공학박사 학위논문

Neural Network Design and Acceleration for Resource-constrained Environments

한정된 자원 환경을 위한 인공신경망 설계 및 가속

2022년 8월

서울대학교 대학원

전기·정보공학부

최 혁 준

Neural Network Design and Acceleration for Resource-constrained Environments

지도교수 윤 성 로

이 논문을 공학박사 학위논문으로 제출함

2022년 8월

서울대학교 대학원

전기·정보공학부

최 혁 준

최혁준의 박사 학위논문을 인준함

2022년 8월

위 원 장 _____ (인)

부위원장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

Abstract

Deep learning methods have become very successful in various applications due to the availability of exponentially growing data and powerful computing resources. Due to remarkable performance, neural model has been applied to various edge devices such as mobile and embedded system. These edge devices usually suffer from constrained resources including computation and energy. To overcome this challenge, low-latency model design, model compression and acceleration are widely researched on both hardware and software side. In this dissertation, we introduce two methods with regard to low-latency model design on algorithm side. Designing compact and low-latency model is important to reduce required resources. For this reason, in aspect of algorithm, we introduced two model design methodology with neural architecture search (NAS) to find compact model: cell-based NAS and graph variational auto-encoder based NAS. Our cell-based NAS approach is based on Differentiable ARchiTecture Search (DARTS) which is well-known differentiable NAS method. Despite the popularity of DARTS, it has been reported that DARTS often shows unrobustness and sub-optimality. Through extensive theoretical analysis and empirical observations, we reveal that this issue occurs as a result of the existence of unnormalized operations. Based on our finding, we propose a novel variance-stationary differentiable architecture search (VS-DARTS). VS-DARTS makes the architecture parameters a more reliable metric for deriving a desirable architecture without increasing the search cost. In addition, we derive comparable architecture using VS-DARTS with soft constrained latency objectives. Another approach to find low-latency model is using graph generative models, which has been recently focused because of their efficiency. We proposed novel graph variational auto-encoder (VAE)

which shows dramatically improvements on cell-based search space. After our graph VAE extracted architectural information from the neural architecture, we conducted novel multi-objective NAS using extracted information for hardware-constrained environments. We showed that the proposed multi-objective NAS can derive various models close to Pareto optimal between latency and accuracy. Also, we evaluated our proposed method on various hardware platforms. In summary, this dissertation proposes two methods for improving performance of NAS, which can use compact and low-latency neural model for computing resource-constrained environments. The proposed methods were evaluated with various experimental studies.

keywords: Deep Learning, Deep Neural Network, Energy Efficiency, Neural Architecture Search, Hardware-aware NAS

student number: 2015-21001

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep neural networks have become very successful in various fields, namely, computer vision [66], natural language processing (NLP) [50], and recommendation system [83]. The success of deep learning is from two major reason: exponentially growing data and powerful computing resources.Because of its noticeable performance, deep neural networks has been applied to edge devices including smartphones [36], the Internet of Things (IoTs) [46], embedded systems for smart factory [7] or autonomous driving [25].

While the Moore's law is predicted to end on a single core, on System on a Chips(SoCs) like Apple's A series, performance is improving much faster than doubling in 2 years [64]. So, as time goes by, it is expected that research interest will gradually shift to edge devices such as mobiles and embedded systems using SoC rather than high-performance GPU servers.

However, paradoxically, the factors that brought about the success of deep learning obstruct deploying to edge devices. In other words, as the model size increases, deployment becomes increasingly difficult in computational and energy resource-constraint environments such as mobile devices or embedded systems.

Therefore, it is natural that the research on low-latency model design and acceleration is conducted in various fields to overcome these challenges. To reduce model size, a lot of researches are being studied in both the algorithm and hardware aspects. On the algorithm side, various techniques can be categorized into following areas: compact model design [57, 4, 67], quantization [16], pruning [56, 43] and tensor decomposition [48]. On the hardware side, many of acceleration techniques are conducted on various environments [85]. Also, various types of neural network accelerators, such as FPGA- and SoC-based accelerator, are actively developed to efficiently handle machine learning workloads [12].

In this dissertation, we introduce two methods with regard to design low-latency model on algorithm side. More specifically, we improved automating model design methodology to find compact model which didn't harm the performance in Chapter 3 and Chapter 4.

Neural architecture search (NAS) is automating neural architecture design, which is the on-going step in automating machine learning (AutoML). Although deep neural networks (DNNs) have become the popular choice of machine learning model for big data processing, it still requires an expert with experience in machine learning and domain knowledge to find a proper neural architecture for the target task. This challenge in designing an appropriate neural architecture calls for automation of the neural architecture search process, and consequently, Neural Architecture Search (NAS) continues to attract a significant amount of attention from the machine learning society. However, the extensive search space and the high evaluation cost of early NAS algorithms [54, 55, 87, 88] based on Evolutionary algorithm (EA) or Reinforcement Learning (RL), required an enormous amount of computational resources, often mounting up to hundreds and thousands of GPU hours. The introduction of the cell-based micro search space [88, 40] and parameter sharing [53] brought upon an im-

pressive reduction in the search cost of NAS. The family of one-shot NAS algorithms, all of which employ one or more of the techniques mentioned above, simultaneously evaluates all candidate architectures by training a supernet. The development of one-shot NAS algorithms greatly contributed to making NAS research more approachable. Among various one-shot NAS algorithms, DARTS [40], which is a gradient descent optimization-based one-shot search algorithm, has become the most widely benchmarked search algorithm because of its simple and intuitive methodological approach. Despite the popularity of DARTS, it has been reported that DARTS often searches for a sub-optimal candidate architecture, and that DARTS may fail miserably depending on the search space design [79]. Most of the works that attempt at amending the performance collapse of DARTS have placed the blame on the increase in the number of skip-connections.

In Chapter 3, we reveal an undiscovered factor that makes the architecture parameters unreliable for the selection rule: unnormalized outputs of intermediate nodes in the super-net. We provide extensive theoretical and empirical analyses to show how unnormalized outputs prevent the architecture parameters from accurately representing operation strengths.

Chapter 3 is based on the following paper:

- Hyeokjun Choe, Byunggook Na, Jisoo Mok, Sungroh Yoon, "Variance-stationary Differentiable NAS," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2021.

The key contributions of Chapter 3 are as follows:

- We investigate the naive continuous relaxation, which leads to the unfair competition in both intra- and inter-edge level, and is unveiling cause of that supernet falling in sub-optima in DARTS.

- To solve the gap between continuous relaxation-based search and the conventional architecture parameter magnitude-based architecture selection, we propose two methods: node normalization and variance-stationary continuous relaxation.

- We improve search performance and robust search of DARTS, specifically, the average accuracy of searched architectures increases by +0.50% on CIFAR-10.

Although NAS continues to attract significant attention from the machine learning society. The representation of architectures has become essential for enhancing NAS performance. Designing an appropriate neural architecture requires automation of the neural architecture search process. To examine how neural architectural properties and their performances are interconnected, NAS researchers are treating neural architecture as a graph and utilizing graph neural networks-based (GNNs-based) variational auto-encoders (VAEs) that represent graphs in continuous latent space to analyze the properties of the graph [44, 6, 82, 42].

However, we observe that the conventional graph generative NAS methods do not work appropriately on cell-based search spaces . Through theoretical analysis, we found that in conventional methods, node generation is performed using only the sub-graph information instead of the full graph information when encoding and decoding a graph with a branch structure. This issue is caused by the short expressive power of GNNs. Based on our findings, we propose a method that uses the entire graph information instead, thereby increasing the explosive power of graphs suitable for directed acyclic graphs (DAGs). In addition, we propose a predictor-based multi-objective NAS method that can derive several models on the Pareto frontier between inference latency and model accuracy on various hardware platforms instead of creating a single model such as a conventional predictor-based NAS. Our experimental

results show that node index embedding improves the reconstruction accuracy of the D-VAE-EMB [6] by 86.99% (13.01% to 100%) for NAS-Bench-201 [19]. Moreover, we show that a multi-objective search using our proposed method can output results very close to Pareto-optimal on various hardware platforms, thereby outperforming conventional GCN- and graph VAE-based methods.

In Chapter 4, we propose hardware-aware neural architecture optimization with graph VAE. In auto-encoder, we adapt our novel decoding strategy and observer which can improve the expressive power of Graph Neural Networks (GNNs). After train VAE, we searched out a set of architectures that is close to the Pareto-frontier through novel predictor-based multi-objective NAS.

Chapter 4 is based on the following paper:

- Hyeokjun Choe, Jeonghee Jo, Byunggook Na, Sungroh Yoon, "Hardware-aware Neural Architecture Search with Graph Variational Auto-Encoder," *in preparation*, 2022.

The Chapter 4 can be summarized as follows:

- We reveal that conventional DAG VAE usually fails to reconstruct on a cell-based search space, a problem caused by insufficient graph expressive power of DAG VAE.
- We propose a novel decode strategy for DAG VAE, a simple yet effective strategy to increase the performance of the graph VAE on a cell-based search space because our proposed method can remedy the shortage of graph expressive power in DAG VAE.
- We propose a novel predictor-based multi-objective NAS with a graph VAE, which can determine a set of architectures close to Pareto-optimal in various hardware environments.

The dissertation is organized as follows: Chapter 2 provides background and previous studies. In Chapter 3, we propose novel cell-based neural architecture search methods to improve the search performance and adapt searching hardware-aware neural architecture. Chapter 4 suggests hardware-aware NAS method with graph VAE using novel graph decoding strategy and scheme. Finally, in Chapter 5, we conclude this dissertation.

Chapter 2

Background

To put our works in a proper context and also to facilitate further understanding, we first provide a brief review of related work.

2.1 Neural Architecture Search

2.1.1 Previous Works on Differentiable NAS

Automation of the neural architecture search process has continuously attracted interest, but wide search space and high evaluation cost require high computational resources[87, 88, 39, 55]. After the cell-based search space[53] and parameter sharing[53] are announced, one-shot neural architecture search (NAS) methods, which train supernet to search and evaluate candidate architectures simultaneously, are widely researched[40, 70, 18]. One-shot NAS can be categorized by two groups: 1) differentiable architecture search which train supernet with continuously relaxed seach space[40, 38, 72, 9, 13, 79, 10, 68] and 2) sampling-based methods which sample the candidate architecture and stochastically update the supernet[18, 70, 86].

DARTS [40],which is the mile-stone research in the first category, has become the

most preferred baseline search algorithm because of its simple and intuitive methodological approach. Even though DARTS shows an impressive performance and computational efficiency, it has been reported that DARTS lacks stablility [79, 10] and often derives a sub-optimal architecture even in the search space where all possible architectures are evaluated [19]. By restricting the search space of DARTS, Zela *et al.* [79] empirically demonstrated the failure cases of DARTS. To make DARTS more robust against such failure cases, Zela *et al.* introduces a regularization term to the inner loop of the bi-level optimization. In the case of the study of Chen *et al.* [10], the instability issue of DARTS is addressed through the learning of weight parameters with respect to the perturbed architecture parameters.

Also, it has been empirically reported that the large number of skip connections in the final architecture derivation step results in a significant performance drop [79, 13, 38]. Chu *et al.* [14] argues that the skip connection operation in fact is performing two roles simultaneously: aiding the training of super-net and the candidate operation. Chu *et al.* separates these two roles of the skip connection operation from each other. However, they do not provide an additional analysis into the architecture selection rule based on the magnitude of α.

2.1.2 Preliminaries on DARTS

Differentiable architecture search (DARTS)[40] aims to search for a cell, which is repeatedly stacked to build the final architecture. In DARTS, two types of a cell are searched: a normal cell and a reduction cell. While the normal cell maintains the dimension of the input feature map, the reduction cell halves it and doubles the number of channels. These cells can be seen as a directed acyclic graph (DAG) with E edges and N nodes, which are associated with operations and feature maps, respectively. The intermediate nodes of a cell are computed by summing up all preceding

nodes: $x_j = \sum_{i<j} o^{(i,j)}(x_i)$. The output of a cell is computed by concatenating all intermediate nodes.

DARTS starts by constructing a super-net, in which each edge includes all candidate operations in the search space. To enable gradient-based optimization, DARTS continuously relaxes the discrete choice of an operation by applying the Softmax function over all candidate operations with the architecture parameters α:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(x) = \sum_{o \in O} \beta_o^{(i,j)} o(x), \tag{2.1}$$

where i and j indicate node indices. $\bar{o}$, the weighted summation of candidate operations, is referred to as a mixed operation. An example of a continuously relaxed cell in DARTS is illustrated in Figure 3.4(a). By applying the continuous relaxation scheme, the problem of architecture search can be equated with bi-level optimization problem as follows:

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \qquad \text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \mathcal{L}_{train}(w, \alpha). \tag{2.2}$$

During the search process, architecture parameters α and operation weights w are updated alternatingly. After the search process, the operation associated with the maximum β is selected per edge. Afterwards, the final architecture is derived by selecting top-2 edges per intermediate node by ranking the edges according to β of the selected operation.

2.2 Graph Variational Auto-Encoder

2.2.1 Graph Representation Learning

In general, a graph G with n nodes is described as node or vertices sets $\{v_i\}_{1\leq i\leq n}$, and edge sets $\{e_{ij}\}_{1\leq i,j\leq n}$, if any two different i-th node v_i and j-th node v_j are connected to each other. Recently, message passing (MP)-based operations for graphs [24] have widely used for graph representation learning in various domain [65, 80, 81, 23]. The most common MP frameworks are composed of two functions: $\mathcal{A}$ and $\mathcal{U}$, which means the aggregation and update function, respectively. $\mathcal{A}_t$ aggregates the localized message of each node representation $\mathbf{x}_i$ in t-th timestep, and $\mathcal{U}_t$ updates node representations of the next $(t+1)$-th timestep based on the aggregated messages. More recent MP-based works showed that the discriminative power of subgraphs is closely related to overall performances of an MP framework in various graphs [71, 45, 49, 51].

In terms of an effective MP algorithms, a number of MP-based researches [69, 26, 1] have concerned that the discriminative or expressive power is the key factor in various types of graph tasks. Some studies [71, 45, 41] pointed out that some aggregation functions failed to distinguish a pair of two different sub-graphs, and eventually it caused harmful effect on overall performances. Several recent works [75, 21] suggested that differentiating nodes can be one of the solution.

