Lightweight Multi-Objective and Many-Objective Problem Formulations for Evolutionary Neural Architecture Search with the Training-Free Performance Metric Synaptic Flow

An Vo, Tan Ngoc Pham, Van Bich Nguyen and Ngoc Hoang Luong
University of Information Technology, Ho Chi Minh City, Vietnam
Vietnam National University, Ho Chi Minh City, Vietnam
E-mail: 19520007@gm.uit.edu.vn, 19520925@gm.uit.edu.vn, vannb@uit.edu.vn, hoangln@uit.edu.vn

Keywords: neural architecture search, evolutionary algorithms, multi-objective optimization

Received: March 11, 2023

Neural architecture search (NAS) with naïve problem formulations and applications of conventional search algorithms often incur prohibitive search costs due to the evaluations of many candidate architectures. For each architecture, its accuracy performance can be properly evaluated after hundreds (or thousands) of computationally expensive training epochs are performed to achieve proper network weights. A so-called zero-cost metric, Synaptic Flow, computed based on random network weight values at initialization, is found to exhibit certain correlations with the neural network test accuracy and can thus be used as an efficient proxy performance metric during the search. Besides, NAS in practice often involves not only optimizing for network accuracy performance but also optimizing for network complexity, such as model size, number of floating point operations, or latency, as well. In this article, we study various NAS problem formulations in which multiple aspects of deep neural networks are treated as multiple optimization objectives. We employ a widely-used multi-objective evolutionary algorithm, i.e., the non-dominated sorting genetic algorithm II (NSGA-II), to approximate the optimal Pareto-optimal fronts for these NAS problem formulations. Experimental results on the NAS benchmark NATS-Bench show the advantages and disadvantages of each formulation.

Povzetek: Uporabljen je algoritem NSGA-II za analizo NAS problemov, tj. za iskanje primerne nevronske arhitekture.

1 Introduction

The goal of Neural Architecture Search (NAS) is to accelerate the design process of high-performing deep neural network architectures by exploring the vast space of possible network configurations and selecting the most promising ones. This process typically involves searching over a large number of potential architectures, evaluating their performance, and iteratively refining the algorithm to converge on the best-performing architectures [12]. However, many state-of-the-art NAS methods require substantial computational resources. For example, Zoph et al. [30] employed 800 GPUs over 28 days to solve NAS using a reinforcement earning algorithm, whereas Real et al. [27] proposed an evolution-based technique (AmoebaNet-A) that took 7 days to execute on 450 K40 GPUs. To reduce such heavy computation costs, current NAS efficiency research proposes the adoption of *training-free performance metrics* [1] as a performance objective rather than network accuracy. These metrics can be computed using network weights at initialization and do not require any training epochs, thus primarily involving network designs. Several such training-free metrics have been shown to be correlated with actual network accuracy to some extent [1]. Hence, optimizing these metrics potentially leads to promising architectures.

While most studies focus on optimizing network architectures for a single objective, such as network accuracy, real-world neural network deployments frequently necessitate the consideration of other important factors, such as FLOPs, number of parameters, and latency. NAS architectures that are optimized just for accuracy may be too cumbersome for resource-constrained embedded devices. Moreover, by solving multi-objective problems, a trade-off front between performance and complexity can be obtained, which provides decision-makers with the necessary information to select an appropriate network. Several research has presented multi-objective NAS (MONAS) formulations that take into consideration important aspects. For example, Lu et al. [20] presented NSGA-Net, which used the non-dominated sorting genetic algorithm II (NSGA-II) [6] to solve an MONAS problem with two conflicting objectives, i.e., the classification error and the number of floating-point operations (FLOPs). In another work [19], NSGA-II was also used to solve a many-objective problem formulation with five optimization objectives, including ImageNet accuracy, number of parameters, number of multiply-add operations, CPU and GPU latency.

Lu et al. [19] also developed a surrogate model to forecast the accuracy of candidate architectures and the predictor was refined during the search process to enhance the performance of NSGA-II in solving MONAS. To build the predictor, a limited number of architectures were sampled from the search space at first. Following that, NSGA-II was used to search for architectures, treating the accuracy predictor as an objective alongside other complexity objectives. Despite the fact that they employed a surrogate model as an objective for NSGA-II to discover architectures, they still trained these architectures and used them as training samples to refine the accuracy predictor. Using complexity metrics and training-free performance metric Synaptic Flow (`synflow`) simultaneously, Phan et al. [25] randomly choose a wide variety of architectures and evaluate their complexity and performance. Non-dominated architectures with high performance and low complexity are then utilized to initialize the population for a bi-objective evolutionary NAS process where network accuracy is used as the primary performance metric. The training-free `synflow` metric is only employed during the *warm-up* phase. During the search phase, candidate architectures still need to be trained and evaluated for their performance. It's also possible to use `synflow` metric to enhance the performance of NSGA-II in solving multi-objective NAS problems as in [26], by developing a training-free multi-objective local search. In each generation, a subset of potential architectures undergoes a local search process that uses `synflow` for improvement checks, eliminating the need for training epochs. In contrast to these works, our approach does not rely on any training process. Instead, we use the training-free performance metric `synflow` to evaluate all candidate architectures during the search. This eliminates the need for training and allows us to search for high-quality architectures more efficiently. Do et al. [7] also propose a completely training-free multi-objective evolutionary NAS framework that employs the number of linear regions $R_{\mathcal{N}}$ and the condition number of the neural tangent kernel $\kappa_{\mathcal{N}}$ to evaluate candidate architectures, which are data-dependent metrics computed using mini-batches from a training dataset. In our work, we use the data-agnostic metric `synflow` as our performance objective. The resulting architectures are thus potentially applicable to a wider range of tasks and datasets.

