths, a basic transformer layer, sharing layers, and an identity layer. A candidate network has a single path for each cell.

4.4.1 Datasets and Implementation Details

Experimental Dataset. To validate the effectiveness of the proposed PSViT, we conduct experiments on image classification. We conduct all classification experiments on the ImageNet 1000-class image classification dataset, which is a widely used benchmark for validating network architecture design. This dataset has about 1 million training data. Part of the training data is used as the validation data. The training data without the validation data is used for training the supernet. The validation data is used by the AutoML for searching from the trained supernet.

Implementation Details. We choose the DeiT (Touvron et al. 2021) mentioned before as our baseline since it is the current SOTA for vision transformer in image classification. DeiT has two major different versions, that is, the Small one and the Tiny one, for efficient training, which are derived from the basic version of DeiT-B that has 12 transformer layers, a fixed token dimension of 768, 12 heads, and feature dimension of each head d_h being 64. The differences between the Small one and the Tiny one are feature dimension and head number. More details about the other related details for DeiT can be found in (Touvron et al. 2021). We follow the framework of DeiT (Touvron et al. 2021) to obtain the feature embeddings, which are sent to transformer encoders for further processing. We implemented our model based on PyTorch.

Training settings. For the supernet training, mini-batch Nesterov SGD optimizer with a momentum of 0.9 is adopted. The initial learning rate is set to 0.2 and adjusted to 0 gradually by the cosine annealing learning rate decay. We train the network with a batch size of 1024 and L2 regularization of 1e-4 for 100 epochs. Besides, the label smoothing is applied with a 0.1 smooth ratio. For subnet searching, we use the same setting for the evolutionary algorithm as that in (Guo et al. 2020b) under the FLOPS constraint. Specifically, the evolutionary algorithm samples $N_s = 1000$ subnets with population size 50, max iterations 20, and retaining top-10 models during the evolutionary search. After the optimal subnet is found, we retrain the subnet. For retraining the subnet, we utilize the same training setting as that in (Touvron et al. 2021) due to its efficiency.

4.4.2 Classification Results

For the classification task, we compare our two type searched models, PSViT-1D and PSViT-2D with two widely applied models, ResNet (He et al. 2016) and ResNext (Xie et al. 2017), and the SOTA pure transformer vision model DeiT (Touvron et al. 2021). More specifically, ResNet18 and ResNet 50 are shown in Table 4.2 since they are provided in (Touvron et al. 2021) and they have similar FLOPs as DeiT-Tiny and DeiT-Small respectively, as pointed out in (Touvron et al. 2021). Moreover, ResNetXt50-32x4d and ResNetXt101-64x4d that improved ResNet by aggregating the residual module are also included for comparison due

TABLE 4.2: Comparison of ImageNet classification performance between different models. 'Acc' denotes the Top-1 accuracy and '*' denotes the performance under our training setting, which is the same as that in DeiT (Touvron et al. 2021).

Model	FLOPS(G)	Acc(%)
ResNet18	1.8	69.8
ResNet18*	1.8	68.5
DeiT-Tiny	1.3	72.2
PSViT-1D-Tiny	1.4	**77.4**
PSViT-2D-Tiny	1.3	**78.8**
ResNet50	4.1	76.1
ResNet50*	4.1	78.5
X50-32x4d	4.3	77.6
X50-32x4d*	4.3	79.1
DeiT-Small	4.6	79.6
PSViT-1D-Small	4.9	**80.7**
PSViT-2D-Small	4.4	**81.6**
X101-64x4d	15.6	79.6
X101-64x4d*	15.6	81.5
ViT-Base	17.6	77.9
DeiT-Base	17.6	81.8
PSViT-1D-Base	18.9	**82.6**
PSViT-2D-Base	15.5	**82.9**

to their outstanding performance in image classifications. For a fair comparison, the above corresponding models trained on our machine using our training settings (the same as DeiT) are further represented with a symbol of * in the table. The results for ViT-Base in Table 4.2 is copied from (Dosovitskiy et al. 2021). Note that using the training setting of DeiT for both ViT and DeiT will make ViT and DeiT having the same accuracy because the only difference between DeiT and ViT is the training setting.

Table 4.2 shows the experimental results comparing our searched models with other methods under different computation budgets. We can find that under similar FLOPS, our models achieve better classification accuracy. For example, the accuracy of our searched 1D/2D model under 1.3 GFLOPS is more than 5% / 6% higher (we use absolute accuracy improvement instead of relative improvement in this chapter) than DeiT-Tiny with similar FLOPS and more than 7% / 8% higher than the ResNet18, whose FLOPS is 1.4 times of ours. Such huge

improvements in terms of better classification accuracy and fewer FLOPS make the proposed PSViT a more attractive and powerful competitor to the recent ViT and DeiT.