2.2.2 Variational Auto-encoder for Graph

Several strategies were proposed for unsupervised learning on graphs. VGAE [31] applied the variational auto-encoder (VAE) [30] to obtain meaningful latent features on graphs. GraphVAE [59] also adopted VAE for small molecular graphs. Graph U-Nets [22] and automatic graph encoder-decoder (AGMC) [78] are another examples

of encoder-decoder model on graphs. Although a fair number of graph types have no intrinsic node orderings, it is not the case for some tasks including neural architecture graphs. Gated graph sequence neural networks (GGS-NNs) is one of the early works for producing graphs in sequences[37], graph convolutional recurrent network (GCRN) [58] to predict sequential structures of graphs using recurrent neural network (RNN).

2.2.3 Neural Architecture Search (NAS) with Generative Models

Many of researches are proposed to reduce evaluation time, because neural network evaluation to obtain performances requires expensive computational costs [44, 53, 8, 74, 19]. One of the approach to accelerate neural architecture evaluation is using performance predictor instead of training model from scratch. NAO [44] treats the neural architectures as the discrete string sequences, and search improved architecture using LSTM-based encoder, decoder and multi-layer perceptron (MLP)-based performance predictor.

Several recent studies [82, 73, 20, 6, 73] formulated neural architecture search (NAS) as a graph generation task. Zhang et al., [82], and Chatzianastasis et al., [6] describes a neural architecture as a DAG, and use VAE to explore the latent space of DAGs. Deviating from the simultaneous MP scheme suitable for undirected graphs, Zhang et al. [82] proposed D-VAE which use an asynchronous MP scheme which sequentially encode each node in computational order on directed acyclic graphs (DAGs). Chatzianastasis et al., [6] proposed D-VAE-EMB which use the learnable operation embeddings instead of fixed operator encoding to make smoother representations of the architectures. But there is a limitation that these methods are not evaluated on cell-based search space. SVGe [42] proposed a two-sided graph VAE which improves the smoothness of latent representations and reconstruction perfor-

mances even on cell-based search space, but they did not provide sufficient explaination about the reason of D-VAE's fail cases on cell-based search spaces [42].

2.2.4 Preliminaries on VAE for DAGs

Neural Architectures as Graphs

Let vertices mean operations while directed edges mean signal flows. Then, with the set of nodes V and of edges E, the computational graph g of an architecture A can described as $g_A = (V, E)$ [82, 6]. Usually, conventional DAG VAE methods convert operation and connection to node and edge, respectively [82, 6]. The converting method can be naturally applicable for ENAS dataset[53] and NAS-Bench-101[74]. On the other hand, in NAS-Bench-201[19], node and edge represent feature-map and operation respectively. BRP-NAS [20], which is one of the graph-based NAS research, swap the node and edge of the NAS-Bench-201 and we also followed.

Variational Auto-Encoder for DAGs

Previous NAS with DAG VAE studies proposed variational auto-encoder (VAE) for DAG using asynchronous MP scheme, the details are as follows [82, 6]. Encoder $q_\pi(z|g)$, one of the VAE components, approximate true posterior $p_\theta(z|g)$ which is the probability distribution of latent representation z given graph g. In other words, encoder map the graph g in G into latent space Z. Decoder $p_\theta(G|Z)$, another component of VAE, is a network for generating a graph g from given hidden representation z. Loss L is defined as Eq. 2.3:

$$L(g, \theta, \pi) = \mathbb{E}_{\boldsymbol{z}}[log\ p_\theta(g|\boldsymbol{z})] - D_{KL}(q_\pi(\boldsymbol{z}|g)||p_\theta(\boldsymbol{z})), \tag{2.3}$$

where θ and π mean parameters each of p_{theta} and q_{π} while D_{KL} denotes Kullback–Leibler divergence [34].

Encoder

As mentioned above, the encoder $q_{\pi}(\boldsymbol{z}|g)$ is a network that projects the input graph g into the continuous probabilistic latent space Z. Latent representation $\boldsymbol{z}$ is derived by encoder $q_{\pi}(\boldsymbol{z}|g)$ as Eq. 2.4 and Eq. 2.5:

$$\boldsymbol{z} \sim q_{\pi}(z|g) = \mathcal{N}(0,1) \times \sigma_g^2 + \mu_g, \tag{2.4}$$

where $\mathcal{N}$ denotes normal distribution, μ and σ^2 indicate mean and variance of the latent distribution, respectively. μ and σ^2 can be obtained by neural networks ψ_1 and ψ_1 as Eq. 2.5:

$$\mu_g, \sigma_g^2 = \psi_1(h_g), \psi_2(h_g), \tag{2.5}$$

where h_g is the hidden state of the graph g.

Graph state h_g is derived from the hidden state of the output node h_{output}. By GNN with asynchronous MP scheme [82], the hidden states of node $\{h_0, h_1, h_2, ..., h_N\}$ are obtained by aggregation function $\mathcal{A}$ and update function $\mathcal{U}$ in topological order. $\mathcal{A}$ derives h_r^{in} which is the aggregation of incoming messages from its a set of predecessor nodes followed by Eq. 2.6:

$$h_r^{in} = \mathcal{A}(h_r,\ \{h_s : s \to r\}), \tag{2.6}$$

where h_r and h_s represent the hidden state of *receiver node* v_r and the *sender node* v_s, respectively. As previous studies, we use a gated sum [82, 6].

Then, the receiver node's hidden state h_r is renewed by update function $\mathcal{U}$:

$$h_r = \mathcal{U}(\mathcal{O}(x_r),\ h_r^{in}), \tag{2.7}$$

where x_r is the operation type of v_r. $\mathcal{O}(x_r)$ is operation embedding function proposed by Chatzianastasis et al., [6]. As in previous studies, we use GRU [15] as update function $\mathcal{U}$ [82, 6].

Decoder

The decoder has the role that translates the latent representation $\boldsymbol{z}$ into graph g. The DAG decoding strategy proposed by Zhang et al., [82] is as follows:

- **Step 1.** Determine the operation type x_k of the node v_k, which is to be generated, through MLP $f_{add\_node}(h_g)$ where $h_g := h_{k-1}$.
- **Step 2.** Connect v_k with previously generated node $v_{prev} \in \{v_0, v_1, v2, ..., v_{k-1}\}$ through f_{add_edge} and its input (x_k, h_{prev}) in node index order.
- **Step 3.** If a new edge is generated in step 2, update the hidden state h_k of the currently generated node v_k.
- **Step 4.** Decoder stops when the number of nodes in graphs reach to the predefined limit or the operation type 'END' node is generated. Otherwise, go back to step 1 for next node generation.

Chapter 3

Neural architecture search for resource-constrained environment

3.1 Introduction

Neural architecture search (NAS) is drawing a considerable amount of attention from the machine learning society as an attractive alternative to hand-crafting a neural architecture [87, 88, 53, 40]. Among various NAS algorithms, DARTS [40], a gradient descent optimization-based search algorithm, has become the most widely benchmarked search algorithm due to its low computational cost and intuitive methodological approach. Despite the popularity of DARTS, it has been reported that DARTS often searches for a sub-optimal candidate architecture [38, 13, 79, 68].

Wang *et al.* [68] argue that the failure cases of DARTS occur because architecture parameters, on which the architecture selection rule of DARTS is based, do not faithfully reflect the true operation strengths. Thus, Wang *et al.*propose a new perturbation-based selection rule, which does not use architecture parameters. However, the perturbation-based selection requires progressive tuning, which results in

a significant increase in the search cost. If the search process can be fixed, so that the architecture parameters function as a reliable metric without additional computational cost, it becomes a more desirable solution for aligning the search process and the selection rule of DARTS.

In this work, we reveal an undiscovered factor that makes the architecture parameters unreliable for the selection rule: unnormalized outputs of intermediate nodes in the super-net. We provide extensive theoretical and empirical analyses to show how unnormalized outputs prevent the architecture parameters from accurately representing operation strengths. To address the problem of unnormalized outputs, we propose **node normalization**. Applying node normalization produces an undesirable side-effect, named the gradient imbalance problem, which is resolved with layer-wise adaptive control (LARC) [76]. We also introduce $\sqrt{\beta}$-continuous relaxation to improve the training stability of the proposed method. Our method that encompasses all of the amendments is named Variance-stationary DARTS (VS-DARTS); the differences between DARTS and VS-DARTS are illustrated in Figure 3.4. VS-DARTS focuses on making appropriate changes to DARTS, such that architecture parameters can function as intended.

VS-DARTS improves the search performance of DARTS by 0.57%p and that of DARTS +PT [68] by 0.18%p, achieving a competitive test accuracy on the CIFAR-10 dataset. Furthermore, the standard deviation of test errors is reduced in VS-DARTS, which indicates the robustness of the search performance. When evaluated across different datasets and search spaces, VS-DARTS once again consistently searches for a successful neural architecture.

The key contributions of this chapter 3 are as follows:

- We investigate the naive continuous relaxation, which leads to the unfair competition in both intra- and inter-edge level, and is unveiling cause of that super-

net falling in sub-optima in DARTS.

- To solve the gap between continuous relaxation-based search and the conventional architecture parameter magnitude-based architecture selection, we propose two methods: node normalization and variance-stationary continuous relaxation.

- We improve search performance and robust search of DARTS, specifically, the average accuracy of searched architectures increases by +0.50% on CIFAR-10.

3.2 Issue of DARTS Architecture Parameter

Previous works mainly attributed the DARTS failure to the optimization of super-net, but the problem of the architecture selection process has largely been overlooked [68]. In this section, we discuss Wang *et al.* [68], which is the one of the first works to address the problem of architecture parameters . Wang *et al.*demonstrate why the magnitude of architecture parameters in the DARTS formulation is not suitable for indicating the contribution of each operation to the super-net's performance. According to Wang *et al.*, when the problem of skip operation domination occurs in the hastily optimized super-net, the architecture parameters are not aligned with the discretization accuracy, *i.e.*the accuracy of the trained super-net when a single operation is selected for a randomly selected edge, while all other edges remain the same. Wang *et al.*thus proposed a new selection rule that re-evaluates the super-net after removing a single operation per edge and selects the operation that yields the largest drop in the super-net's validation accuracy upon removal; this selection rule is coined the perturbation-based selection rule (DARTS+PT) and does not directly utilize architecture parameters. However, to successfully execute the proposed perturbation-based selection rule, it requires the progressive tuning process that incurs a considerable

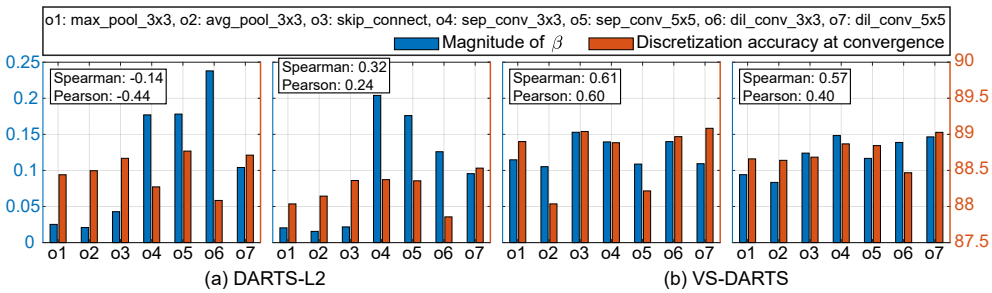

Figure 3.1: The magnitude of β vs. discretization accuracy (%) at convergence of 2 edges from a trained (a) DARTS-L2 supernet and (b) VS-DARTS supernet. While optimized β of the VS-DARTS supernet faithfully reflects the discretization accuracy of the corresponding operation, *i.e.*operation strength, that of the DARTS-L2 supernet fails to do so.

amount of additional computation cost.

3.3 Lack of Reliability of β

From here on, we refer to the inability of β to appropriately reflect the operation strengths as *unreliability of* β. Eliminating the unreliability of β is crucial for improving the accuracy of the selection rule based on the magnitude of β. If we can improve the reliability of β, it becomes unnecessary to employ additional techniques, such as progressive tuning [68], which causes a non-negligible increase in the search cost, during the search process.

As mentioned in Section 3.2, if the input feature maps to each operation are close to being optimal, β_{skip} begins to grow larger than β of other operations. However, we demonstrate that the previously-observed dominance of skip connection caused by the optimized input feature maps is insufficient for explaining the unreliability of β. We train a DARTS super-net with strong regularization (DARTS-L2) by increasing the weight decay value from 3e-4 to 5e-3, such that the feature maps do not hastily converge to optimal in the middle of the search process. Figure 3.1 shows the magnitude of β and the operation strength in 2 randomly selected edges of the super-net,

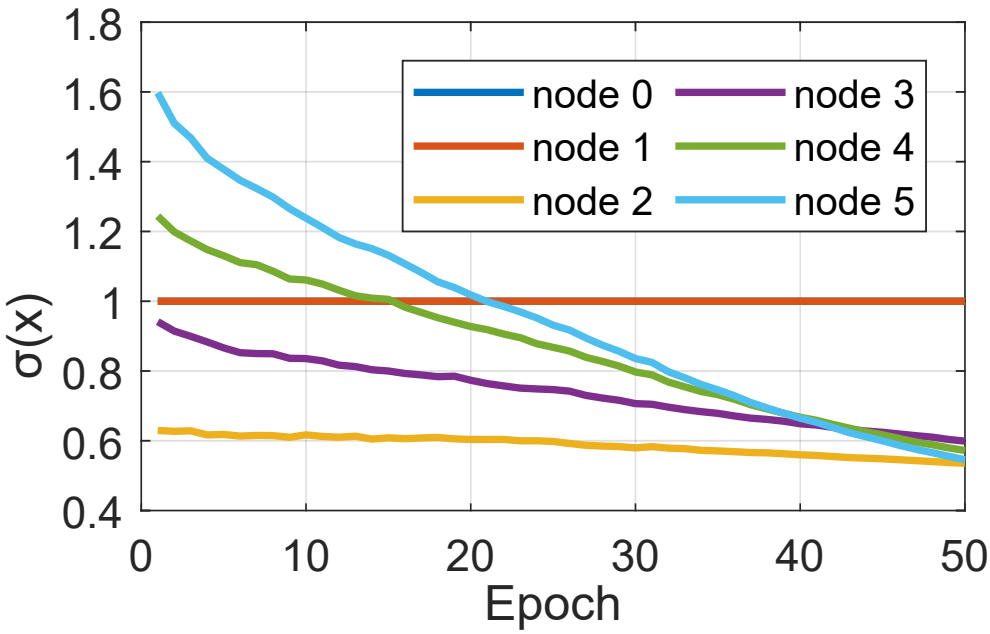

Figure 3.2: Change in the variance of the each node output feature map during the DARTS search process.

where the operation strength is represented with the discretization accuracy at convergence [68]. As can be seen from Figure 3.1(a), even after increasing the regularization effect, the magnitude of β and the operation strength in the DARTS-L2 super-net are not aligned, indicating that the unreliability of β still remains.