This article extends our SoICT 2022 conference paper on training-free multi-objective and many-objective evolutionary NAS [29]. In [29], we discussed several multi-objective and many-objective NAS problem formulations and employed the well-known multi-objective evolutionary algorithm NSGA-II to solve these formulations. Moreover, we exclusively used the data-agnostic training-free metric `synflow` to evaluate candidate architecture performance without any training. In this article, we extend the analysis in our preliminary work by adding the hypervolume performance indicator results instead of only Inverted Generational Distance (IGD). While IGD exhibits the convergence behavior of a multi-objective algorithm, it cannot be used in real-world situations due to its requirement of the Pareto-optimal front (see Sections 2.1 and 4.2.1). The hypervolume, which requires only a reference nadir point, is a more practical performance indicator for evaluating and comparing multi-objective NAS approaches (see Section 4.2.2). Employing both IGD and hypervolume thus yields more detailed investigations into the effectiveness of different NAS problem formulations. We present the IGD and hypervolume results in terms of GPU hours rather than the number of generations, which allows us to better assess the efficiency of our approaches. Our experimental results demonstrate that Training-Free Many-Objective Evolutionary NAS (TF-MaOENAS) provides several advantages when achieving competitive results while taking only 3 GPU hours.

2 Backgrounds

2.1 Multi-objective neural architecture search

Multi-Objective NAS (MONAS) [20, 26] can be formulated as searching for high-quality architectures in a search space Ω where m different aspects (e.g., error rate, model size, or latency) are optimized simultaneously. Each aspect is modeled as a separate objective $f_i(\boldsymbol{x})$, $i \in \{1, \ldots, m\}$, and each candidate architecture $\boldsymbol{x} \in \Omega$ thus has a corresponding vector of objective values $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$. All objectives, without loss of generality, are assumed to be minimized.

An architecture $\boldsymbol{x}$ *dominates* another architecture $\boldsymbol{y}$ if and only if $\boldsymbol{x}$ strictly outperforms $\boldsymbol{y}$ in at least one aspect and $\boldsymbol{x}$ is never outperformed by $\boldsymbol{y}$ in any aspects:

$$\boldsymbol{x} \prec \boldsymbol{y} \iff \forall i, f_i(\boldsymbol{x}) \le f_i(\boldsymbol{y}) \wedge \exists i, f_i(\boldsymbol{x}) < f_i(\boldsymbol{y})$$

If some objectives conflict with each other, e.g., network prediction accuracy versus network complexity, there will not exist an *ideal* architecture optimizing all those competing objectives. Instead, there exists a ***Pareto set*** P_S of architectures, in which all can be considered Pareto-optimal because they are not dominated by any other architectures:

$$P_S = \{\boldsymbol{x} \in \Omega \mid \nexists \boldsymbol{x}' \in \Omega, \boldsymbol{x}' \prec \boldsymbol{x}\}$$

The images of all Pareto-optimal architectures in P_S together form a ***Pareto-optimal front*** P_F in the objective space [5, 23]:

$$P_F = \{\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^m \mid \boldsymbol{x} \in P_S\}$$

Each point on P_F denotes the vector of objective values $\boldsymbol{f}(\boldsymbol{x})$ of a Pareto-optimal architecture $\boldsymbol{x}$, which exhibits an optimal trade-off among the competing objectives. For example, from a Pareto-optimal architecture $\boldsymbol{z}$, if we modify $\boldsymbol{z}$ to improve network performance (e.g., prediction accuracy), network complexity (e.g., model size or FLOPs) must be increased as well. In other words, there exists no means in the search space Ω to alter $\boldsymbol{z}$ in order to increase accuracy performance without incurring additional computation cost. A Pareto-optimal front P_F thus exhibits insightful information for decision-makers, e.g., which Pareto-optimal architecture on P_F exhibits the most desirable trade-off between network latency and accuracy.

The optimal solution of MONAS is not a single ideal architecture but the Pareto set P_S. However, achieving the entire P_S is prohibitively costly (if there are many Pareto-optimal architectures) and unnecessary (if choosing between architectures close to each other on the Pareto-optimal front P_F does not make considerable differences). It is often more practical to find an *approximation set* S that yields a good *approximation front* $\boldsymbol{f}(S)$ well approximating the Pareto-optimal front P_F in terms of both proximity and diversity [6, 21].

2.2 Non-dominated sorting genetic algorithm II

Evolutionary Algorithms (EAs) are often employed for handling multi-objective optimization problems because their intrinsic population-based operations are well-suited for the goal of finding multiple non-dominated solutions to approximate Pareto-optimal fronts [5, 15, 23]. In this article, we consider the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [6] as the optimization algorithm. NSGA-II has also been widely used for solving different MONAS problem instances as well [19, 20]. In the following paragraph, we provide a brief description of NSGA-II, and further details can be found in [6].

The NSGA-II population P is initialized with N individuals, where each individual corresponds with a candidate architecture randomly sampled from the search space Ω. Until the computation budget is over, or other termination criteria are met, NSGA-II operates in a generational manner as follows. In every generation, a set S of N promising individuals in terms of Pareto dominance are selected from P via binary tournament selection. N new candidate architectures (i.e., set O of offspring individuals) are generated from the parent architectures (i.e., set S of selected individuals) via variation operators (i.e., crossover and mutation) and are evaluated for their objective values. The current population P and the offspring population are then merged into a pool $(P+O)$ where all $2N$ individuals are sorted into their *non-domination ranks* $0, 1, 2, \ldots$. Rank 0 consists of individuals that are not dominated by any other individuals in $(P + O)$, and rank i consists of individuals that would be non-dominated if individuals from lower ranks ($< i$) are omitted. A group of N promising individuals are then selected from the pool $(P + O)$ via truncation selection to form the population for the next generation. Lower-rank individuals are selected first, and if selections need to be performed among individuals of the same rank, far-apart individuals are preferred.