For ResNet50, ResNetXt, and DeiT-Small, our corresponding improved model obtained by the PSViT-1D also outperforms them respectively by a significant margin of 2.2%, 1.6%, and 1.1%. The PSViT-2D is 0.9% better than the PSViT-1D with fewer FLOPS. While for the large model comparisons among ResNetXt-101, DeiT-Base, and our base one, our respective classification gain is more than 1% for the PSViT-2D-Base model.

All the above three results for image classification show the effectiveness of the proposed PSViT scheme. We attribute the gain of our searched models to the two key parts advanced in this work: the proposed token pooling module that can more effectively capture different levels of image features, and the attention sharing mechanism that flexibly adapts the computational cost for different transformer layers, achieving a better trade-off between accuracy and model redundancy.

TABLE 4.3: ImageNet top-1 classification accuracy for models with different mechanisms, $i.e.$ attention sharing, token pooling, and using AutoML or manual design for these two factors.

Attention Sharing	Token Pooling	AutoML	FLOPS (G)	acc-1D %	acc-2D %
			1.3	72.2	72.2
✓			1.3	73.8	-
	✓		1.3	76.3	76.7
✓	✓	✓	1.3	77.4	78.8

4.4.3 Ablation Study

To have a better understanding of the proposed PSViT, we carry out some ablation studies in this section, which are detailed as follows. All experimental results are conducted on ImageNet using the same training setting mentioned in Section 4.4.1.

Effectiveness of Token Pooling, Attention Sharing, and AutoML. First of all, we evaluate our proposed mechanism through simple manually designed models as in Table 4.3. We first devise the ViT model with attention sharing between two adjacent layers. To balance

TABLE 4.4: Token Number Setting based on PSViT-1D. Tiny/8 and Tiny/16 have differences in the input token numbers. To ensure Tiny/8 and Tiny/16 have similar FLOPs, token dimension and number of heads in the multi-head attention are adjusted correspondingly. Similarly for Small/8 and Small/16. 'Token Dimensions' denotes numbers of features per token for the three stages, similarly for 'Token numbers'. 'Num. heads' denotes the number of heads in the multi-head attention for the three stages.

Model	Tiny/8	Tiny/16	Small/8	Small/16
Token Dimensions	[64, 144, 192]	[192, 288, 384]	[144, 256, 384]	[288, 512, 768]
Num. heads	[1, 3, 3]	[3, 6, 6]	[3, 4, 6]	[6, 8, 12]
Token Numbers	[785, 393, 197]	[197, 99, 50]	[785, 393, 197]	[197, 99, 50]
Params(M)	3.8	15.6	13.9	53.8
FLOPS(G)	1.5	1.3	4.9	4.0
Top1 Acc	72.71%	77.40%	80.70 %	78.32 %

the computation, we add additional layers at the end of the original model to reach similar FLOPS. The attention sharing improves the accuracy by 1.6%. Then, we evaluate the proposed token pooling with the setting of 'Token Dimension2' described in Table 4.1. Token pooling improves the performance of the model significantly, that is, from 72.2% to 76.3% for PSViT-1D and to 76.7% for PSViT-2D.

Moreover, we study the effectiveness of AutoML. We utilize SPOS method to search for an optimal architecture under the FLOPS constraint, considering both token pooling and attention sharing. The supernet for the SPOS is shown in Fig. 4.4. As shown in Table 4.3, there is obvious performance improvement of 1.1%/2.1% for PSViT-1D/PSViT-2D, which demonstrates the efficacy of the utilized AutoML method of SPOS.

Token Number Settings. As we mentioned before, the hyper-parameters about attention pooling have multiple choices. We evaluate the different settings as in Table 4.4. Since we follow the principle of increasing token dimensions while decreasing token numbers, the only hyper-parameters should be the initial/final token numbers. We evaluate two types of token numbers based on PSViT-1D, corresponding to the way that input images are split. For a model with input token having size 16×16 (models with '/16'), the initial/final tokens number is 197/50 while the model with input token size being 8×8 (models with '/8') has 785/197 initial/final tokens. The model Tiny/16 is our searched model as mentioned and other

models are obtained through scaling up (Small/16) or tuning the initial token numbers (Tiny/8, Small/16). For the Tiny models, the model with fewer token numbers performs much better. The Tiny model with more tokens has too many tokens with lower feature dimension in the final stage which leads to poor representation ability. Different from Tiny model, the Small model with more tokens achieves better performance. The reason is that the token dimensions are enough for the small model and the performance of the small model with few tokens drops due to the possible overfitting caused by the superfluous parameters.