We further analyze the computational flow in the DARTS super-net to reveal another reason for **unreliability of β problem**. According to Eq.(1), the influence of an operation on the $\bar{o}$ can be estimated by $\beta_o o(x)$. Therefore, for β to be a trustworthy measure of the operation strength, the outputs from $o(x)$ must be scaled to a similar range of values. However, as an example, when the scale of $o(x)$ is far smaller than 1, o has a smaller influence over $\bar{o}$, even if β_o is the highest among β of candidate operations.

We observe that in DARTS, the scales of $o_{\text{skip}}(x)$ differs from those of the other operations. We use the variance of outputs from $o(x)$ to represent their scales, since

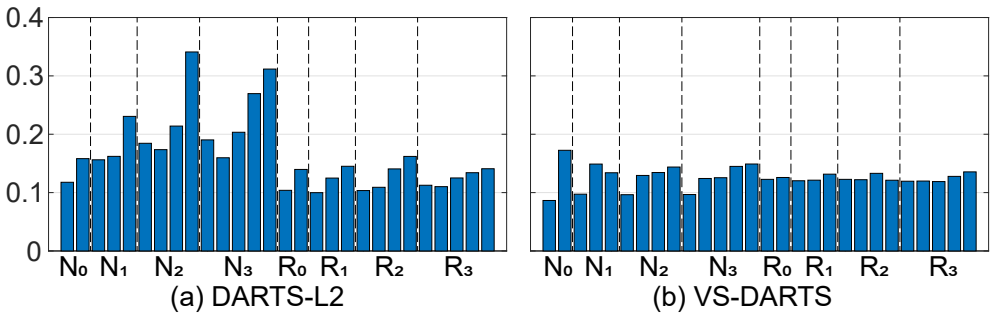

Figure 3.3: The magnitude of β_{zero} of (a) DARTS-L2 and (b) VS-DARTS. The bar graphs in N_i and R_i correspond to the incoming edges into the i-th node of the normal cell and of the reduction cell, respectively.

they have the mean of zero. Figure 3.2 shows the output feature map variance for each node of a randomly selected cell during DARTS super-net training. We find that while the outputs of other operations are normalized to $(0, 1)$ due to their last normalization layer, the output of the skip operation is not normalized. There exist two reasons why the output of the skip operation cannot normalized in DARTS: 1) the input feature map to the skip operation is the output of the previous node, which is unnormalized as shown in Figure 3.2 and 2) the skip operation itself does not have the ability to normalize it. If nodes remain unnormalized as in DARTS, to select an operation based solely upon the magnitude of β becomes unreliable and thus results in a sub-optimal architecture. Therefore, we proposed VS-DARTS, which is described in Section 3.4, and VS-DARTS makes β more reliable as shown in Figure 3.1(b).

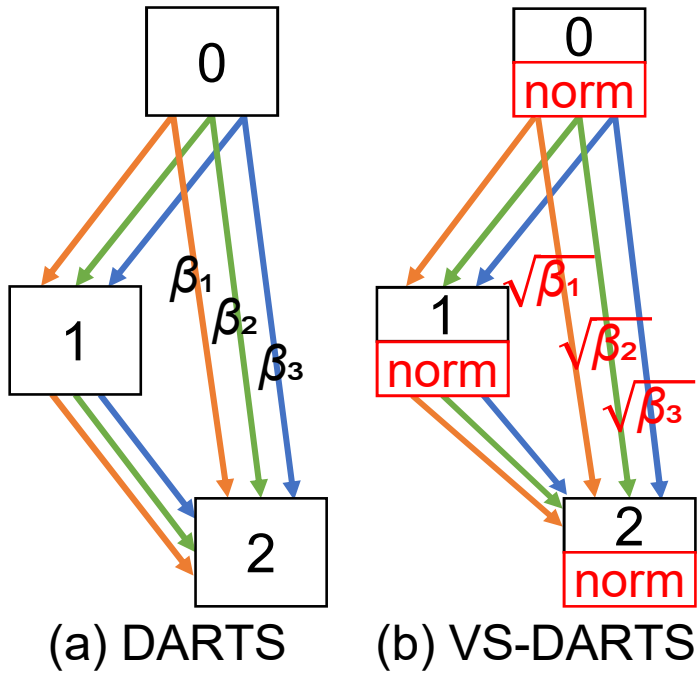

Figure 3.4: Overview of (a) DARTS and (b) VS-DARTS (proposed) which includes node normalization and $\sqrt{\beta}$-continuous relaxation, depicted in red colors.

3.4 Variance-stationary DARTS (VS-DARTS)

3.4.1 Node Normalization

To remedy the problems caused by the unnormalized nodes, we propose a simple yet effective method of normalizing the output of every intermediate node as follows:

$$\hat{x_j} = \frac{x_j - \mu_{x_{j,B}}}{\sigma_{x_{j,B}}}, \text{ where } x_j = \sum_{i<j} \bar{o}^{(i,j)}(x_i), \tag{3.1}$$

$\hat{x_j}$ is the normalized output feature map of node N_j, and $\mu_{x_{j,B}}$ and $\sigma^2_{x_{j,B}}$ are the mean and the variance of x_j within a single mini-batch, respectively. Applying the proposed node normalization to DARTS prevents the input to the skip operation from being unnormalized. Consequently, by scaling outputs of $o(x)$ to a similar range, node normalization contributes to amending the unreliability of β.

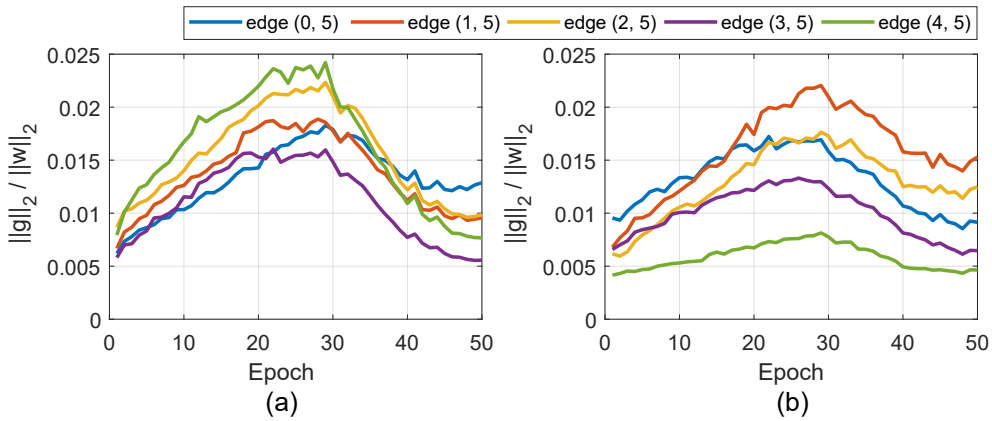

Figure 3.5: The change in the scale of gradient of 3x3 separable convolution operation at last intermediate node (a) before and (b) after applying node normalization.

Node normalization additionally addresses the problem of suppression by zero operation. The zero operation was initially introduced to indicate the lack of connection between two nodes [40]. Against the initial expectation, the zero operation is in fact performing the role of scaling output feature maps in DARTS. As shown in Figure 3.3(a), without node normalization, the average β_{zero} of incoming edges grows large to make the variances of each intermediate node similar. The large value of β_{zero} in turn suppresses β of other operations, which we call suppression by zero. The unreliability of β exacerbated by this phenomenon. It can be clearly observed from Figure 3.3(b) that applying node normalization effectively prevents β_{zero} from exploding. Under the architecture selection rule based on the magnitude of β, we believe that node normalization is crucial for DARTS-based algorithms to function properly.

3.4.2 Remedying Gradient Imbalance

While node normalization is effective at making β reliable, from the perspective of super-net training, it has a minor side-effect: the gradient imbalance problem. In Figure 3.5(a) and (b), we visualize the gradients with respect to the weights of the 3x3

separable convolution operation before and after applying node normalization to empirically support the occurrence of the gradient imbalance problem. In Figure 3.5(a), the amount of gradient against weight of each edges are similar scale in early epochs, while not in Figure 3.5(b). If a different amount of gradient signal is delivered to each edge, operations in the edge with large gradient will likely converge faster than others. Therefore, the scale of gradient should be similar in the early stage of search for fair competition among different edges. The reason of the gradient imbalanced problem is that node normalization change the scale of node output feature maps.

Node normalization scales the output feature maps using their $sigma$ as in Eq.(3.1), and scaling the feature maps also scales the gradient with respect to the corresponding feature maps. For instance, as in Figure 3.2, when σ of the unnormalized feature map is 1.6, the gradient is divided by 1.6 and becomes smaller; on the contrary, when σ is 0.6, the gradient becomes larger. This results in the gradient becoming unbalanced across incoming edges of a single intermediate node.

We deal with the gradient imbalance problem by applying the operation-wise adaptive learning rate. This adaptive learning rate scheme is realized by Layer-wise Adaptive Rate Control (LARC) [76]. Through LARC, the local learning rate for each operation is computed by multiplying the global learning rate by $||w||/||g||$, eliminating the above issue entirely. Therefore, LARC resolves the gradient imbalance problem caused by node normalization, and applying node normalization and LARC together is important for improving the reliability of β.

3.4.3 $\sqrt{\beta}$-Continuous Relaxation

Aside from improving the reliability of β, we propose an additional method to improve the search performance of DARTS: $\sqrt{\beta}$-continuous relaxation. Figure 3.6 shows the change in $\sigma(\bar{o}(x))/\sigma(x)$, where $\sigma^2(\bar{o}(x))$ and $\sigma^2(x)$ denote the variance of the

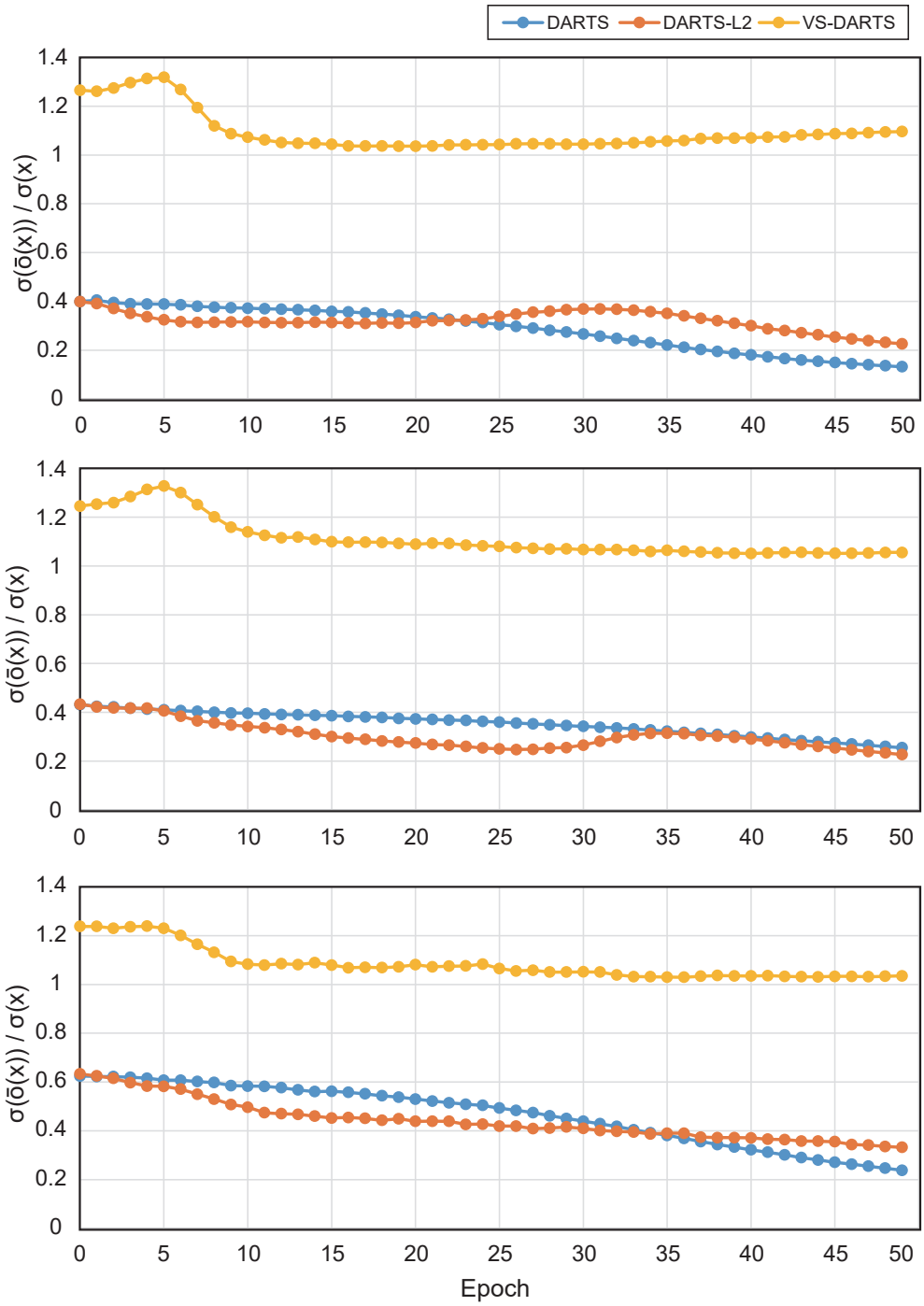

Figure 3.6: The ratio of variance of the output of the mixed operation to that of the input of the mixed operation in a randomly selected three edges.