2.3 Training-free performance metric synaptic flow

Synaptic Flow (`synflow`) is a metric for measuring the importance of each parameter in a neural network architecture, based on the inter-layer interaction of other network parameters. Tanaka et al. [28] first introduced the `synflow` score for single parameter $w_{ij}^{[l]}$ in the l-th layer of a fully-connected neural network as follows:

$$\mathcal{P}(w_{ij}^{[l]}) = \left[\mathbf{1}^T \prod_{k=l+1}^{N} \left|W^{[k]}\right|\right]_i \left|w_{ij}^{[l]}\right| \left[\prod_{k=1}^{l-1} \left|W^{[k]}\right| \mathbf{1}\right]_j \tag{1}$$

where $W^{[k]}$ is the weight matrix of the k-th layer of the network. The `synflow` score for a parameter $w_{ij}^{[l]}$ takes into account the product of the absolute values of the weights of all the layers downstream from the current layer l, as well as the product of the absolute values of the weights of all the layers upstream from the current layer l. Thus, the `synflow` score of a parameter $\mathcal{P}_S(w_{ij}^{[l]})$ reflects the contribution of that parameter to the information flow of the network.

Abdelfattah et al. [1] then extended the `synflow` score to evaluate the entire network architecture $\boldsymbol{x}$, which is sum of the `synflow` scores for all M parameters in the network as follows:

$$\mathcal{S}(\boldsymbol{w}(\boldsymbol{x})) = \sum_{i=1}^{M} \mathcal{P}(w_i(\boldsymbol{x})) \tag{2}$$

According to [1], the training-free performance metric `synflow` exhibits a strong correlation with the final accuracy of the network in the NAS-Bench-201 architecture search space, with Spearman ρ coefficients of 0.74, 0.76, and 0.75 on CIFAR-10, CIFAR-100, and ImageNet16-120, respectively. The bi-objective space of test accuracy after 200 training epochs versus FLOPs for all architectures in NATS-Bench is depicted in Figure 1. According to the graph, architectures with greater `synflow` scores tend to have higher test accuracy. Furthermore, `synflow` is a data-agnostic metric that can be computed solely based on network weights (see Equation 1). Unlike other performance metrics that do not require training, such as `jacob_cov` [22] or the condition number of the NTK [4], `synflow` does not need any data mini-batches to be used as input for the network. It means that `synflow` can be used to measure the performance of a neural network without having to pass any data through it. This is beneficial since it allows for a more efficient evaluation of a neural network's performance without having to expend resources on data collection and pre-processing. Since `synflow` is data-independent and does not quire any training epochs, it can serve as an effective proxy for optimizing network accuracy in tackling NAS problems [1].

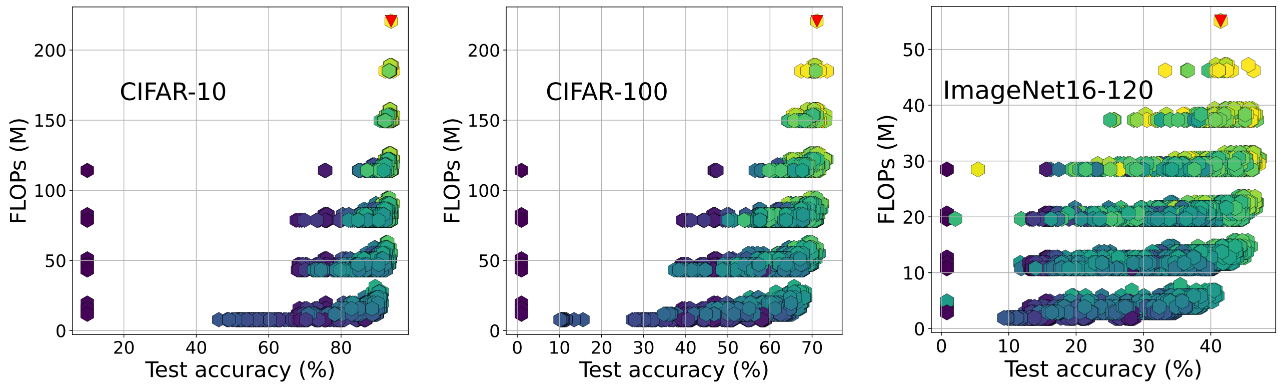

Figure 1: Illustration of all network architectures in the NATS-Bench search space. Brighter hexagons indicate greater values of `synflow`, while red triangles denote the architectures with the highest `synflow` values.

3 NAS problem formulations

3.1 Multi-objective NAS problem formulation

NAS can be formulated as a multi-objective optimization problem, which seeks to simultaneously optimize two objectives, such as accuracy and computational complexity, as follows:

$$\begin{aligned} \min \quad & \boldsymbol{F}(\boldsymbol{x}) = (f_{\text{err}}(\boldsymbol{x}, \boldsymbol{w}^*(\boldsymbol{x}), \mathcal{D}_{\text{val}}), f_{\text{comp}}(\boldsymbol{x})) \in \mathbb{R}^2, \\ \text{st} \quad & \boldsymbol{w}^*(\boldsymbol{x}) \in \arg\min \mathcal{L}(\boldsymbol{x}, \boldsymbol{w}(\boldsymbol{x}), \mathcal{D}_{\text{train}}), \\ & \boldsymbol{x} \in \Omega_{\text{arch}}, \quad \boldsymbol{w}(\boldsymbol{x}) \in \Omega_{\text{weight}}(\boldsymbol{x}), \end{aligned} \tag{3}$$

where $\boldsymbol{x}$ denotes an architecture in the search space Ω_{arch}. Multi-Objective NAS (MONAS), aiming to find a set of architectures that exhibit optimal trade-offs between accuracy and complexity, is a bi-level optimization problem. At the upper level, it seeks high-quality candidate architectures that optimize both error rate f_{err} and complexity f_{comp}, while at the lower level, it searches for the proper network weight values $\boldsymbol{w}^*(\boldsymbol{x})$ for each candidate architecture $\boldsymbol{x}$. The network weight values $\boldsymbol{w}(\boldsymbol{x})$ must be specified in order to accurately evaluate the error rate of a network architecture $\boldsymbol{x}$. This requires solving a lower-level optimization problem over the network weight space $\Omega_{\text{weight}}(\boldsymbol{x})$ of the given architecture $\boldsymbol{x}$. By doing so, we can obtain a set of weight values that minimize the error rate and maximize the performance of the network. This is typically done by employing a stochastic gradient descent (SGD) algorithm to perform many iterative updates on network weight values in order to minimize a loss function $\mathcal{L}$, which measures the difference between network predictions and ground-truth targets for data items in a training dataset D_{train}. This process can be computationally expensive and time-consuming, but is necessary in order to accurately obtain the proper values of $\boldsymbol{w}(\boldsymbol{x})$ for any given architecture.