Attention Sharing Settings. We further study different attention sharing paradigms. And the experimental results based on 1D pooling are shown in Table 4.5. In the table, 'sharing 2' means every 2 adjacent layers share the same attention. 'sharing 3' denotes every 3 adjacent layers share the same attention, $i.e.$ more layers sharing the same attention. We can clearly see that applying sharing can yield obvious improvements in classification accuracy compared to the vanilla ViT that does not share attention ('no sharing' in Table 4.5), with a promising margin of 1.6%, and 0.9% respectively for sharing in 2 layers and 3 layers. Sharing between 2 layers achieves better performance than sharing among 3 because features will change more when going through 3 layers and the attention may be different in 3 layers. For a better performance, we select sharing within 2 layers as our network design element in the supernet.

TABLE 4.5: ImageNet top-1 classification accuracy for different sharing settings. 'no sharing' denotes the network without sharing attention. 'sharing 2' and 'sharing 3' respectively denote every 2 and 3 adjacent transformer layers share the same attention map.

model	FLOPS (G)	acc-1D (%)
no sharing	1.3	72.2
sharing 2	1.3	73.8
sharing 3	1.3	73.1

4.4.4 Visualization

Visualization. In Fig. 4.5, we show the visualization of learned features from DeiT (the 2nd and 5th columns) and our PSViT (the 3rd and 6th columns). We utilize the Grad-CAM in (Selvaraju et al. 2017) to show where is 'important' for predictions. We can find that our

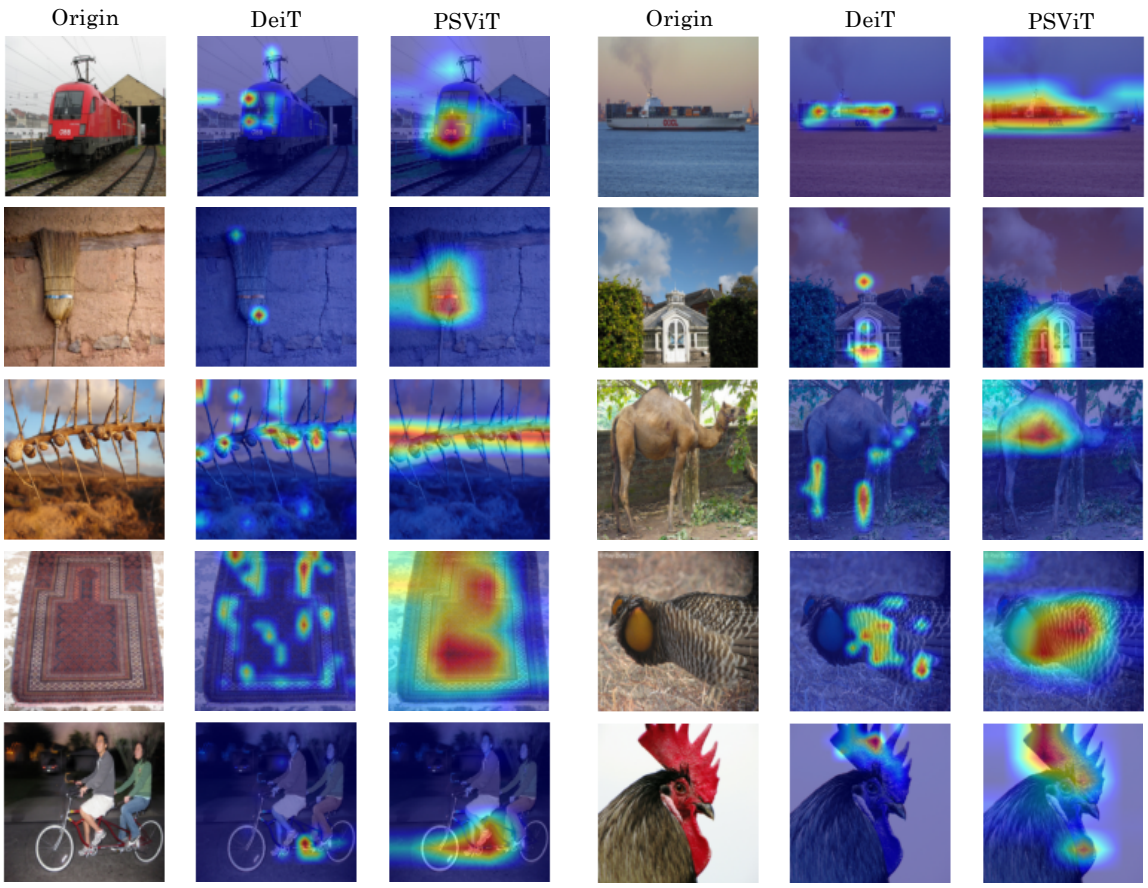


FIGURE 4.5: Visualization of features for DeiT and our PSViT. Images in the 1st and 4th columns are from ImageNet.