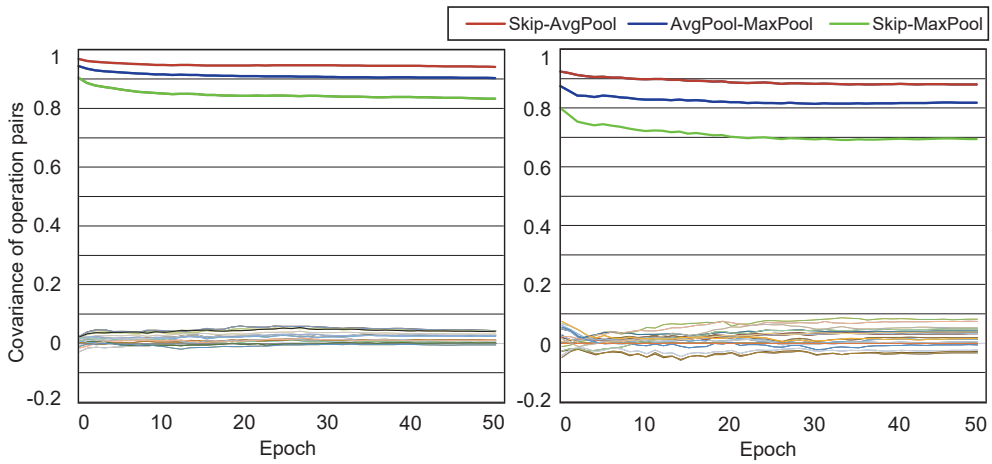

Figure 3.7: Covariances between each pair of operations in the search process using DARTS with node normalization.

output and input of the mixed operation respectively, as the search process proceeds. We observe that the ratio starts with 0.4 and decreases monotonically in DARTS. Through this observation, we want to answer the following questions: 1) why the scale of the output feature map is smaller than that of the input and 2) why the ratio of the output to the input varies in the mixed operation with the conventional DARTS continuous relaxation.

To address the first question, let us denote the variance of a feature map generated by the mixed operation as $\sigma^2(\bar{o}(x))$ which is expressed as:

$$\sigma^2(\bar{o}(x)) = \sum_{o \in O} \beta_o^2 \sigma^2(o(x)) + \sum_i \sum_{j \neq i} \beta_{o_i} \beta_{o_j} Cov(o_i(x), o_j(x)), \tag{3.2}$$

where Cov denotes the covariance between the feature maps generated by two different operations within the mixed operation. We assume that every output feature map of the operations is normalized. The ratio of the variance of the output to that of the input of the mixed operation is 1 if and only if all of the covariances are equal to 1. We empirically show that this condition is never satisfied by plotting the covariances

between each pair of operations in DARTS with node normalization in Figure 3.7. The covariances between the unparametrized operations stay above 0.7 throughout the search process, because the feature maps of unparametrized operations are highly correlated. On the contrary, covariances between all operations and parametrized operations remain close to zero as a result of the negligible correlation between them. This observation provides an answer to the first question.

Based on Figure 3.7, we also find that the covariances between unparametrized operations become stationary approximately after the first 10 epochs. Once the covariances between unparametrized operations become stabilized, the covariance term in Eq.(3.2) can be treated as a constant C. Therefore, with the proposed node normalization, $\sigma^2(\bar{o}(x))$ can now be approximated as:

$$\sigma^2(\bar{o}(x)) \approx \sum_{o \in O} \beta_o^2 \sigma^2(o(x)) + C \approx \sum_{o \in O} \beta_o^2 + C. \tag{3.3}$$

During the search process, $\sigma^2(\bar{o}(x))$ is bound to change because values of β no longer follow the uniform distribution and start to vary. Consequently, in the mixed operation with node normalization, although the variance of the input feature map $\sigma^2(x)$ is equal to 1 because of node normalization, the variance of the output feature map $\sigma^2(\bar{o}(x))$ changes during the search process. This observation explains why the ratio of the output to the input varies in the mixed operation, providing an answer to the second question.

Batch normalization is a commonly adopted technique to make the ratio between the input and the output stable, and it also keeps the input and the output at a similar level. However, directly applying batch normalization causes the unreliability of β because of re-scaling. Before Node Normalization is adapted to j-th node x_j, the

feature map of node is as follows:

$$x_j = \sum_{i<j} \bar{o}^{(i,j)}(\hat{x}_i) = \sum_{i<j} \sum_{o \in O} \beta_o^{(i,j)} o(\hat{x}_i). \tag{3.4}$$

Simply introducing Batch Normalization to the above expression yields:

$$x_j = \sum_{i<j} BN(\bar{o}^{(i,j)}(\hat{x}_i)) = \sum_{i<j} \frac{\bar{o}^{(i,j)}(\hat{x}_i) - \mu_{\bar{o}^{(i,j)}(\hat{x}_i)}}{\sigma_{\bar{o}^{(i,j)}(\hat{x}_i)}} = \sum_{i<j} \sum_{o \in O} \frac{\beta_o^{(i,j)} o(\hat{x}_i)}{\sigma_{\bar{o}^{(i,j)}(\hat{x}_i)}}, \tag{3.5}$$

where BN means Batch Normalization and $\mu_{\bar{o}^{(i,j)}(x_i)}$ is negligible because all operations are normalized. According to Eq. (3.5), introducing batch normalization to mixed operation rescales $\beta o(x)$. Therefore, comparing the values of $beta$ once again leads to the lack of reliability of β problem because the variances of edges are almost always different.

To aid training of the super-net by mimicking the effect of batch normalization, we instead propose $\sqrt{\beta}$-continuous relaxation that substitutes β in the continuous relaxation with $\sqrt{\beta}$. Let us denote the mixed operation with $\sqrt{\beta}$-continuous relaxation as $\tilde{o}(x)$. Then, Eq.(3.3) is modified as:

$$\sigma^2(\tilde{o}(x)) \approx \sum_{o \in O} \sqrt{\beta_o}^2 \sigma^2(o(x)) + C \approx \sum_{o \in O} \sqrt{\beta_o}^2 + C = \sum_{o \in O} \beta_o + C = 1 + C. \tag{3.6}$$

Eq.(3.6) indicates that $\sigma^2(\tilde{o}(x))$ is maintained as a constant 1+C, while $\sigma^2(x)$ is 1. If C is kept small, then $\sqrt{\beta}$-continuous relaxation can obtain the effect of batch normalization while avoiding another cause of the unreliability of β problem. This property is empirically supported by Figure 3.6. As shown in Figure 3.6, the ratio of the variance of the output of the mixed operation to that of the input of the mixed operation is close to 1 after 10 epochs when using node normalization and $\sqrt{\beta}$-Continuous Re-

laxation, which implies C in Eq.(3.6) remains relatively unchanged after the early stages in the search process. In DARTS, this ratio monotonically decreases, and in DARTS-L2, this ratio is visibly quite volatile; but in VS-DARTS, we observe that the variances of the input and the output feature stay close to 1, making the optimization landscape smoother.

3.5 Experimental Results

3.5.1 Settings

All of our experiments are conducted using a single NVIDIA Tesla V100 GPU. For the search process, following Zela *et al.* [79], we use a larger weight decay factor than default DARTS [40] to account for the significant performance drop caused by the increase in the number of skip connections. The remaining hyperparameters of the search process are kept the same as those of default DARTS [40]. The searched architecture is retrained for final evaluation, which is the standard practice in NAS. We follow the experimental protocols of default DARTS [40] for the hyperparameters of the retrain process.

Settings for Search (CIFAR-10): Our search algorithm is evaluated under the cell-based micro search space [40], where the cell consists of two input nodes from two previous cells and four intermediate nodes, and its output is a concatenation of the intermediate nodes. For a fair comparison, following usual settings of DARTS [40], the super-net consists of eight cells, *i.e.*, six normal cells and two reduction cells, and its first stem layer is based on 3x3 convolutional layer with 16 initial channels. The search process is executed for 50 epochs. We use 64 as a batch size. Weights of operations in the super-net are optimized by momentum SGD, with a momentum of 0.9, an initial learning rate of 0.1, and a L2 regularization (weight decay factor) of 5E-

3, where the learning rate is annealed down to zero according to a cosine schedule. Following Zela *et al.* [79], we use a larger weight decay factor than default DARTS to account for the serious performance drop due to the increase in the number of skip connections. Architecture parameters α are optimized by the Adam [29]. We use a fixed learning rate of 0.0003, a momentum of (0.5, 0.999), and a L2 regularization factor of 1E-3.

Settings for Retraining (CIFAR-10): For the evaluation of the searched architectures, we follow DARTS settings [40] to train the final architectures on CIFAR-10, split the dataset 50,000 and 10,000 for train and test set, respectively. The network with 36 initial channels consists of 20 cells, *i.e.*, 18 normal cells and two reduction cells; each type of the cells share the cell architectures obtained in the search process. In the evaluation, the network including an auxiliary loss is trained for 600 epochs. We used 96 as a mini-batch size. The momentum SGD optimizer is used, with an initial learning rate of 0.025 following cosine scheduled annealing, a momentum of 0.9, a L2 regularization factor of 3E-4, and a norm of gradient clipping at 5. With aforementioned data augmentation techniques, cutout[17] is additionally used, and drop-path with a rate of 0.2 is used for regularization.

Settings for Retraining (ImageNet): We evaluate two architectures on ImageNet, where 1.28M training images and 5K test images are included: the best architecture searched from CIFAR-10 and the architecture directly searched on ImageNet using proxy dataset proposed by Na *et al.* [47]. We follow PC-DARTS [72] settings, while using four V100 GPUs for ImageNet training with batch size of 1536. The network consists of 14 cells: 12 normal cells and two reduction cells. Before stacking the cells, the network with 48 initial channels is equipped with stem layers which are composed of three 3x3 convolution layers of stride 2 to reduce 224×224 input images to 28×28. In the training process, the network is trained from scratch for 250

Table 3.1: Comparison of architectures searched by various NAS algorithms on CIFAR-10. Cost refers the search cost with GPU days.

Method	Params (M)	Test Err. (%)	Cost
NASNet-A[88]	3.3	2.65	2000
ENAS[53]	4.6	2.89	0.5
AmoebaNet-B[55]	2.8	2.55	3150
PNAS[39]	4.6	3.41	225
DARTS^{1st}[40]	3.3	3.00	1.5
DARTS^{2nd}[40]	3.3	2.76	4.0
SNAS[70]	2.8	2.85 ± 0.30	1.5
GDAS[18]	3.4	2.93	0.2
P-DARTS[9]	3.4	2.50	0.2
PC-DARTS[72]	3.6	2.57	0.1
R-DARTS[79]	-	2.95 ± 0.21	1.6
FairDARTS[13]	3.3 ± 0.5	2.54 ± 0.05	0.4
DARTS-[14]	3.5	2.59 ± 0.08	0.4
PR-DARTS[86]	3.4	2.32	0.17
DARTS+PT[68]	3.0	2.61 ± 0.08	0.8
DrNAS[11]	4.1	2.46 ± 0.03	0.6
VS-DARTS (avg)	3.4 ± 0.3	2.50 ± 0.05	0.4
VS-DARTS (best)	3.38	2.43	0.4

epochs, and the warm-up process is applied in the first 5 epochs. The momentum SGD optimizer is used, with an initial learning rate of 0.5 which is decayed down to zero linearly, a momentum of 0.9, a L2 regularization factor of 3E-5, and a norm gradient clipping at 5. Label smoothing with a rate of 0.1 and an auxiliary loss are used to enhance the training.

3.5.2 Results in DARTS Search Space

CIFAR-10

In Table 3.1, we compare the search performance of VS-DARTS on the CIFAR-10 dataset against that of various NAS Algorithms.

To validate the stability of the search performance, we repeat the search and re-train processes three times with different seeds; hence, we report the average performance of our searched architectures with its standard deviation. As shown in Table 3.1, the performance of VS-DARTS architectures is comparable to the state-of-the-art performance. Figure 3.8 shows the best derived architecture searched by VS-DARTS on DARTS search space. and The other architectures are visualized in Figure 3.9.

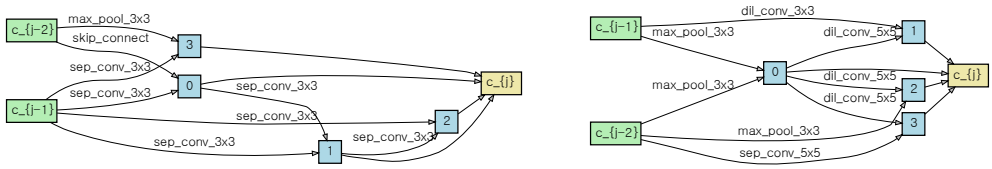

Figure 3.8: Normal(left) and reduction(right) cells derived by VS-DARTS on CIFAR-10 and DARTS search space. Validation results are shown in Table 4.1(best) and Table 4.2.

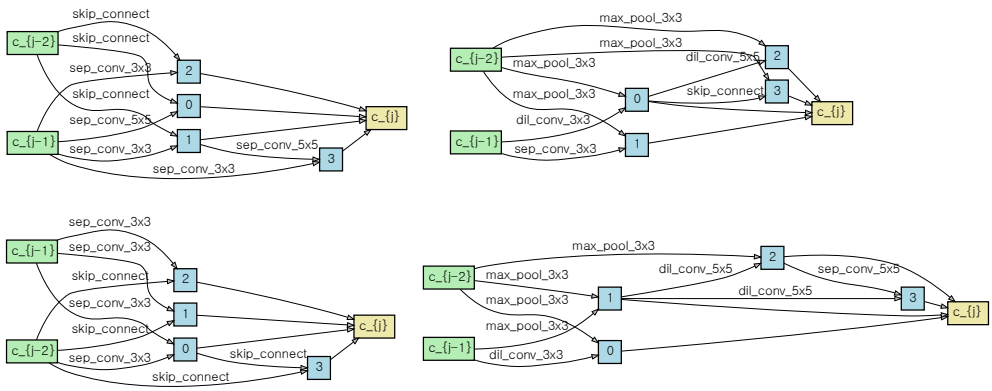

Figure 3.9: Two additional normal(left) and reduction(right) cells derived by VS-DARTS on CIFAR-10 and DARTS search space. These architectures are used for VS-DARTS results(avg) in Table 4.1.

Table 3.2: Performance comparison of architectures on ImageNet (mobile setting).