The two optimization objectives at the upper level of MONAS are minimizing error rate f_{err} and minimizing complexity f_{comp}. The complexity of a network can be assessed via metrics such as the number of floating point operations (FLOPs), latency, the number of parameters (#parameters), or the number of multiply-accumulate units (#MACs). These metrics can be calculated without the weights of the network, and thus require minimal computing time. Besides, in order to prevent overfitting to the training dataset D_{train}, it is important to calculate error rate f_{err} on a separate validation dataset D_{val} that has not been used for training. At the end of a search, the error rates and weight of resulting architectures should be tested on a new dataset D_{test} to measure their ability to generalize.

3.2 Training-free multi-objective NAS problem formulation

Evaluating the prediction performance of multiple candidate architectures in MONAS requires computationally intensive training procedures (i.e., lower-level optimization) which consume a significant amount of computing time (see Equation 3). To eliminate this cost, several NAS formulations have been proposed to use training-free performance metrics as proxies for the network error rate f_{err}. This approach allows us to quickly evaluate candidate architectures without having to perform costly training procedures. We present a training-free bi-objective NAS formulation that uses the `synflow` metric as an alternative to f_{err} as follows:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{F}(\boldsymbol{x}) = (f_{\text{SF}}(\boldsymbol{x}, \boldsymbol{w}(\boldsymbol{x})), f_{\text{comp}}(\boldsymbol{x})) \in \mathbb{R}^2, \\ \text{subject to} \quad & \boldsymbol{x} \in \Omega_{\text{arch}}, \quad \boldsymbol{w}(\boldsymbol{x}) \in \Omega_{\text{weight}}(\boldsymbol{x}), \end{aligned} \tag{4}$$

where $f_{\text{SF}}(\boldsymbol{x}, \boldsymbol{w}(\boldsymbol{x})) = -\mathcal{S}(\boldsymbol{w}(\boldsymbol{x}))$ (as `synflow` should be maximized). At the start of the lower level optimization process, $\boldsymbol{w}(\boldsymbol{x})$ can be initialized randomly to compute their `synflow` scores. We name this formulation TF-MONAS.

3.3 Training-free many-objective NAS problem formulation

The TF-MONAS formulation in Equation 4 can be further extended by incorporating many objectives simultaneously. This approach allows for the simultaneous consideration of multiple complexity metrics, such as FLOPs, latency, #parameters, and #MACs. By optimizing the neural network architecture from various perspectives, this approach can ensure that the resultant architecture is suitable for a variety of applications. A common scenario is the use of deep neural networks on embedded devices, such as drones or smart watches. This requires taking into account a variety of hardware limitations, such as model size and storage capacity, as well as usage requirements like response time. All of these must be considered when deploying deep neural networks on embedded devices in order to ensure reliable performance. Additionally, we note that these complexity metrics can be evaluated without incurring too much computing cost. We formulate training-free many-objective evolutionary NAS (TF-MaOENAS) that does not require any training and consists of five objectives as follows:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{F}(\boldsymbol{x}) = (f_{\text{SF}}(\boldsymbol{x}, \boldsymbol{w}(\boldsymbol{x})), f_{\text{MACs}}(\boldsymbol{x}), \\ & f_{\text{latency}}(\boldsymbol{x}), f_{\text{FLOPs}}(\boldsymbol{x}), f_{\text{params}}(\boldsymbol{x})), \\ \text{subject to} \quad & \boldsymbol{x} \in \Omega_{\text{arch}}, \quad \boldsymbol{w}(\boldsymbol{x}) \in \Omega_{\text{weight}}(\boldsymbol{x}), \end{aligned} \tag{5}$$

The complexity metrics used in this work are FLOPs , latency, the number of parameters (#parameters), and #MACs. We name this formulation TF-MaOENAS. We will compare solving one TF-MaOENAS formulation that involves four optimization objectives for network complexity against solving separately four different TF-MOENAS formulations in which each model considers only one complexity objective. Moreover, we also compare TF-MaOENAS with (training-based) MaOENAS to demonstrate the benefits of the training-free performance metric `synflow`.

4 Experiments

4.1 Experimental details

Our experiments are carried out on NATS-Bench [8], which is an extended version of NAS-Bench-201 [11]. NATS-Bench comprises 15,625 architectures and provides a variety of metrics for evaluating different architectures, such as accuracy, number of parameters, and training time, across three datasets: CIFAR-10, CIFAR-100, and ImageNet16-120. We experiment with four MONAS approaches on NATS-Bench, each with specific optimization objectives as follows:

1. **04 MOENAS variants**: 01 training-based performance metric (validation accuracy after 12 training epochs as in other related works [11, 26]) versus 01 complexity metric (FLOPs, #parameters, latency, or #MACs).

2. **04 TF-MOENAS variants**: 01 training-free performance metric (`synflow`) versus 01 complexity metric (FLOPs, #parameters, latency, or #MACs).

3. **01 MaOENAS variant**: 01 training-based performance metric (validation accuracy after 12 training epochs) versus 04 complexity metrics (FLOPs, latency, #parameters, and #MACs).

4. **01 TF-MaOENAS variant**: 01 training-free performance metric (`synflow`) versus 04 complexity metrics (FLOPs, latency, #parameters, and #MACs).

We use the NSGA-II [6] as our multi-objective search algorithm. We set the population size to 20, the number of generations to 50, and used random initialization. We also employ binary tournament selection, two-point crossover with a probability of 0.9, and polynomial mutation with a probability of $1/l$, where l is the encoding length of each individual.

Besides, we also implement an *elitist archive* [21] to save non-dominated architectures discovered so far throughout the NAS process. When an architecture is evaluated, it is included in the elitist archive if it is not dominated by any existing architectures in the elitist archive. Existing architectures that are dominated by newly added architecture will be removed from the elitist archive. The non-dominated architectures in the elitist archive, therefore, constitute an approximation set, which may be regarded as the NSGA-II optimization result. The elitist archive is only used for result logging and does not impact the workings of NSGA-II. Because non-dominated solutions might be lost due to the stochasticity of the variation and selection operators, an elitist archive is desirable for multi-objective evolutionary algorithms.