PSViT can focus on a more complete region-of-interests than DeiT. In addition, our model has larger feature dimensions which can capture more information than the baseline model, leading to possible better classification or recognition results.

4.4.5 Searched Architectures

Fig. 4.6 shows the searched architectures of PSViT-1D and PSViT-2D. There are about 50% layers sharing the attention, which shows that the proposed attention sharing mechanism can achieve a great trade-off between accuracy and computation costs. For the computation allocation among the three stages, both models allocate less computation in the first stage and more computation in the last stage. The features in the first stage have a larger spatial size. Applying self-attention to these features leads to a heavy computation burden. We think that

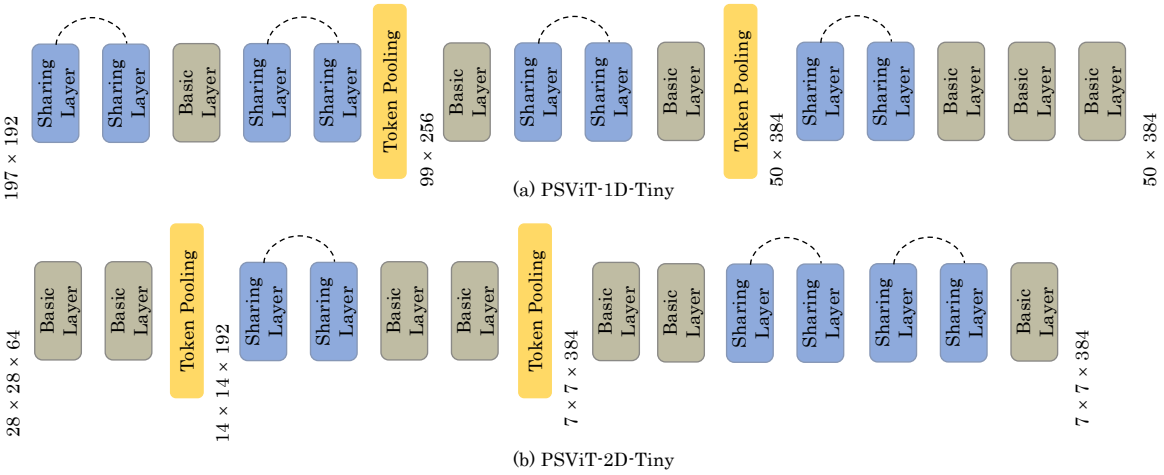


FIGURE 4.6: Searched architectures of PSViT-1D-Tiny and PSViT-2D-Tiny. The feature size does not change in the same stage.

is why the first stage tends to adopt attention sharing more times or has fewer total layers. It is a good strategy to reduce unnecessary computation. Differently, applying self-attention on deeper layers which have less tokens is better. The saved computation allows us to increase the feature dimension or layer number properly for a stronger feature representation. For the last layer of the two models, there is no attention sharing. The last layer deserves an independent self-attention operation because its output is directly used for the final classification. More computation on the last layer will produce more accurate classification results.

TABLE 4.6: Performance comparison between different backbones on the object detection and instance segmentation tasks.

Backbone	Object Detection					
	AP	AP50	AP75	APs	APm	All
ResNet18	34.8	56.3	37.5	20.1	37.0	45.2
PSViT-2D-Tiny	40.8	64.7	44.0	25.3	43.8	53.9
ResNet50	38.3	60.4	41.4	23.3	41.6	48.9
ResNet101	39.5	61.3	43.0	23.4	42.7	51.1
Backbone	Instance Segmentation					
	AP	AP50	AP75	APs	APm	All
ResNet18	32.8	53.4	34.8	17.4	25.1	44.7
PSViT-2D-Tiny	37.7	60.1	39.9	21.2	40.6	52.8
ResNet50	35.5	57.2	37.8	19.6	38.6	47.7
ResNet101	36.5	58.2	39.1	19.6	39.7	49.4

4.4.6 Comparison between PSViT-1D and PSViT-2D

Our PSViT-1D and PSViT-2D have different searched architectures as shown in Fig. 4.6. For PSViT-1D, we set the output token number as 50 (with the CLS token). Similarly, the spatial size of the output feature in PSViT-2D is 7×7. Both models have token pooling layers with a stride of 2, which means token number will reduce 50% in PSViT-1D and 75% for the PSViT-2D. In PSViT-1D, the token pooling is applied to 1D features. The token number is 197 for the first stage and 99 for the second stage. For PSViT-2D, the token number of the first stage and the second stage are 28×28 and 14×14, respectively. As the Table 4.2 shows, our PSViT-2D models achieve better performance than PSViT-1D models. We think the main reason is that 2D token pooling can keep the structure information of 2D images while the 1D token pooling only considers the neighbouring tokens on the x-axis but ignores those on the y-axis. Besides, 2D features from PSViT-2D is more suitable for downstream tasks.