Method	Search space	Params (M)	FLOPs (M)	Test Err. (%) Top1	Top5
MobileNet[28]	-	4.2	569	29.4	10.5
ShuffleNet2×(v2)[84]	-	∼5	524	25.1	-
NASNet-A[88]	cell	5.3	564	26.0	8.4
PNAS[39]	cell	5.1	588	25.8	8.1
DARTS^{2nd}[40]	cell	4.7	574	26.7	8.7
SNAS (mild)[70]	cell	4.3	522	27.3	9.2
PC-DARTS (CIFAR-10)[72]	cell	5.3	586	25.1	7.8
GDAS (CIFAR-10)[18]	cell	5.3	581	26.0	8.5
P-DARTS (CIFAR-10)[9]	cell	4.9	557	24.4	7.4
P-DARTS (CIFAR-100)[9]	cell	5.1	577	24.7	7.5
FairDARTS-B (CIFAR-10)[13]	cell	4.8	541	24.9	7.5
PR-DARTS (CIFAR-10)[86]	cell	5.0	543	24.1	7.3
DARTS+PT (CIFAR-10)[68]	cell	4.6	-	25.5	8.0
Once-For-All†[5]	-	4.4	230	24.0	-
MnasNet-92†[62]	stage-wise	4.4	388	25.2	-
ProxylessNAS†[4]	layer-wise	7.1	581	24.9	7.5
BigNAS-L†[77]	layer-wise	6.4	586	20.5	-
EfficientNet-B1[61]	-	7.8	734	20.8	-
VS-DARTS (CIFAR-10)	cell	5.3	589	24.8	7.5
VS-DARTS +Proxy†[47]	cell	5.7	640	24.7	7.7
VS-DARTS +Proxy(HW-aware)†	cell	5.3	587	24.6	7.5

† This mark means that the architecture is directly derived on ImageNet dataset. Otherwise, the architecture is derived on CIFAR-10 or CIFAR-100 and transferred to ImageNet.

Table 3.3: Performance comparison (test error (%)) across three datasets and four search spaces, which are constrained from the cell-based search space [79]. †: evaluated ourselves while settings are the same as cited paper.

Benchmark		DARTS	R-DARTS(L2)	DARTS-ES	DARTS-ADA	PR-DARTS† [86]	DARTS+PT [68]	VS-DARTS (ours)
C10	S1	3.84	**2.78**	3.01	3.10	3.26	3.50	**2.58**
	S2	4.85	3.31	3.26	3.35	3.63	**2.79**	**2.57**
	S3	3.34	**2.51**	2.74	2.59	3.99	**2.49**	2.52
	S4	7.20	3.56	3.71	4.84	**2.59**	2.64	**2.49**
C100	S1	29.46	24.25	28.37	**24.03**	32.74	24.48	**23.87**
	S2	26.05	**22.24**	23.25	23.52	32.52	23.16	**21.49**
	S3	28.90	23.99	23.73	23.37	26.96	**22.03**	**22.03**
	S4	28.85	21.94	21.26	23.20	**20.80**	**20.80**	**20.69**
SVHN	S1	4.58	4.79	2.72	**2.53**	3.33	2.62	**2.37**
	S2	3.53	**2.51**	2.60	2.54	4.48	2.53	**2.37**
	S3	3.41	2.48	2.50	2.50	2.75	**2.42**	**2.37**
	S4	3.05	2.50	2.51	2.46	2.95	**2.42**	**2.32**

ImageNet

To evaluate the performance of the VS-DARTS architecture on ImageNet, we transfer the architecture that yields the lowest test error on CIFAR-10 to ImageNet. As shown in Table 3.2, the performance of the searched architecture achieves is comparable to that of the state-of-the-art architecture. We highlight that VS-DARTS achieves 1.9%p improvement from DARTS, the baseline for our method, and 0.7%p improvement from DARTS+PT [68]. When directly searching an architecture on ImageNet using proxy dataset [47], VS-DARTS yields a test error of 24.7%, which is comparable to the recently searched architectures.

3.5.3 Results in RobustDARTS Search Space

We further evaluate the search performance of VS-DARTS on three dataset, namely SVHN, CIFAR-10 and -100, and four search spaces denoted by S1∼S4 [79]. These search spaces are designed to evaluate the robustness of the search algorithm against changes in candidate operations. We followed the experimental protocols of Zela *et al.* [79]. The evaluation process for the CIFAR-10 dataset is identical to the one described in Section 3.5.2. For CIFAR-100 and SVHN datasets, we use the architecture

Table 3.4: Ablation studies on VS-DARTS.

NodeNorm	LARC	$\sqrt{\beta}$	Test Acc.
✗	✗	✗	97.09%
✓	✗	✗	97.29%
✓	✓	✗	97.40%
✓	✓	✓	97.57%

that consists of eight cells, *i.e.*, six normal cells and two reudction cells.

The search results are reported in Table 3.3. VS-DARTS clearly outperforms recent DARTS variants, indicating that in VS-DARTS, the selection rule based on the magnitude of β can function properly to derive the optimal architecture. In particular, on S4, which includes the meaningless noise operation as a candidate operation, the cells searched by VS-DARTS do not contain any noise operation at all, while those of other NAS algorithms [14, 79] do contain some amount of noise operations.

3.5.4 Ablation Study

In this section, we provide a comprehensive ablation study of VS-DARTS to demonstrate the contribution of each added component to the increase in search performance. We also provide experimental results about orthogonality of our VS-DARTS. The experimental settings are as same as Section 3.5.2. In Table 3.4, we report the search performance in test accuracy after applying node normalization, LARC, and $\sqrt{\beta}$. Just by adding node normalization, the test accuracy is improved by 0.2%p (97.09% $\rightarrow$ 97.29%). However, because the problem of imbalanced gradient remains unaddressed, the search performance only with adding node normalization has room for improvement. Applying LARC and introducing $\sqrt{\beta}$ each brings upon 0.31%p (97.09% $\rightarrow$ 97.40%) and 0.48%p (97.09% $\rightarrow$ 97.57%) improvement in test accuracy. Our ablation results support that all of the applied techniques are crucial for substantially improving the search performance.

Table 3.5: Performance on DrNAS [11] w/ and w/o our VS-DARTS on CIFAR-10.

Method	Test Acc.
DrNAS	97.54%
VS-DrNAS	97.58%

Also, We show that VS-DARTS can be adapted to combine with prior DARTS variants. Table 3.5 shows the performance of DrNAS [11] with and without our VS-DARTS. Both experiments are executed on CIFAR-10 dataset with DARTS search space. As shown in Table 3.5, DrNAS with our VS-DARTS surpasses original DrNAS. According to the result, VS-DARTS can be adapted to prior DARTS variants and help to finding the architecture which is more closer to optimal.

3.6 Summary

In this chapter, we revealed that unnormalized node in a continuously-relaxed cell leads DARTS to yield a sub-optimal neural architecture because the architecture parameters do not accurately represent operation strengths. Node normalization was proposed as a simple yet effective solution to address this issue. After applying node normalization, we found that the gradient imbalance problem becomes prominent, and thus, to remedy this problem, the local adaptive learning rate strategy was utilized. Lastly, to further stabilize training of the super-net, we newly introduced $\sqrt{\beta}$-continuous relaxation, which makes the scales of the input and the output feature maps to be similar. We provided through theoretical analysis and empirical results to support the effectiveness of each component. By combining all the components, VS-DARTS successfully searched for a competitive architecture on CIFAR-10. This work alerts that when constructing a search space, the influence of normalization or

lack there of must be carefully considered.

Chapter 4

Platform-aware Neural Architecture Search with Graph Variational Auto Encoder

4.1 Introduction

Hardware resources and dataset sizes are getting larger. In the last ten years, advancements in hardware and exponential data growth have facilitated remarkable improvements in deep neural networks (DNNs) [2]. Thus, neural model compression and its acceleration have become essential issues in on-device deployment. Because large DNNs have exhibited excellent performances in various research fields, deploying neural models to edge devices has become issue. However, designing a compact model is vital to reduce the suffering of model deployment, enabling the application of high-performance deep neural models in resource-constraint environments. Still, this trade-off makes it challenging to find a Pareto frontier between accuracy and computational cost. Owing to these difficulties, researchers are investigating

hardware-aware neural architecture search (hardware-aware NAS), which automatically finds models with small FLOPs without compromising performance [4]. It is crucial to investigate the relationship between models' architectural properties and performance metrics to find a model suitable for each hardware.

Neural architecture search (NAS) continues to attract significant attention from the machine learning society. The representation of architectures has become essential for enhancing NAS performance. Designing an appropriate neural architecture requires automation of the neural architecture search process. To examine how neural architectural properties and their performances are interconnected, NAS researchers are treating neural architecture as a graph and utilizing graph neural networks-based (GNNs-based) variational auto-encoders (VAEs) that represent graphs in continuous latent space to analyze the properties of the graph [44, 6, 82, 42].

However, we observe that the conventional graph generative NAS methods do not work appropriately on cell-based search spaces . Through theoretical analysis, we found that in conventional methods, node generation is performed using only the sub-graph information instead of the full graph information when encoding and decoding a graph with a branch structure. This issue is caused by the short expressive power of GNNs.

In this chapter, we propose a method that uses the entire graph information instead, thereby increasing the explosive power of graphs suitable for directed acyclic graphs (DAGs). In addition, we propose a predictor-based multi-objective NAS method that can derive several models on the Pareto frontier between inference latency and model accuracy on various hardware platforms instead of creating a single model such as a conventional predictor-based NAS. Our experimental results show that node index embedding improves the reconstruction accuracy of the D-VAE-EMB [6] by 86.99% (13.01% to 100%) for NAS-Bench-201 [19]. Moreover, we show that a

multi-objective search using our proposed method can output results very close to Pareto-optimal on various hardware platforms, thereby outperforming conventional GCN- and graph VAE-based methods. Summarily:

- We reveal that conventional DAG VAE usually fails to reconstruct on a cell-based search space, a problem caused by insufficient graph expressive power of DAG VAE.
- We propose a novel decode strategy for DAG VAE, a simple yet effective strategy to increase the performance of the graph VAE on a cell-based search space because our proposed method can remedy the shortage of graph expressive power in DAG VAE.
- We propose a novel predictor-based multi-objective NAS with a graph VAE, which can determine a set of architectures close to Pareto-optimal in various hardware environments.

4.2 Proposed Methods on Graph Variational Auto-Encoder

4.2.1 Fail Case Study

We observed that the existing asynchronous MP-based graph VAE methods which were originally evaluated on ENAS [53], showed limited reconstruction performances on cell-based NAS Benchmarks [74, 19]. We found out that this performance discrepancy occurred in the decoding stage. In decoding **step 1**, the current graph state h_g is usually same as the hidden representation of the previous node h_k [82, 6]. This type of method may cause a reconstruction failure on graphs with tree or loops. The graph described in Fig. 4.2(b) is one example of the reconstruction failure case. In this case, v_{k+1} has two predecessor nodes v_{k1} and v_{k2} with the identical node property

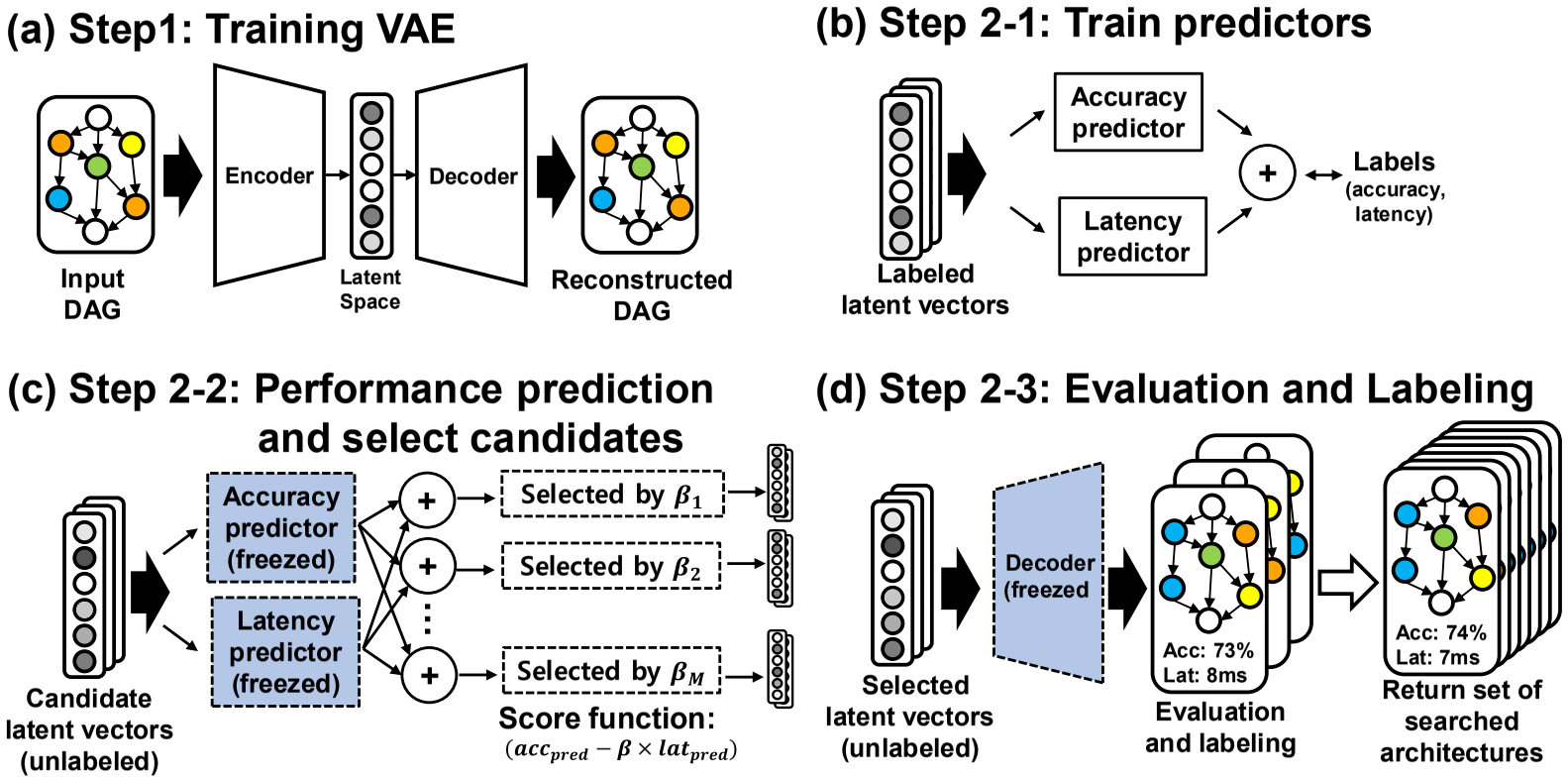

Figure 4.1: Overview of the predictor-based multi-objective NAS with graph VAE.

and sharing the predecessor v_{k-1}. Then, decoder cannot distinguish Fig. 4.2(b) from Fig. 4.2(c), which eventually leads to the reconstruction failure of the decoder. In other words, any pair of two nodes with the same node type and sharing all predecessors will have the same hidden representations is *not distinguishable* in the decoding step.