We conduct 21 independent runs of NSGA-II for each problem formulation presented in Section 3 on CIFAR-10, CIFAR-100, and ImageNet16-120 of NATS-Bench. All of our experiments can be performed using Google Colab.

4.2 Performance metric

4.2.1 Inverted generational distance

To compare an approximation set S of non-dominated architectures against the Pareto-optimal front P_F of the most efficient trade-off architectures, we employ the *Inverted Generational Distance* (IGD) [3] which is defined as:

$$\text{IGD}(\mathcal{S}, P_F) = \frac{1}{|P_F|} \sum_{p \in P_F} \min_{\boldsymbol{x} \in \mathcal{S}} \|p - \boldsymbol{f}(\boldsymbol{x})\|_2 \tag{6}$$

The smaller IGD indicates the better approximation front achieved by the current solutions. For example, if $\text{IGD}(\mathcal{S}_1, P_F) < \text{IGD}(\mathcal{S}_2, P_F)$, then $\mathcal{S}_1$ is a better approximation front compared to $\mathcal{S}_2$ regarding P_F. The Pareto-optimal front P_F can be obtained by iterating over all architectures in the NAS benchmark. The Pareto-optimal front P_F is computed by querying the database of NATS-Bench for test accuracy values after 200 epochs. Approximation sets $\mathcal{S}$ are taken from the elitist archive obtained from the search process. The IGD value between the archive and the Pareto-optimal front P_F can be calculated after each evolutionary generation to measure how close the current approximation front of the algorithm is to the front of

Pareto-optimal architectures P_F. The test accuracy after 200 epochs and IGD values are only used to assess the effectiveness of the search algorithms and are not employed to direct the search process.

4.2.2 Hypervolume

Hypervolume [3, 18] is also a measure of the quality of a set of non-dominated solutions in multi-objective optimization besides IGD. It can be computed by measuring the area covered by the solution points on the approximation front with regard to a *reference point*. In contrast to IGD, which requires the Pareto-optimal front P_F to compute (IGD can thus hardly be used in real-world multi-objective optimization), hypervolume only need a reference point to be specified, which is usually the nadir point (the worst point in the objective space). The higher hypervolume implies that the corresponding method achieves a better approximation front. For example, if Hypervolume$(\mathcal{S}_1) >$ Hypervolume$(\mathcal{S}_2)$, then $\mathcal{S}_1$ is a better approximation front compared to $\mathcal{S}_2$.

4.3 Result analysis

	IGD	Hypervolume	Test accuracy	Search cost (hours)
	Space: Test accuracy - FLOPs			
(1)	0.0198 ± 0.0171	1.0332 ± 0.0013	94.28 ± 0.17	53.7
(2)	0.0250 ± 0.0133	1.0223 ± 0.0027	94.29 ± 0.17	0.7
(3)	0.0308 ± 0.0177	$\underline{1.0334 \pm 0.0011}$	94.27 ± 0.13	54.8
(4)	$\underline{0.0096 \pm 0.0021}$	1.0298 ± 0.0022	$\underline{94.37 \pm 0.00}$	2.7
	Space: Test accuracy - Latency			
(1)	$\underline{\mathbf{0.0228 \pm 0.0019}}$	$\underline{\mathbf{1.0050 \pm 0.0006}}$	94.30 ± 0.09	54.1
(2)	0.0532 ± 0.0056	0.9431 ± 0.0200	94.29 ± 0.14	1.1
(3)	0.0277 ± 0.0060	0.9967 ± 0.0168	94.27 ± 0.13	54.8
(4)	0.0412 ± 0.0060	0.9581 ± 0.0098	$\underline{94.37 \pm 0.00}$	2.7
	Space: Test accuracy - #Parameters			
(1)	0.0180 ± 0.0138	1.0332 ± 0.0014	94.27 ± 0.18	53.8
(2)	0.0314 ± 0.0170	1.0233 ± 0.0027	94.24 ± 0.22	0.8
(3)	0.0309 ± 0.0176	$\underline{1.0334 \pm 0.0011}$	94.27 ± 0.13	54.8
(4)	$\underline{0.0098 \pm 0.0022}$	1.0296 ± 0.0023	$\underline{94.37 \pm 0.00}$	2.7
	Space: Test accuracy - #MACs			
(1)	0.0195 ± 0.0131	1.0331 ± 0.0017	94.24 ± 0.22	53.8
(2)	0.0189 ± 0.0069	1.0280 ± 0.0034	94.35 ± 0.03	0.8
(3)	0.0266 ± 0.0150	$\underline{1.0333 \pm 0.0011}$	94.27 ± 0.13	54.8
(4)	$\underline{\mathbf{0.0104 \pm 0.0023}}$	1.0292 ± 0.0025	$\underline{94.37 \pm 0.00}$	2.7

Table 1: Results of search and evaluation directly on CIFAR-10: (1) MOENAS, (2) TF-MOENAS, (3) MaOENAS, (4) TF-MaOENAS. Results that are $\underline{\text{underlined}}$ indicate the best method and results that are **bolded** denote the best method with statistical significance (p-value < 0.01)

Figure 2 demonstrates that TF-MaOENAS achieves superior IGD convergence results compared to other approaches while taking just 3 GPU hours in most cases, with the exception of test accuracy versus latency space. However, in terms of hypervolume, MaOENAS and MOENAS alternatively surpass other approaches on CIFAR-10 and ImageNet16-120. Table 1, Table 2, and Table 3 show comprehensive results on CIFAR-10, CIFAR-100 and ImageNet16-120. It is noted that the hypervolume of TF-MaOENAS still outperforms other methods in the majority of cases on CIFAR-100, and its hypervolume is only slightly lower than that of other training-based methods on