4.4.7 Object detection and instance segmentation results

To evaluate the transferability of our models, we contrast our PSViT-2D model to ResNet on the downstream tasks, $i.e.$ object detection and instance segmentation. We apply the Mask-RCNN-FPN (He et al. 2017) as our framework. We conduct experiments on MSCOCO 2017 dataset, which consists of 118K training images and 5K validation images. We report the performance on the validation set.

For the network training, we first initialize the backbone parameters with our pre-trained weights on ImageNet and adopt Xavier initialization on other layers in the neck and heads. Our models are trained with the batch size of 16 on 8 V100 GPUs and optimized by AdamW with the initial learning rate of 0.0001, 0.05 weight decay. We follow the standard common 1x training setting (12 epochs) to train the whole Mask R-CNN models with our searched models and ResNet as the backbone. During the training stage, we follow the multi-scale training setting in (Sun et al. 2021): resizing the shorter side of input between 480 and 800 while the longer side is at most 1333. During testing, the shorter side of the input is fixed as 800 pixels.

The results are shown in Table 4.6. Our PSViT-2D-Tiny achieves 40.8% mAP on the object detection task and 37.7% mAP on the instance segmentation task, which is 6 and 4.9 points higher than that of ResNet18, respectively. Our PSViT-2D-Tiny even achieves better performance than ResNet101. The much better performance than ResNets on both object detection and instance segmentation shows the great transferability of our models.

4.5 Summary

In this chapter, we have introduced a new PSViT by making use of token pooling and attention sharing. To effectively enhance the performance of current ViT, token pooling and attention sharing are first designed. Then, the search space for a PSViT stage is designed. Based on the search space, we further apply an efficient AutoML method, SPOS, to obtain the optimal module design. Extensive comparison experiments on the ImageNet image classification dataset show the effectiveness of the proposed PSViT since it can increase the classification accuracy a lot. The great performance on object detection and instance segmentation also validates the transferability of our models.

CHAPTER 5

Conclusion

In this thesis, we propose three new methods for convolutional network and Vision Transformer network architecture search. The proposed BNNAS aims to improve the searching efficiency and the proposed GLiT and PSViT aim to search new models for Vision Transformers with better performance.

Computation cost in the process of neural architecture search is always a burden for its wide application. In Chapter 2, we propose a novel BN-based indicator and apply this indicator to weight-sharing based neural architecture search methods, including Evolutionary Algorithms and Gradient-based methods. With training only BN parameters, the searching cost is further reduced. We greatly reduce these methods' searching costs without any performance drop. Although recent models all apply the BN layers for stable and fast training, it cannot be ignored that our method highly relies on the BN layers in the convolutional networks. There exists the possibility for a more generalized indicator that can be applied to the networks without BN layers.

Recently, the new transformer model has been introduced into the field of computer vision. The Vision Transformer model follows the network design in the Natural Language Processing (NLP) tasks, which may not be the optimal choice. In Neural Architecture Search for Global and Local Image Transformer (GLiT), we introduce the convolutional layer into the pure Vision Transformer model and treat the convolutional layer as the local information extractor and search a new model architecture considering the Global and Local information. Although we introduce the convolutional layer into Vision Transformer, there are still many techniques applied in traditional convolutional neural networks, such as downsampling, should be

applied in the Vision Transformer models. Combining with neural architecture search, these techniques will further improve the performance of Vision Transformer models.

In Chapter 4, we have a further investigation on the key part of the Vision Transformer model, self-attention. We find two different redundancy on the spatial level and attention level. According to the observation, we propose token pooling and attention-sharing techniques to improve the capability of the Vision Transformer model. With the help of neural architecture search, we achieve better performance than the baseline model with similar model sizes. Despite achieving competitive results on different computer vision tasks, the searching cost cannot be ignored. The searching method designed for convolutional neural networks is applied in this new model. A transformer-specific searching method can be proposed to further improve the effectiveness and efficiency of neural architecture search on Vision Transformer models.

Bibliography

Ba, Jimmy Lei, Jamie Ryan Kiros and Geoffrey E Hinton (2016). 'Layer normalization'. In: *arXiv preprint arXiv:1607.06450*.

Bello, Irwan et al. (2017). 'Neural Optimizer Search with Reinforcement Learning'. In: *ICML*.