This problem occurrence is rarely expected in ENAS dataset, which has sequential graph data. Because any two node cannot have the same node type and predecessors in this type of graphs by design. It means that all nodes of a graph have unique topological order in ENAS dataset. Accordingly, the MP strategy in the decoder should be *more expressive* to overcome the reconstruction failure.

4.2.2 Proposed Update Function

The update function $\mathcal{U}$ in Eq. 2.7 should be modified to differentiate between the hidden representations of any two nodes by adding additional information. Therefore, we propose a simple yet effective solution: **a node index** as an additional input to the update function because node indices are always uniquely defined by nodes regardless of the given graph structure. The update function in the encoder should not be modified. If a node index is also given to the update function in the encoder, then the two isomorphic DAGs would have different latent representations with respect to arbitrary node orderings. For this reason, we modified the update function in the **decoder** only to receive the node index r in the graph described in Eq. 4.1.

$$h_r = \mathcal{U}(\mathcal{O}(x_r),\ h_r^{in}, r), \tag{4.1}$$

Previous studies have also used a node index for the MP. More specifically, they concatenated the node index with the hidden states of the nodes [82, 6]. However,

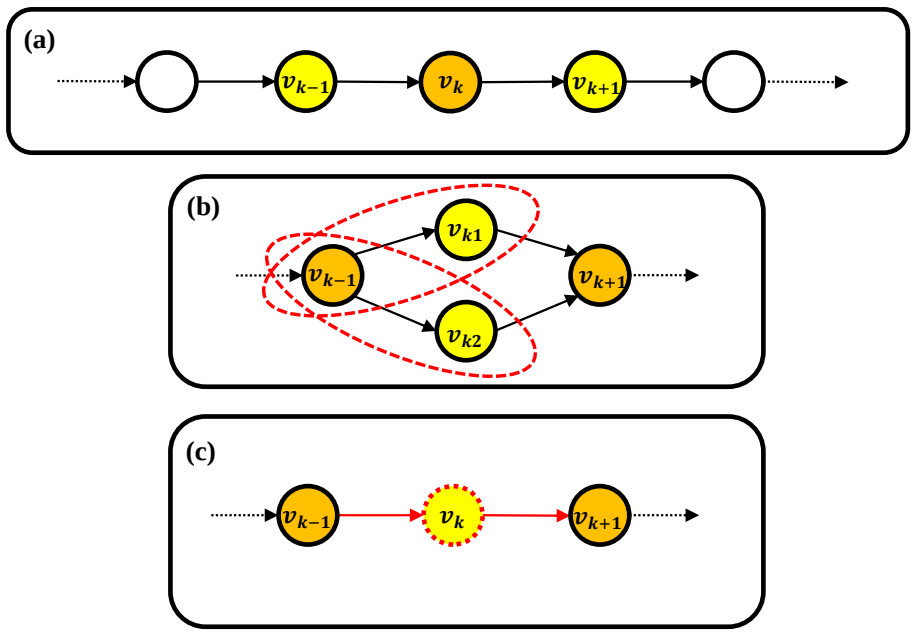

Figure 4.2: **The simplified architecture topologies and an example of a fail case.** We describe two types of simplified NAS architecture types. Note that a node represent an operation, such as convolution or max-pooling. **(a)** ENAS architecture type. In this type of architecture, nodes are sequentially connected along a simple path, without a branch or a cycle. **(b)** NAS-Bench-101 and 201 architecture type. **(c)** An example of reconstruction error from type (b) architecture. Contrary to (a), message propagation paths in (b) are more complicated and have an acyclic loop or branches. We examined whether the architectures of the (b) type are vulnerable to graph isomorphism tests. Both messages from v_{k1} and v_{k2} may not be distinguished because both nodes have the same type and share the same neighbor. We represent these pictures based on the early paper [52].

their results showed slight improvement. In this study, we assumed that the node index information in the encoder of the D-VAE and D-VAE-EMB is not useful for graph reconstruction tasks. More specifically, our method uses a node index only in the decoding step, and any two isomorphic DAGs can always be mapped to the same latent space.

4.2.3 Observer Node

As mentioned in Section 2.2.4, in previous DAG VAE methods, the graph state hg is usually the same as the hidden state of the last generated node, where the D-VAE typically generates nodes with the hidden state of an unconnected node. A node generated from the hidden state of another node without dependencies may cause inefficiency. However, our decoding strategy uses an observer node, which can always reflect all graph properties.

4.3 Proposed Predictor-based Multi-objective NAS

In this section, we describe the detailed process of our multi-objective NAS with graph VAE. Overview of the search process is presented at Fig. 4.1.

4.3.1 Training Graph VAE (Step 1)

In Step 1, D-VAE-TMB trains the graph reconstruction tasks. We train both auto-encoder and predictors simultaneously, because we wanted to latent space of auto-encoder be trained based on the accuracy and latency of predictors, as well as reconstruction of decoders. In other words, we made latent spaces be trained for heterogeneous tasks. Our improved performances are showed in Section 4.4.2.

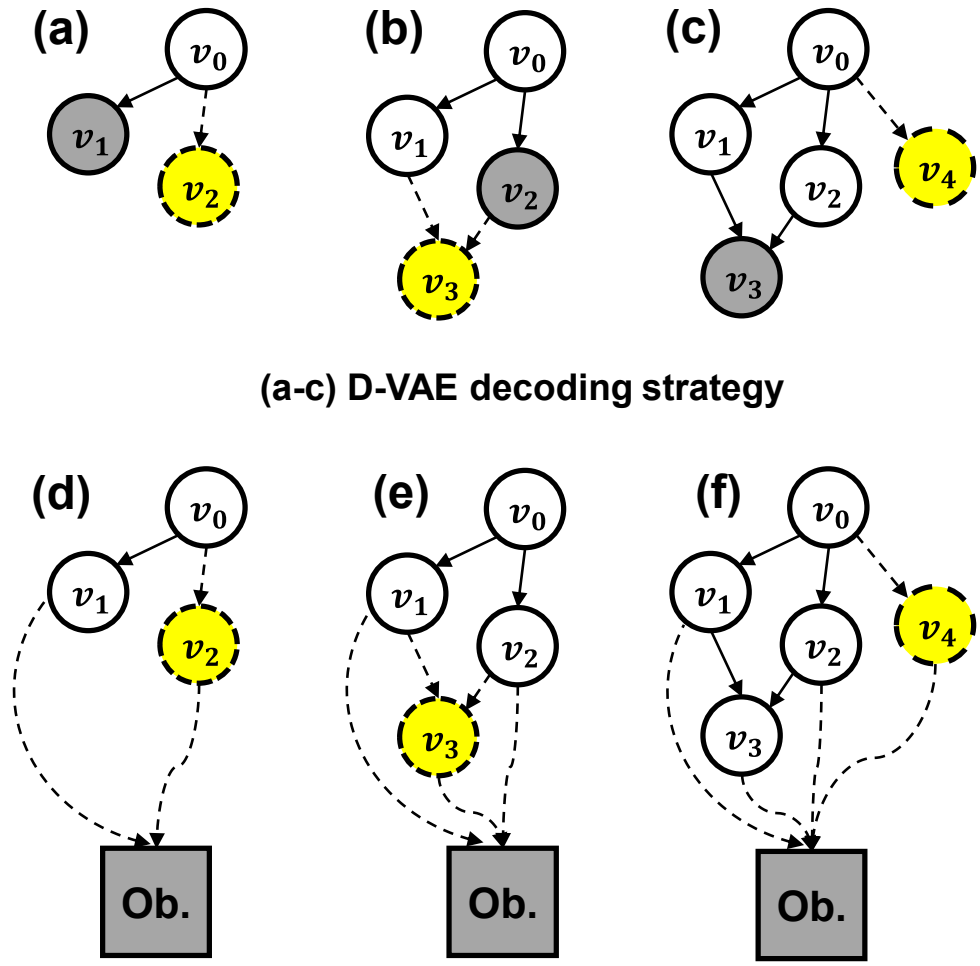

Figure 4.3: Decoding strategy comparison between D-VAE and our proposed method. Yellow node and dashed edges mean a node and edges currently being generated. Graph state h_g is obtained from the hidden state of the gray-colored node. **(a-c)** Decoding strategy of D-VAE [82]. **(d-f)** Our proposed decoding strategy.

Algorithm 1 Predictor-based Multi-objective NAS

Input: X(graph or latent vector), maximum iteration I, sampling number per iteration K, initial sample number N_{init}. **Output**: Searched architectures

$X_{train} \leftarrow$ RandomSample(X, N_{init})
Eval(X_{train}) % Measure the performances (acc. and lat.)
$X_{candidate} \leftarrow X \setminus X_{train}$
$i \leftarrow 0$
while $i \leq I$ **do**
    Initialize and Train $Pred_{acc}$ and $Pred_{lat}$ with X_{train}
    $Scores_{acc} \leftarrow Pred_{acc}(X_{candidate})$ / $\sigma(y_{train})$
    $Scores_{lat} \leftarrow Pred_{lat}(X_{candidate})$ / $\sigma(l_{train})$
    X_{pick} = { }
    $k = K/n$
    for β **in [**β_1**, ...,** β_n**]**
        $ScoreSum \leftarrow -Scores_{acc}$ + β * $Scores_{lat}$
        Sort $\mathbf{x_1}, ..., \mathbf{x_n}$ by $ScoreSum$
        $X_{new} \leftarrow$ Top-k $\mathbf{x_1}, ..., \mathbf{x_k}$ from $X_{candidate}$
        $X_{pick} \leftarrow X_{pick} \cup X_{new}$
    Eval(X_{pick})
    $X_{train} \leftarrow X_{train} \cup X_{pick}$
    $X_{candidate} \leftarrow X_{candidate} \setminus X_{pick}$
    $i \leftarrow i + 1$
end while
$X_{searched} \leftarrow X_{train}$
Return $X_{searched}$

4.3.2 Search Process (Step 2)

For hardware-aware NAS using VAE, we conducted a predictor-based multi-objective search as Algorithm 1. The search process is largely composed of three steps, and details are as follows.

Step 2-1: Training the accuracy and latency predictor (line 1-6)

After training graph VAE, initial train set X_{train} is composed with randomly selected graphs from X. The other graphs are set to the $X_{candidate}$. Then, we encode with the latent space using the frozen encoder and put it in the latent vector training set X_{train}. Then, each samples of the X_{train} is evaluated to label both accuracy and latency.

We trained both predictors with the same input Z_{train} and two different labels of accuracies and latencies for each predictor, respectively. Basically, the predictors are regressors which take latent vectors as input. Inspired by the universal approximation theorem[27], we build a multi-layer perceptron (MLP) model which have only one hidden layer and a non-linearity for each predictor.

Step 2-2: Selection of the architectures for evaluation. (line 7-13)

Inspired by multi-objective Bayesian optimization techniques [60, 3], we do not aim to search just a single architecture, but to find a Pareto-frontier in a given search space and hardware environment. To do so, we calculate $ScoreSum$ by adding the two scores with list of β. (We use [0.01, 0.02, ..., 0.25] as the list of β.) The search cost of searching Pareto-frontier is similar to the case of finding a single architecture at once. This soft constraint method is more powerful than hard constraint when the number of X_{train} is not sufficient at the early stage of search process, because predictors are not well trained yet.

Step 2-3: Evaluate the picked architectures and add to the train set (line 14-18)

After selection of the next X_{pick}, we evaluate the architectures to get real accuracy and latency, e.g. training each of architectures from scratch or request metrics on benchmark dataset. After evaluation, we label the accuracy and the latency on X_{pick}, then put them on X_{train} for next iteration. We initialize parameter of both accuracy and latency predictor at the start of iteration.

4.3.3 Return the set of the searched architectures (line 21-22)

After the iteration terminated in line 5-20, a total of $I \times k$ architectures and their performances are obtained. If we draw a Pareto-frontier for $X_{searched}$, we can obtain an architecture that satisfies several latency constraints as in the result of Section 4.4.4.

4.4 Experimental Results

4.4.1 Settings

To evaluate our proposed methods, we adapt proposed methods to D-VAE [82] with operation embeddings [6] which shows state-of-the-art VAE performance on ENAS [53] then compare with D-VAE [82], D-VAE-EMB [6], arch2vec [73] and SVGe [42].

ENAS macro search space [53] is an entire network design space. In this space, 8 nodes are sequentially connected. In other words, there is always directed edge between two adjacent indexed nodes v_{k-1} and v_k. Also, the existence of skip connection between two non-adjacent indexed nodes is a component of ENAS macro search space. Each node can be 6 types of operation except input and output nodes. Following previous works[82, 6], 19,020 graphs are sampled from ENAS search space.

NAS-Bench-101 [74] is the first published benchmark for NAS research to alleviate burden of large computation. This search space is a kind of cell-based search

space which design only small module called cell and connect them together to form a network. A cell consists of 7 or less nodes and 9 edges. The number of candidate operations is three: $\{3 \times 3\ max-pool, 1 \times 1\ convolution, 3 \times 3\ convolution\}$. Accordingly, NAS-Bench-101 consists of 423k unique architectures and their CIFAR-10 accuracy.

NAS-Bench-201 [19] is a kind of cell-based search space like NAS-Bench-101. Dong *et al.*[19] denoted operations as edge properties and intermediate featuremap as a node. For compatibility, we converted edges to nodes and nodes to edges. A cell consists of 6 nodes and fixed connections. Candidate operations are $\{zero, skip-connect, 1 \times 1\ convolution, 3 \times 3\ convolution., 3 \times 3\ avgpool\}$. NAS-Bench-201 consists of 15,625 architectures and their accuracy for three image classification dataset, namely CIFAR-10, CIFAR-100 [32], and tiny version of ImageNet (ImageNet-16-120) [33].

4.4.2 VAE Performance Comparison

In this experiment, we compared the auto-encoder reconstruction performances of the proposed method with those of others on three different NAS search spaces. For a fair comparison, we used metrics following [82, 6]: (1) **Accuracy** to show proportion of perfectly reconstructed DAGs on the test set. (2) **Validity** show the proportion of valid DAGs, which generate graphs belonging to their entire search space (we decoded ten randomly sampled 1,000 latent representations z to g followed by [82, 6]. In total, 10,000 graphs were generated. In other words, N_{decode} was 10,000.), (3) **Uniqueness** indicates to show the proportion of unique to valid DAGs. **# of Unique DAGs** is the number of unique DAGs in the generated graphs, which can be derived as $Validity/100 \times Uniqueness/100 \times N_{decode}$.