	IGD	Hypervolume	Test accuracy	Search cost (hours)
	Space: Test accuracy - FLOPs			
(1)	0.0384 ± 0.0086	0.7958 ± 0.0015	72.39 ± 0.21	53.8
(2)	0.0493 ± 0.0176	0.7851 ± 0.0036	72.56 ± 0.44	0.8
(3)	0.0334 ± 0.0128	0.7964 ± 0.0019	72.40 ± 0.30	54.8
(4)	$\underline{\mathbf{0.0122 \pm 0.0045}}$	$\underline{\mathbf{0.7993 \pm 0.0019}}$	$\underline{73.49 \pm 0.07}$	2.7
	Space: Test accuracy - Latency			
(1)	$\underline{0.0318 \pm 0.0070}$	$\underline{0.7960 \pm 0.0013}$	72.68 ± 0.68	54.0
(2)	0.1182 ± 0.0139	0.7460 ± 0.0149	$\underline{73.51 \pm 0.00}$	1.0
(3)	0.0352 ± 0.0084	0.7701 ± 0.0057	72.40 ± 0.30	54.8
(4)	0.0446 ± 0.0057	0.7539 ± 0.0076	73.49 ± 0.07	2.7
	Space: Test accuracy - #Parameters			
(1)	0.0369 ± 0.0029	0.7960 ± 0.0013	72.47 ± 0.23	53.8
(2)	0.0189 ± 0.0038	0.7883 ± 0.0025	73.47 ± 0.11	0.8
(3)	0.0335 ± 0.0127	0.7963 ± 0.0019	72.40 ± 0.30	54.8
(4)	$\underline{\mathbf{0.0123 \pm 0.0045}}$	$\underline{\mathbf{0.7990 \pm 0.0020}}$	$\underline{73.49 \pm 0.07}$	2.7
	Space: Test accuracy - #MACs			
(1)	0.0313 ± 0.0094	0.7956 ± 0.0021	72.39 ± 0.36	53.8
(2)	0.0156 ± 0.0025	0.7941 ± 0.0041	$\underline{73.51 \pm 0.00}$	0.8
(3)	0.0270 ± 0.0096	0.7961 ± 0.0018	72.40 ± 0.30	54.8
(4)	$\underline{\mathbf{0.0126 \pm 0.0041}}$	$\underline{0.7985 \pm 0.0021}$	73.49 ± 0.07	2.7

Table 2: Results of search and evaluation directly on CIFAR-100: (1) MOENAS, (2) TF-MOENAS, (3) MaOENAS, (4) TF-MaOENAS. Results that are $\underline{\text{underlined}}$ indicate the best method and results that are **bolded** denote the best method with statistical significance (p-value < 0.01)

CIFAR-10 and ImageNet16-120. Furthermore, because it is a training-free approach, it only requires 3 GPU hours as opposed to dozens to hundreds of GPU hours for training-based methods like MOENAS and MaOENAS. Regarding test accuracy, TF-MaOENAS discovers top-performing architectures on NATS-Bench and outperforms other methods in the majority of situations.

The experimental results also show that TF-MaOENAS and TF-MOENAS (using `synflow`) perform better than MaOENAS and MOENAS (using validation accuracy after 12 training epochs), respectively. This indicates that using `synflow` is more effective at optimizing for multiple objectives simultaneously than using the validation accuracy after 12 training epochs. This might reflect that the training-free `synflow` metric is more capable of measuring and balancing between optimizing for accuracy and other complexity objectives. Moreover, `synflow` is a training-free metric, it just takes a few seconds to compute, resulting in a lower computing cost than a training-based metric. On the other hand, TF-MaOENAS, which employs five objectives concurrently, outperforms TF-MOENAS, which employs only two objectives. This is due to the addition of MACs, the number of parameters, and latency as complexity objectives in addition to `synflow` and FLOPs. Most of the time, optimizing more objectives is favorable while not incurring considerably more computing expenses. This will provide a fuller picture of the complexity of achieved architectures, enabling a more precise evaluation of the trade-offs between performance and complexity. Additionally, the penta-objective approximation fronts obtained by TF-MaOENAS can be projected into different bi-objective spaces (i.e., test accuracy versus one complexity metric) and still achieve better results than the corresponding TF-MOENAS variants. This means that running TF-MaOENAS once in the penta-objective space can obtain good approximation fronts in different bi-objective