Beltagy, Iz, Matthew E Peters and Arman Cohan (2020). 'Longformer: The long-document transformer'. In: *arXiv preprint arXiv:2004.05150*.

Bender, Gabriel et al. (2018). 'Understanding and simplifying one-shot architecture search'. In: *ICML*.

Cai, Han, Ligeng Zhu and Song Han (2019). 'ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware'. In: *ICLR*.

Cai, Han et al. (2018). 'Efficient architecture search by network transformation'. In: *AAAI*.

Carion, Nicolas et al. (2020). 'End-to-End Object Detection with Transformers'. In: *ECCV*.

Chen, Boyu et al. (2018a). 'Real-time'Actor-Critic'Tracking'. In: *ECCV*.

Chen, Boyu et al. (2021a). 'BN-NAS: Neural Architecture Search with Batch Normalization'. In: *ICCV*.

Chen, Boyu et al. (2021b). 'Glit: Neural architecture search for global and local image transformer'. In: *ICCV*.

Chen, Hanting et al. (2021c). 'Pre-trained image processing transformer'. In: *CVPR*.

Chen, Liang-Chieh et al. (2018b). 'Searching for Efficient Multi-Scale Architectures for Dense Image Prediction'. In: *NeurIPS*.

Chen, Mark et al. (2020). 'Generative pretraining from pixels'. In: *ICML*.

Chen, Minghao et al. (2021d). 'AutoFormer: Searching Transformers for Visual Recognition'. In: *ICCV*.

Chen, Wuyang et al. (2019a). 'Fasterseg: Searching for faster real-time semantic segmentation'. In: *arXiv preprint arXiv:1912.10917*.

Chen, Xin et al. (2019b). 'Progressive Differentiable Architecture Search: Bridging the Depth Gap Between Search and Evaluation'. In: *ICCV*.

Chen, Yukang et al. (2019c). 'DetNAS: Backbone Search for Object Detection'. In: *NeurIPS*.

Chu, Xiangxiang, Bo Zhang and Ruijun Xu (2021). 'Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search'. In: *ICCV*.

Ci, Yuanzheng et al. (2021). 'Evolving Search Space for Neural Architecture Search'. In: *ICCV*.

Contributors, MMSegmentation (2020). *MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark*. https://github.com/open-mmlab/mmsegmentation.

Cordts, Marius et al. (2016). 'The cityscapes dataset for semantic urban scene understanding'. In: *CVPR*.

Dauphin, Yann N. et al. (2017). 'Language Modeling with Gated Convolutional Networks'. In: *ICML*.

Deng, Jia et al. (2009). 'ImageNet: A large-scale hierarchical image database'. In: *CVPR*.

Devlin, Jacob et al. (2018). 'Bert: Pre-training of deep bidirectional transformers for language understanding'. In: *arXiv preprint arXiv:1810.04805*.

Dong, Xuanyi and Yi Yang (2019a). 'Network Pruning via Transformable Architecture Search'. In: *NeurIPS*.

— (2019b). 'One-shot neural architecture search via self-evaluated template network'. In: *CVPR*.

— (2019c). 'Searching for a robust neural architecture in four gpu hours'. In: *CVPR*.

— (2020). 'Nas-bench-201: Extending the scope of reproducible neural architecture search'. In: *arXiv preprint arXiv:2001.00326*.

Dosovitskiy, Alexey et al. (2021). 'An image is worth 16x16 words: Transformers for image recognition at scale'. In: *ICLR*.

Elsken, Thomas, Jan Hendrik Metzen and Frank Hutter (2018). 'Efficient multi-objective neural architecture search via lamarckian evolution'. In: *arXiv preprint arXiv:1804.09081*.

Frankle, Jonathan, David J. Schwab and Ari S. Morcos (2020). 'Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs'. In: *CoRR* abs/2003.00152.

Ghiasi, Golnaz, Tsung-Yi Lin and Quoc V Le (2019). 'Nas-fpn: Learning scalable feature pyramid architecture for object detection'. In: *CVPR*.

Gulati, Anmol et al. (2020). 'Conformer: Convolution-augmented Transformer for Speech Recognition'. In: *INTERSPEECH*.

Guo, Jianyuan et al. (2020a). 'Hit-detector: Hierarchical trinity architecture search for object detection'. In: *CVPR*.

Guo, Zichao et al. (2020b). 'Single Path One-Shot Neural Architecture Search with Uniform Sampling'. In: *ECCV*.

He, Kaiming et al. (2016). 'Deep Residual Learning for Image Recognition'. In: *CVPR*.

He, Kaiming et al. (2017). 'Mask r-cnn'. In: *ICCV*.

He, Xin, Kaiyong Zhao and Xiaowen Chu (2021). 'AutoML: A Survey of the State-of-the-Art'. In: *Knowledge-Based Systems*.