Our proposed node index embedding shows a powerful performance in the cell-

Table 4.1: Comparison of VAE performances on ENAS [52] and NAS-Bench-201 [19].We evaluated three types of generation performances, accuracy, validity, and uniqueness. Our model showed the best performances in accuracy and validity, in all datasets. Compared to our baseline model D-VAE-EMB, our proposed method with additional indices accomplished remarkable improvement in the graph reconstruction task. The best results in each setting remarked as bold.

Dataset	Method	Accuracy	Validity	Uniqueness	# Unique
ENAS	D-VAE [82]	99.96	**100**	37.26	3726
	D-VAE-EMB [6]	99.99	**100**	39.15	**3915**
	SVGe [42]‡	99.63	**100**	39.03	3903
	D-VAE+ours	**100**	**100**	26.82	2682
NB101	arch2vec† [73]	**100**	43.70	10.00	437
	D-VAE [82]‡	25.89	82.55	19.84	1638
	D-VAE-EMB [6]	60.88	**98.31**	28.05	**2756**
	SVGe [42]‡	**100**	79.16	32.10	2541
	D-VAE+ours	**100**	95.61	28.38	2713
NB201	arch2vec† [73]	**100**	95.93	7.33	703
	D-VAE[82]	13.11	1.34	64.93	87
	D-VAE-EMB [6]	13.01	1.18	61.86	73
	SVGe [42]‡	99.99	**100**	8.28	828
	D-VAE+ours	**100**	**100**	11.28	1128
	D-VAE+ours+pre-predictor	**100**	**100**	16.17	**1617**

† The inputs are 8x7 operation and 8x8 adjacency-edge matrix, the others' inputs are directed acyclic graphs (DAGs).
‡ We got the results from [42]. The other results are tested by ourselves with each of published code.

based search space including NAS-Bench-101 and NAS-Bench-201 as well as the ENAS search space. The comparison results are presented in Table 4.1. In the NAS-Bench-201 dataset, our proposed method showed impressive results in accuracy and validity against the baseline model D-VAE-EMB [6]. For the ENAS dataset, the proposed method shows the highest accuracy. The uniqueness drops slightly against the baseline model D-VAE–EMB [6] but not significantly. In the case of the NAS-Bench-101, only our algorithm shows high accuracy and validity. It can be inferred that SVGe [42] performs well even though it is based on the D-VAE because the operation smoothness of D-VAE is insufficient. However, its performance in D-VAE-EMB [6], to which operation embedding is applied, is suboptimal. Therefore, op-

erational smoothness is not the root cause of the poor performance of D-VAE and D-VAE-EMB in cell-based search spaces.

4.4.3 Pre-predictor

To analyze how the latent space changes according to the application of the predictor, we visualized it using T-SNE. When a predictor is applied, there is an effect of mapping architectures with similar latency to a similar latent space. Since our purpose is to perform NAS using an autoencoder, a pre-predictor is attached to the autoencoder so that the performance of neural DAGs can be melted into the latent space.

As shown in Figure 4.4, there is tendency according to the latency in the latent space compared to when the pre-predictor is not used.

4.4.4 Search Performance Comparison

Fig. 4.5 shows the multi-objective search results for LatBench [20], which is the latency benchmark constructed on NAS-Bench-201 [19]. We present searched models close to Pareto-optimal when the total number of searched architectures is 150, 350, and 550. The performances of 200 architectures were randomly sampled and used as the pre-predictors mentioned in Section 4.4.3). More specifically, we trained the proposed VAE with 13862 unlabeled and 200 labeled architectures, while 1563 were used for the test set. It can be seen that the Pareto-frontier of our D-VAE- TMB (w/. predictor) is better than BRP-NAS [20] and close to the real Pareto-optimal except for Pixel3. We used the GCN-based predictor proposed in BRP-NAS [34] for both latency and accuracy prediction, unlike in previous studies, where we believe that the binary search of BRP-NAS did not fit well with the multi-objective search.

Table 4.2 shows the search performance of BRP-NAS and our algorithm on the three hardware platforms based on the latency constraint. For the desktop CPU and

GPU, 3, 5, 7ms, and for the embedded GPU, 14, 19, 25ms were set as the constraints, respectively. In most cases, our proposed methods with pre-predictor shows better performance than BRP-NAS. The performance of the proposed methods differs significantly depending on whether the pre-predictor is used. The number of latency examples used in the pre-predictor is not large at 200, and it also measures inference latency, considering that it does not take a long time to obtain. Compared to this BRP-NAS, it can be said that it has a great strength.

4.5 Discussions

4.5.1 Node Index Order in DAG VAE

In contrast to previous reports, we found that the topological order of node indices in the search space is critical to improving the reconstruction performance because the hidden state of the last node is used as a graph state in our proposed method; therefore, it is easier to generate nodes in the decoding step. We found that using the observed node state can constantly improve the performance, regardless of the node indexing strategy. Empirically, the depth-first-order strategy significantly outperforms the breast-first-order search.

4.5.2 Model size reduction while keep the reconstructive performance

For a fair evaluation, we set a model complexity similar to that in other existing works, such as D-VAE. Our proposed methods showed robust performance even when the latent dimension was reduced to 128 from 501. Our model converged faster than previous methods. In addition, it can reduce the sparsity of latent representation; therefore, it will eventually help improve architecture search. An additional evaluation of the relationship between model complexity and other datasets may need to be

conducted.

4.5.3 Convergence Acceleration

In addition, we compare the convergence speeds of the D-VAE and our proposed method. As shown in Fig. 4.6, our proposed method has a much faster convergence speed than D-VAE for the training loss and KL-divergence. Speculatively, our proposed decoding strategy and observer node reduce the risk of generating an isomorphic graph and increase the learning rate. In D-VAE and its variants, the encoder attempts to map the isomorphic graph to the same hidden space. In unsupervised and semi-supervised learning, when graph g is an input, it is difficult to target all isomorphic graphs of graph g. The proposed method seems to make the learning speed extremely fast by giving direction to any of the isomorphic graphs in the decoding step.

4.6 Summary

In this chapter, we proposed platform-aware neural architecture optimization using a variational graph auto-encoder with decoder which use node index. node index is entered into the message-passing function to improve the primary VAE performance in a non-sequential search space. Then, the continuous latent spaces of the candidate neural architectures are well trained, and we attempt to search Pareto-optimal in each hard-constrained latency.

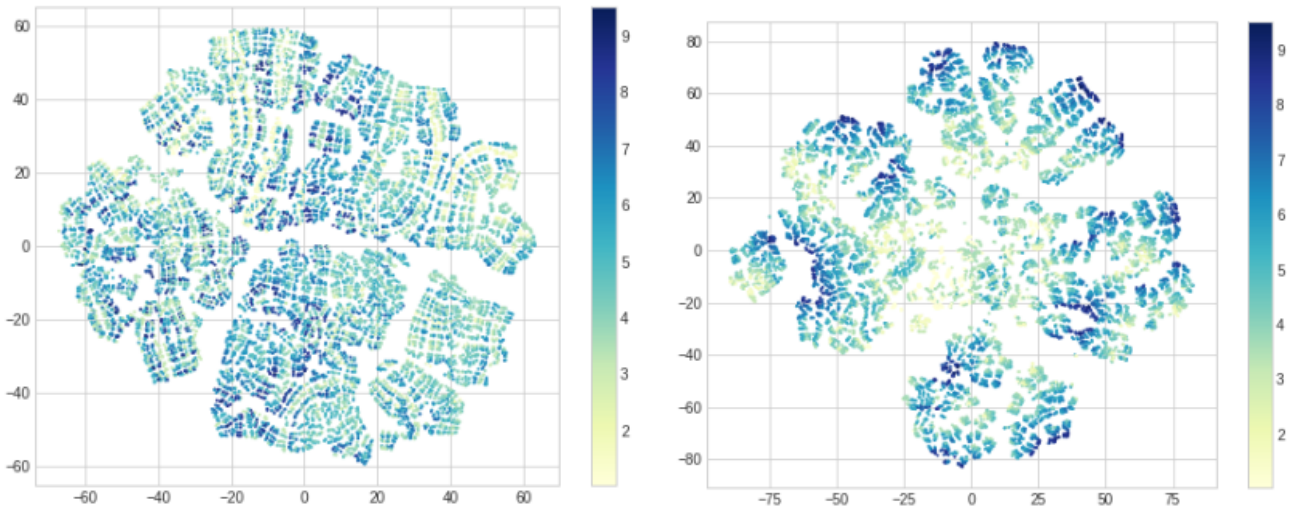

Figure 4.4: Latent space comparison between auto-encoder with and without pre-predictor. Left: D-VAE+ours without pre-predictor, right: D-VAE+ours with pre-predictor. T-SNE [63] is used to reduce dimension and visualize the latent spaces.

Table 4.2: Comparison of search results on NAS-Bench-201 [19] and LatBench [20]. All of the results is obtained after 350 architectures evaluation.

Hardware	Const.	Method	Latency	Accuracy
Desktop CPU	3ms	BRP-NAS[20]	1.4ms	63.89%
		Ours(w/o pred.)	1.7ms	64.09%
		Ours(w/. pred.)	2.7ms	67.55%
	5ms	BRP-NAS[20]	4.1ms	70.79%
		Ours(w/o pred.)	4.9ms	71.84%
		Ours(w/. pred.)	4.6ms	72.02%
	7ms	BRP-NAS[20]	6.4ms	72.98%
		Ours(w/o pred.)	6.8ms	72.95%
		Ours(w/. pred.)	6.1ms	73.09%
Desktop GPU	3ms	BRP-NAS[20]	1.8ms	64.81%
		Ours(w/o pred.)	1.1ms	58.56%
		Ours(w/. pred.)	2.5ms	67.55%
	5ms	BRP-NAS[20]	4.6ms	72.02%
		Ours(w/o pred.)	4.6ms	72.02%
		Ours(w/. pred.)	4.6ms	72.02%
	7ms	BRP-NAS[20]	6.4ms	72.98%
		Ours(w/o pred.)	6.2ms	73.02%
		Ours(w/. pred.)	6.1ms	73.09%
Embedded GPU	14ms	BRP-NAS[20]	13.3ms	70.24%
		Ours(w/o pred.)	13.7ms	70.43%
		Ours(w/. pred.)	13.4ms	70.69%
	19ms	BRP-NAS[20]	17.9ms	71.72%
		Ours(w/o pred.)	18.7ms	71.11%
		Ours(w/. pred.)	18.2ms	71.56%
	25ms	BRP-NAS[20]	24.7ms	72.64%
		Ours(w/o pred.)	24.2ms	73.02%
		Ours(w/. pred.)	23.6ms	73.09%

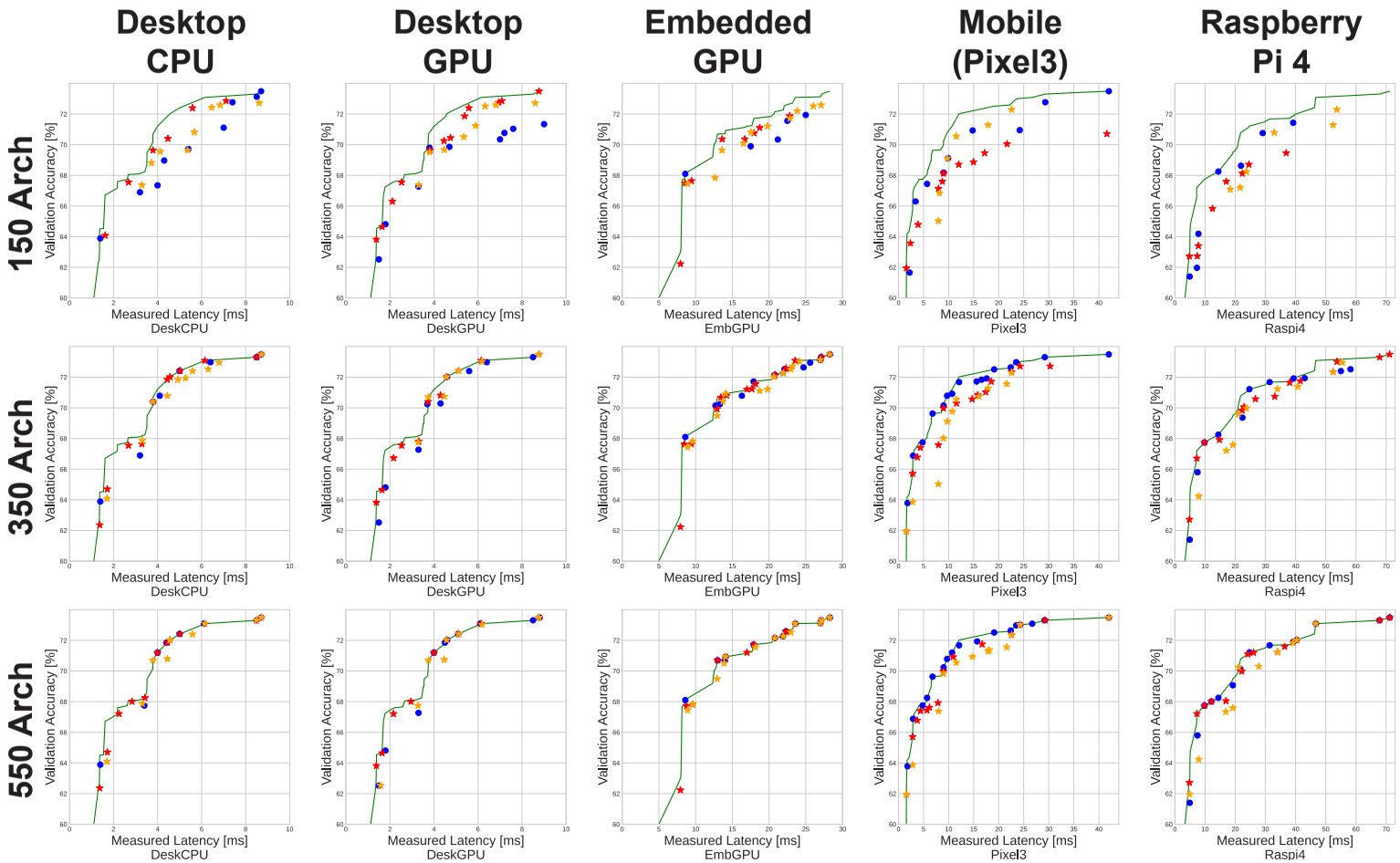

Figure 4.5: The NAS-Bench-201 [19] search results of five different hardware environments on accuracy vs. measured latency [20, 35]. Red star, orange star, and blue circle represents Pareto-optimal models searched with trained D-VAE+ours (w/. pre-predictor), D-VAE+ours (w/o pre-predictor) and GCN-based predictor proposed by [20], respectively. GCN-based accuracy predictor and GCN-based latency predictor proposed by BRP-NAS were used. Note that the accuracy predictor is not binary. The reason is that the binary predictor of BRP-NAS may not be unsuitable for multi-objective search. The algorithms were tested on the search method we proposed. Green line represent ground-truth Pareto-frontier. DeskCPU: Desktop CPU Intel core-i7 7820x, DeskGPU: Desktop GPU NVIDIA GTX 1080ti, EmbGPU: Embedded GPU Jetson nano, Pixel3: Google smartphone Pixel3, Raspi4: Raspberry Pi 4.