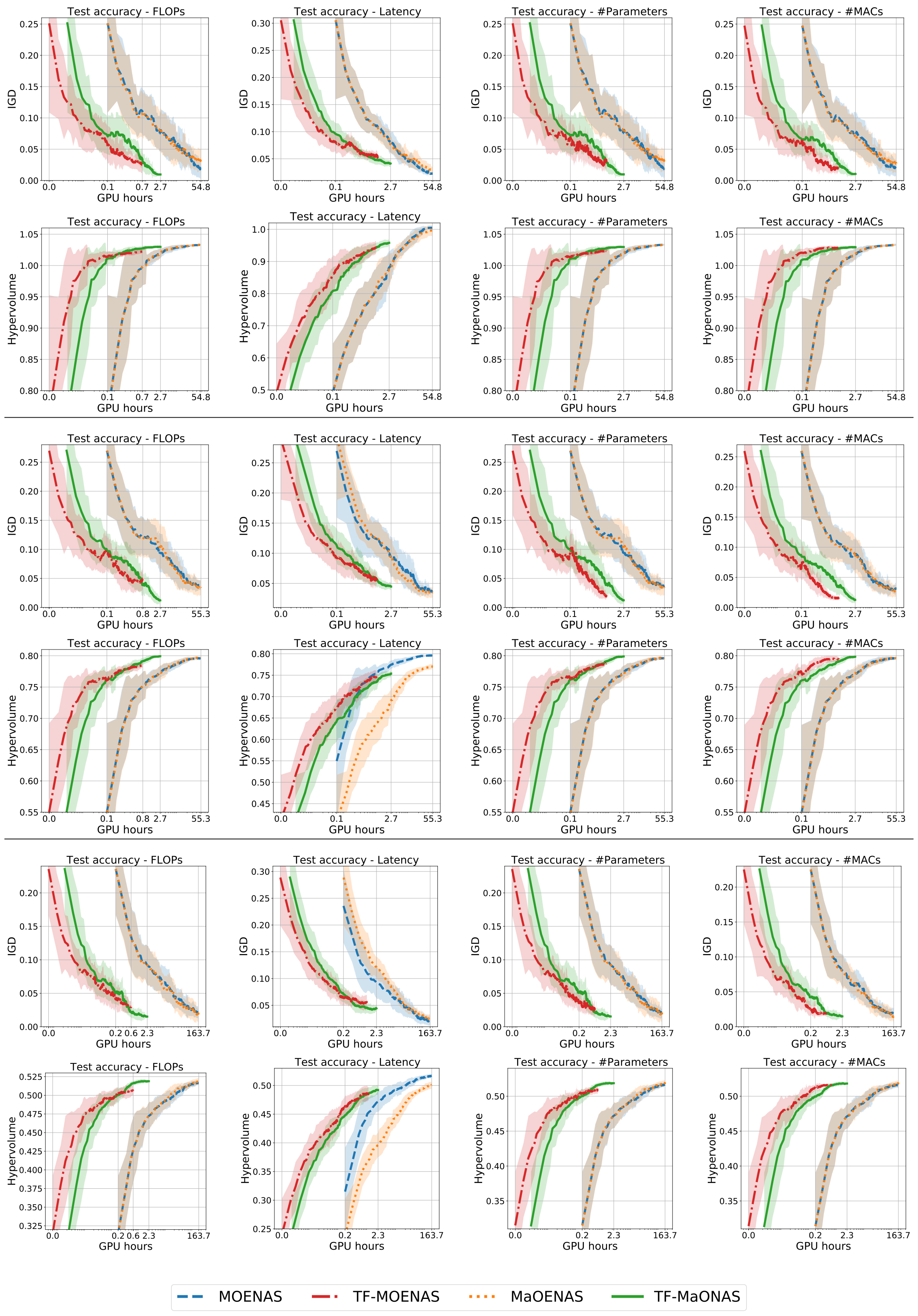

Figure 2: IGD and hypervolume comparisons in terms of GPU hours (log scale) on four different bi-objective spaces (plot title) across CIFAR-10 (top two rows), CIFAR-100 (middle two rows) and ImageNet16-120 (bottom two rows). The figures depict the mean values with lines and the standard deviation with shaded areas over 21 runs.

	IGD	Hypervolume	Test accuracy	Search cost (hours)
		Space: Test accuracy - FLOPs		
(1)	0.0217 ± 0.0087	0.5165 ± 0.0026	46.34 ± 0.35	161.8
(2)	0.0296 ± 0.0089	0.5062 ± 0.0062	46.25 ± 0.15	0.6
(3)	0.0192 ± 0.0165	$\underline{0.5193 \pm 0.0032}$	46.41 ± 0.43	163.7
(4)	$\underline{0.0151 \pm 0.0019}$	0.5189 ± 0.0017	$\underline{46.57 \pm 0.05}$	2.3
		Space: Test accuracy - Latency		
(1)	0.0281 ± 0.0047	$\underline{0.5171 \pm 0.0019}$	$\underline{46.62 \pm 0.52}$	162.2
(2)	0.0543 ± 0.0112	0.4852 ± 0.0118	46.52 ± 0.17	1.1
(3)	$\underline{0.0192 \pm 0.0165}$	0.5012 ± 0.0059	46.41 ± 0.43	163.7
(4)	0.04428 ± 0.0062	0.4922 ± 0.0086	46.57 ± 0.05	2.3
		Space: Test accuracy - #Parameters		
(1)	0.0194 ± 0.0067	0.5171 ± 0.0019	46.46 ± 0.24	161.8
(2)	0.0264 ± 0.0114	0.5092 ± 0.0047	46.40 ± 0.18	0.8
(3)	0.0194 ± 0.0165	$\underline{0.5191 \pm 0.0032}$	46.41 ± 0.43	163.7
(4)	$\underline{0.0153 \pm 0.0019}$	0.5186 ± 0.0017	$\underline{46.57 \pm 0.05}$	2.3
		Space: Test accuracy - #MACs		
(1)	0.0198 ± 0.0073	0.5153 ± 0.0039	46.17 ± 0.46	161.8
(2)	0.0188 ± 0.0020	0.5161 ± 0.0020	$\underline{46.57 \pm 0.02}$	0.6
(3)	0.0156 ± 0.0107	$\underline{0.5188 \pm 0.0032}$	46.41 ± 0.43	163.7
(4)	$\underline{\mathbf{0.0148 \pm 0.0022}}$	0.5181 ± 0.0017	46.57 ± 0.05	2.3

Table 3: Results of search and evaluation directly on ImageNet16-120: (1) MOENAS, (2) TF-MOENAS, (3) MaOENAS, (4) TF-MaOENAS. Results that are underlined indicate the best method and results that are **bolded** denote the best method with statistical significance (p-value < 0.01)