Howard, Andrew et al. (2019). 'Searching for mobilenetv3'. In: *ICCV*.

Hu, Jie, Li Shen and Gang Sun (2018a). 'Squeeze-and-Excitation Networks'. In: *CVPR*.

— (2018b). 'Squeeze-and-excitation networks'. In: *CVPR*.

Hu, Yiming et al. (2020). 'Angle-based search space shrinking for neural architecture search'. In: *ECCV*.

Huang, Gao et al. (2017). 'Densely connected convolutional networks'. In: *CVPR*.

Ioffe, Sergey and Christian Szegedy (2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *ICML*.

Kang, Minsoo and Bohyung Han (2020). 'Operation-Aware Soft Channel Pruning using Differentiable Masks'. In: *CoRR* abs/2007.03938.

Kietzmann, Tim C et al. (2019). 'Recurrence is required to capture the representational dynamics of the human visual system'. In: *Proceedings of the National Academy of Sciences*.

Krizhevsky, Alex, Geoffrey Hinton et al. (2009). 'Learning multiple layers of features from tiny images'. In.

Krizhevsky, Alex, Ilya Sutskever and Geoffrey E Hinton (2012). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *NeurIPS*.

Larsson, Jonas and David J Heeger (2006). 'Two retinotopic visual areas in human lateral occipital cortex'. In: *Journal of neuroscience*.

LeCun, Yann, Yoshua Bengio et al. (1995). 'Convolutional networks for images, speech, and time series'. In: *The handbook of brain theory and neural networks*.

Li, Liam and Ameet Talwalkar (2020). 'Random search and reproducibility for neural architecture search'. In: *Uncertainty in artificial intelligence*.

Li, Peixia et al. (2019). 'GradNet: Gradient-guided network for visual object tracking'. In: *ICCV*.

Li, Xian et al. (2020a). 'Deep Transformers with Latent Depth'. In: *Advances in Neural Information Processing Systems*.

Li, Xiang et al. (2020b). 'Improving one-shot nas by suppressing the posterior fading'. In: *CVPR*.

Li, Yuhong et al. (2023). 'Extensible and Efficient Proxy for Neural Architecture Search'. In: *ICCV*.

Liang, Feng et al. (2020). 'Computation Reallocation for Object Detection'. In: *ICLR*.

Lin, Ming et al. (2021). 'Zen-nas: A zero-shot nas for high-performance image recognition'. In: *ICCV*.

Lin, Tsung-Yi et al. (2014). 'Microsoft coco: Common objects in context'. In: *ECCV*.

Liu, Chenxi et al. (2019a). 'Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation'. In: *CVPR*.

Liu, Chenxi et al. (2019b). 'Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation'. In: *CVPR*.

Liu, Hanxiao, Karen Simonyan and Yiming Yang (2019c). 'DARTS: Differentiable Architecture Search'. In: *ICLR*.

Liu, Hanxiao et al. (2017a). 'Hierarchical representations for efficient architecture search'. In: *arXiv preprint arXiv:1711.00436*.

Liu, Jie et al. (2021). 'Inception convolution with efficient dilation search'. In: *CVPR*.

Liu, Zhuang et al. (2017b). ‘Learning Efficient Convolutional Networks through Network Slimming’. In: *ICCV*.

Ma, Ningning et al. (2018). ‘ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design’. In: *ECCV*.

Newell, Alejandro, Kaiyu Yang and Jia Deng (2016). ‘Stacked hourglass networks for human pose estimation’. In: *ECCV*.

Parmar, Niki et al. (2018). ‘Image transformer’. In: *ICML*.

Pham, Hieu et al. (2018). ‘Efficient Neural Architecture Search via Parameter Sharing’. In: *ICML*.

Real, Esteban et al. (2019). ‘Regularized Evolution for Image Classifier Architecture Search’. In: *AAAI*.

Real, Esteban et al. (2020). ‘Automl-zero: Evolving machine learning algorithms from scratch’. In: *ICML*.

Ren, Shaoqing et al. (2015). ‘Faster r-cnn: Towards real-time object detection with region proposal networks’. In: *Advances in neural information processing systems* 28, pp. 91–99.

Sandler, Mark et al. (2018). ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’. In: *CVPR*.

Selvaraju, Ramprasaath R. et al. (2017). ‘Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization’. In: *ICCV*.

Shaopeng, Guo et al. (2020). ‘DMCP: Differentiable Markov Channel Pruning for Neural Networks’. In: *CVPR*.

Shi, Han et al. (2021). ‘SparseBERT: Rethinking the Importance Analysis in Self-attention’. In: *arXiv preprint arXiv:2102.12871*.