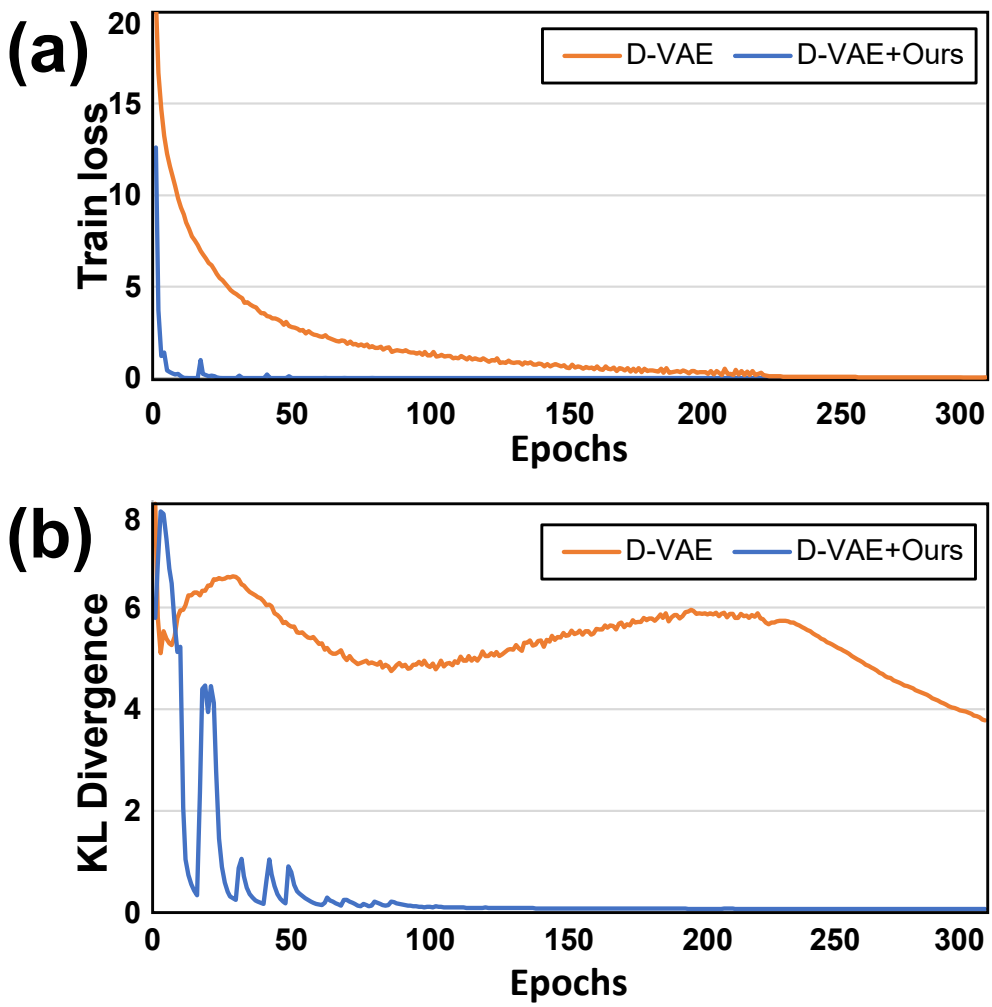

Figure 4.6: (a) Train loss convergence and (b) KL-divergence convergence speed comparison between D-VAE and D-VAE+our proposed method described in Section 5.2.2 and 5.2.3. Both (a) and (b) were performed on ENAS macro search space.

Chapter 5

Conclusion

Low-latency neural model design has been more and more important issue to deployments. As large size deep neural model shows better performances in various research field, it is hard to deploy neural model to edge devices. To use high performance deep neural models in resource-constraint environments, design compact model and its acceleration is absolutely necessary. In this regard, we have introduced two algorithm side methods to design compact neural model in this dissertation, including neural architecture search (NAS), and neural architecture optimization using graph variational auto-encoder.

In Chapter 3, we revealed that unnormalized node in a continuously-relaxed cell leads DARTS to yield a sub-optimal neural architecture because the architecture parameters do not accurately represent operation strengths. Node normalization was proposed as a simple yet effective solution to address this issue. After applying node normalization, we found that the gradient imbalance problem becomes prominent, and thus, to remedy this problem, the local adaptive learning rate strategy was utilized. Lastly, to further stabilize training of the super-net, we newly introduced $\sqrt{\beta}$-continuous relaxation, which makes the scales of the input and the output feature

maps to be similar. We provided through theoretical analysis and empirical results to support the effectiveness of each component. By combining all the components, VS-DARTS successfully searched for a competitive architecture on CIFAR-10. Last but not least, our VS-DARTS with soft constrained on MAC shows competitive performance to cell-based NAS method on ImageNet mobile setting.

In Chapter 4, we proposed platform-aware neural architecture optimization using a variational graph auto-encoder with decoder which use node index. node index is entered into the message-passing function to improve the primary VAE performance in a non-sequential search space. Then, the continuous latent spaces of the candidate neural architectures are well trained, and we attempt to search Pareto-optimal in each hard-constrained latency.

In this dissertation, we have covered comprehensive topics for compact neural network design methodology to deploy resource-constraint environments, including hardware-aware neural architecture search using differentiable NAS and graph VAE.

Bibliography

[1] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[2] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1): 1–74, 2021.

[3] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

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

[6] Michail Chatzianastasis, George Dasoulas, Georgios Siolas, and Michalis Vazirgiannis. Graph-based neural architecture search with operation embeddings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 393–402, 2021.

[7] Baotong Chen, Jiafu Wan, Lei Shu, Peng Li, Mithun Mukherjee, and Boxing Yin. Smart factory of industry 4.0: Key technologies, application case, and challenges. *Ieee Access*, 6:6505–6519, 2017.

[8] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2020.

[9] X. Chen, L. Xie, J. Wu, and Q. Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1294–1303, 2019. doi: 10.1109/ICCV.2019.00138.

[10] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. 2020.

[11] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2020.

[12] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020.

[13] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. In *16th Eu-*

ropoean Conference On Computer Vision, 2020. URL https://arxiv.org/abs/1911.12126.pdf.

[14] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: Robustly stepping out of performance collapse without indicators. 2021.

[15] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[17] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[18] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.

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

[20] Łukasz Dudziak, Thomas Chau, Mohamed S Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D Lane. Brp-nas: Prediction-based nas using gcns. *arXiv preprint arXiv:2007.08668*, 2020.

[21] Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020.

[22] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[23] Floris Geerts, Filip Mazowiecki, and Guillermo Perez. Let's agree to degree: Comparing graph convolutional networks in the message-passing framework. In *International Conference on Machine Learning*, pages 3640–3649. PMLR, 2021.

[24] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[25] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[26] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[28] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets:

Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[31] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[32] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[34] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[35] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2020.

[36] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.

[37] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[38] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

[39] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[40] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[41] Andreas Loukas. How hard is to distinguish graphs with graph neural networks? In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3465–3476. Curran Associates, Inc., 2020.

[42] Jovita Lukasik, David Friede, Arber Zela, Frank Hutter, and Margret Keuper. Smooth variational graph embeddings for efficient neural architecture search. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[43] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[44] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural archi-

tecture optimization. *Advances in neural information processing systems*, 31, 2018.

[45] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019.

[46] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[47] Byunggook Na, Jisoo Mok, Hyeokjun Choe, and Sungroh Yoon. Accelerating neural architecture search via proxy data. In *International Joint Conference on Artificial Intelligence*, 2021.

[48] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. *arXiv preprint arXiv:1509.06569*, 2015.

[49] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

[50] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2020.

[51] Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[52] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

[53] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4095–4104, 2018.

[54] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911, 2017.

[55] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[56] Russell Reed. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5):740–747, 1993.

[57] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[58] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[59] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

[60] Shinya Suzuki, Shion Takeno, Tomoyuki Tamura, Kazuki Shitara, and Masayuki Karasuyama. Multi-objective Bayesian optimization using pareto-frontier entropy. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9279–9288. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/suzuki20a.html`.

[61] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[62] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[63] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[64] D Vellante. A new era of innovation: Moore's law is not dead and ai is ready to explode. *SiliconANGLE URL: https://siliconangle. com/2021/04/10/newera-innovation-moores-law-not-dead-ai-ready-explode*, 2021.

[65] Clement Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. *arXiv preprint arXiv:2006.15107*, 2020.

[66] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Efty-

chios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[67] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020.

[68] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. In *International Conference on Learning Representations*, 2021.

[69] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[70] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

[71] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[72] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

[73] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsuper-

vised architecture representation learning help neural architecture search? *arXiv preprint arXiv:2006.06936*, 2020.

[74] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[75] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *International Conference on Machine Learning*, pages 7134–7143. PMLR, 2019.

[76] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[77] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020.

[78] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Auto graph encoder-decoder for neural network pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6362–6372, 2021.

[79] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.

[80] Fuyang Zhang, Nelson Nauata, and Yasutaka Furukawa. Conv-mpn: Convolutional message passing neural network for structured outdoor architecture re-

construction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2798–2807, 2020.

[81] Li Zhang, Dan Xu, Anurag Arnab, and Philip HS Torr. Dynamic graph message passing networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3726–3735, 2020.

[82] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *arXiv preprint arXiv:1904.11088*, 2019.

[83] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[84] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[85] Ruizhe Zhao, Wayne Luk, Xinyu Niu, Huifeng Shi, and Haitao Wang. Hardware acceleration for machine learning. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 645–650. IEEE, 2017.

[86] Pan Zhou, Caiming Xiong, Richard Socher, and Steven Chu Hong Hoi. Theory-inspired path-regularized differential network architecture search. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8296–8307, 2020.

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

[88] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

초 록

딥러닝은 빅데이터와 강력한 병렬 프로세서가 사용 가능해지면서 다양한 분야에서 성공적인 모습을 보여주고 있으며, 모바일과 임베디드 시스템 같은 엣지 디바이스에도 다양하게 적용되고 있다. 그러나, 엣지 디바이스는 일반적으로 컴퓨팅 및 전력 자원이 부족한 환경이다. 이를 해결하기 위해서, 저-레이턴시 모델 디자인, 모델 압축 기법과 뉴럴 모델 가속 등이 알고리즘과 하드웨어 양쪽 측면에서 활발하게 연구되고 있다. 본 논문에서는 알고리즘 측면에서 저-레이턴시 모델 디자인을 위한 두 가지 새로운 방법론을 제시한다. 필요한 컴퓨팅 자원을 줄이기 위해서는 먼저 컴팩트하고 레이턴시가 작은 모델을 디자인하는 것이 중요하므로, 알고리즘 측면에서 우리는 두가지 신경구조탐색을 이용한 모델 디자인 방법론을 제시하였다: 셀 기반 신경구조탐색과 그래프 배리에이셔널 오토인코더를 이용한 신경구조탐색이다. 본 학위논문의 셀 기반 신경구조탐색은 널리 알려진 differentiable NAS 방법론인 DARTS (Differentiable ARchiTecture Search)에 기반한다. DARTS는 여러 연구에서 베이스라인 방법론으로 널리 사용되고 있음에도 불구하고, 종종 학습 불안정성과 최적화가 부족하다는 점이 이미 보고된 바 있으나 그 근본적인 원인에 대해서는 일부 밝혀지지 않았었다. 우리는 이론적 분석과 관찰을 통해서 그 근본적인 문제가 각 오퍼레이션의 정규화되지 않은 출력에 기인하는 것을 밝히고, 이 문제점을 해결할 수 있는 방법론인 VS-DARTS (variance stationary DARTS)를 제안하였다. VS-DARTS는 구조 변수(architectural parameter)의 신뢰성을 높여서 탐색 비용을 늘이지 않고 성능을 높였다. 또한, 우리는 VS-DARTS에 레이턴시에 대한 연성 제약(soft constraint)을 적용함으로써 기존 셀 기반 방법론에 비견되는 성능의 구조를 탐색하였다. 또 다른 저 레이턴시 모델을 찾는 접근 방법으로는 그래프 생성 모델(graph generative model)을 적용하였다. 우리는 새로운 그래프 배리에이셔널

오토인코더 (graph variational auto-encoder) 방법론을 제안하여 셀 기반 탐색 공간에 대한 오토인코더 성능을 크게 향상시켰다. 이후, 제안한 배리에이셔널 오토인코더를 이용하여 뉴럴 구조의 임베딩 정보를 추출하고, 추출된 구조 임베딩 정보를 새로이 제안한 다중목적탐색(multi-objective search)에 이용하여 레이턴시-정확도에서 파레토 최적(Pareto optimal)에 가까운 구조들을 찾을 수 있음을 보였다. 또한 제안한 신경구조탐색 방법론을 다양한 하드웨어 플랫폼 상에서 검증하였다. 요약하면, 본 학위논문에서는 보다 컴팩트하고 레이턴시가 작은 모델을 찾는데 사용할 수 있는 신경구조탐색 방법론을 두가지 제안하였고, 이를 이용하여 컴퓨팅 자원이 부족한 환경을 위한 인공 신경망의 자동 설계 방법에 관하여 기술하였으며, 다양한 실험을 통해서 검증하였다.

주요어: 딥 러닝, 인공신경망, 에너지 효율, 신경 구조 탐색, 하드웨어-어웨어 나스
학번: 2015-21001