Simonyan, Karen and Andrew Zisserman (2015). ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’. In: *ICLR*.

Srinivas, Aravind et al. (2021). ‘Bottleneck transformers for visual recognition’. In: *CVPR*.

Sun, Peize et al. (2021). ‘Sparse r-cnn: End-to-end object detection with learnable proposals’. In: *CVPR*.

Szegedy, Christian et al. (2015). ‘Going deeper with convolutions’. In: *CVPR*.

Tan, Mingxing and Quoc V. Le (2019). 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks'. In: *ICML*.

Tan, Mingxing, Ruoming Pang and Quoc V. Le (2020). 'EfficientDet: Scalable and Efficient Object Detection'. In: *CVPR*.

Tian, Yuan et al. (2020). 'Off-policy reinforcement learning for efficient and effective GAN architecture search'. In: *ECCV*.

Touvron, Hugo et al. (2021). 'Training data-efficient image transformers & distillation through attention'. In: *ICML*.

Tsai, Henry et al. (2020). 'Finding Fast Transformers: One-Shot Neural Architecture Search by Component Composition'. In: *arXiv preprint arXiv:2008.06808*.

Vaswani, Ashish et al. (2017). 'Attention is all you need'. In: *Advances in neural information processing systems*, pp. 5998–6008.

Wan, Alvin et al. (2020). 'Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions'. In: *CVPR*.

Wang, Huiyu et al. (2021a). 'Max-deeplab: End-to-end panoptic segmentation with mask transformers'. In: *CVPR*.

Wang, Xiaolong et al. (2018). 'Non-Local Neural Networks'. In: *CVPR*.

Wang, Yuqing et al. (2021b). 'End-to-end video instance segmentation with transformers'. In: *CVPR*.

Wu, Bichen et al. (2018). 'Mixed precision quantization of convnets via differentiable neural architecture search'. In: *arXiv preprint arXiv:1812.00090*.

Wu, Bichen et al. (2019). 'FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search'. In: *CVPR*.

Xiang, Lichuan et al. (2021). 'Zero-cost proxies meet differentiable architecture search'. In: *arXiv preprint arXiv:2106.06799*.

Xie, Saining et al. (2017). 'Aggregated residual transformations for deep neural networks'. In: *CVPR*.

Xie, Sirui et al. (2019). 'SNAS: stochastic neural architecture search'. In: *ICLR*.

Xu, Jin et al. (2020). 'Task-Agnostic and Adaptive-Size BERT Compression'. In.

Yang, Fuzhi et al. (2020a). ‘Learning texture transformer network for image super-resolution’. In: *CVPR*.

Yang, Zhaohui et al. (2020b). ‘CARS: Continuous Evolution for Efficient Neural Architecture Search’. In: *CVPR*.

You, Haoran et al. (2019). ‘Drawing early-bird tickets: Towards more efficient training of deep networks’. In: *CoRR* abs/1909.11957.

You, Shan et al. (2020). ‘GreedyNAS: Towards Fast One-Shot NAS With Greedy Supernet’. In: *CVPR*.

Zeiler, Matthew D and Rob Fergus (2014). ‘Visualizing and understanding convolutional networks’. In: *ECCV*.

Zhang, Mingyang et al. (2023). ‘ShiftNAS: Improving One-shot NAS via Probability Shift’. In: *ICCV*.

Zhang, Qian et al. (2020). ‘Transformer Transducer: A Streamable Speech Recognition Model with Transformer Encoders and RNN-T Loss’. In: *ICASSP*.

Zhao, Hengshuang, Jiaya Jia and Vladlen Koltun (2020). ‘Exploring Self-Attention for Image Recognition’. In: *CVPR*.

Zhou, Bolei et al. (2019). ‘Semantic understanding of scenes through the ade20k dataset’. In: *IJCV*.

Zhou, Dongzhan et al. (2020). ‘EcoNAS: Finding Proxies for Economical Neural Architecture Search’. In: *CVPR*.

Zhou, Haoyi et al. (2021). ‘Informer: Beyond efficient transformer for long sequence time-series forecasting’. In: *AAAI*.

Zhu, Wei et al. (2020a). ‘AutoTrans: Automating Transformer Design via Reinforced Architecture Search’. In: *arXiv preprint arXiv:2009.02070*.

Zhu, Xizhou et al. (2020b). ‘Deformable detr: Deformable transformers for end-to-end object detection’. In: *arXiv preprint arXiv:2010.04159*.

Zoph, Barret and Quoc V Le (2016). ‘Neural architecture search with reinforcement learning’. In: *arXiv preprint arXiv:1611.01578*.

Zoph, Barret et al. (2018). ‘Learning Transferable Architectures for Scalable Image Recognition’. In: *CVPR*.