Alg.	CIFAR-10 (direct)	CIFAR-100 (transfer)	ImageNet16-120 (transfer)	Search cost (hours)
		Space: Test accuracy - FLOPs		
(1)	1.0332 ± 0.0013	0.7962 ± 0.0043	0.5167 ± 0.0051	53.7
(2)	1.0223 ± 0.0027	0.7830 ± 0.0054	0.5061 ± 0.0057	0.7
(3)	$\underline{1.0334 \pm 0.0011}$	$\underline{0.7996 \pm 0.0023}$	$\underline{0.5191 \pm 0.0028}$	54.8
(4)	1.0298 ± 0.0022	0.7958 ± 0.0039	0.5169 ± 0.0015	2.7
		Space: Test accuracy - Latency		
(1)	$\underline{1.0050 \pm 0.0006}$	$\underline{0.7589 \pm 0.0028}$	0.4861 ± 0.0056	54.1
(2)	0.9431 ± 0.0200	0.6425 ± 0.0284	0.4164 ± 0.0179	1.1
(3)	0.9967 ± 0.0168	0.7545 ± 0.0159	$\underline{0.4897 \pm 0.0067}$	54.8
(4)	0.9581 ± 0.0098	0.7234 ± 0.0140	0.4710 ± 0.0055	2.7
		Space: Test accuracy - #Parameters		
(1)	1.0332 ± 0.0014	0.7963 ± 0.0044	0.5155 ± 0.0055	53.8
(2)	1.0233 ± 0.0027	0.7824 ± 0.0054	0.5056 ± 0.0057	0.8
(3)	$\underline{1.0334 \pm 0.0011}$	$\underline{\mathbf{0.7995 \pm 0.0023}}$	$\underline{0.5189 \pm 0.0028}$	54.8
(4)	1.0296 ± 0.0023	0.7954 ± 0.0040	0.5166 ± 0.0016	2.7
		Space: Test accuracy - #MACs		
(1)	1.0331 ± 0.0017	0.7964 ± 0.0048	0.5165 ± 0.0055	53.8
(2)	1.0280 ± 0.0034	0.7803 ± 0.0058	0.5042 ± 0.0060	0.8
(3)	$\underline{1.0333 \pm 0.0011}$	$\underline{0.7992 \pm 0.0023}$	$\underline{0.5186 \pm 0.0028}$	54.8
(4)	1.0292 ± 0.0025	0.7947 ± 0.0042	0.5160 ± 0.0016	2.7

Table 5: Hypervolume on transfer learning task: (1) MOENAS, (2) TF-MOENAS, (3) MaOENAS, (4) TF-MaOENAS. Results that are underlined indicate the best method and results that are **bolded** denote the best method with statistical significance (p-value < 0.01)

Alg.	CIFAR-10 (direct)	CIFAR-100 (transfer)	ImageNet16-120 (transfer)	Search cost (hours)
		Space: Test accuracy - FLOPs		
(1)	0.0198 ± 0.0171	0.0465 ± 0.0183	0.0316 ± 0.0147	53.7
(2)	0.0250 ± 0.0133	0.0322 ± 0.0103	0.0400 ± 0.0145	0.7
(3)	0.0308 ± 0.0177	0.0299 ± 0.0106	0.0230 ± 0.0091	54.8
(4)	$\underline{0.0096 \pm 0.0021}$	$\underline{\mathbf{0.0125 \pm 0.0017}}$	$\underline{\mathbf{0.0161 \pm 0.0016}}$	2.7
		Space: Test accuracy - Latency		
(1)	$\underline{\mathbf{0.0228 \pm 0.0019}}$	0.0419 ± 0.0056	0.0416 ± 0.0103	54.1
(2)	0.0532 ± 0.0056	0.0932 ± 0.0100	0.0841 ± 0.0175	1.1
(3)	0.0277 ± 0.0060	$\underline{0.0390 \pm 0.0093}$	$\underline{0.0369 \pm 0.0049}$	54.8
(4)	0.0412 ± 0.0060	0.0612 ± 0.0091	0.0577 ± 0.0097	2.7
		Space: Test accuracy - #Parameters		
(1)	0.0180 ± 0.0138	0.0413 ± 0.0171	0.0342 ± 0.0139	53.8
(2)	0.0314 ± 0.0170	0.0502 ± 0.0144	0.0306 ± 0.0092	0.8
(3)	0.0309 ± 0.0176	0.0300 ± 0.0106	0.0231 ± 0.0090	54.8
(4)	$\underline{0.0098 \pm 0.0022}$	$\underline{\mathbf{0.0124 \pm 0.0017}}$	$\underline{\mathbf{0.0164 \pm 0.0017}}$	2.7
		Space: Test accuracy - #MACs		
(1)	0.0195 ± 0.0131	0.0348 ± 0.0129	0.0250 ± 0.0069	53.8
(2)	0.0189 ± 0.0069	0.0322 ± 0.0085	0.0197 ± 0.0036	0.8
(3)	0.0266 ± 0.0150	0.0247 ± 0.0083	0.0188 ± 0.0060	54.8
(4)	$\underline{\mathbf{0.0104 \pm 0.0023}}$	$\underline{\mathbf{0.0137 \pm 0.0024}}$	$\underline{0.0163 \pm 0.0022}$	2.7

Table 4: IGD on transfer learning task: (1) MOENAS, (2) TF-MOENAS, (3) MaOENAS, (4) TF-MaOENAS. Results that are underlined indicate the best method and results that are **bolded** denote the best method with statistical significance (p-value < 0.01)

spaces simultaneously, rather than having to run separately TF-MOENAS many times for each bi-objective space.

We note that the variation in the obtained results across the datasets (see Tables 1, 2, 3) can be attributed to the following reasons. First, the performance metrics (i.e., accuracy or `synflow`) and some complexity metrics (e.g., latency or FLOPs) of each candidate architecture vary across the datasets (e.g., the accuracy of an architecture on CIFAR-10 is different from its accuracy on ImagetNet16-120). Therefore, the IGD and hypervolume results of each NAS method are different from one dataset to another. Second, we assess the effectiveness of NAS methods using the test accuracy after 200 training epochs but, during the search process of each NAS algorithm, the validation accuracy after 12 training epochs (for training-based approaches) or `synflow` (for training-free approaches) are employed as the performance objective (see experimental details in Section 4.1). The correlation of 12-epoch validation accuracy or `synflow` with the final test accuracy (after 200 epochs) varies per dataset [1] (e.g., the correlation coefficients of `synflow` for CIFAR-10, CIFAR-100, and ImageNet16-120 are 0.74, 0.76, and 0.75, respectively). Therefore, the rankings of the considered NAS methods might differ across the datasets.

4.4 Tranferability

This section explores the potential of transfer learning in NAS by evaluating the transferability of architectures discovered through multi-objective and many-objective NAS problem formulations. The final approximation front (i.e., the elitist archive) of architectures on CIFAR-10 is re-evaluated on CIFAR-100 and ImageNet16-120 for their performance and complexity. Transfer learning in NAS offers several benefits, including the reduced computational cost and the potential for faster deployment of deep learning models in real-world applications by identifying architectures that are highly transferable across datasets.

Table 4 and Table 5 show that TF-MaOENAS yields better IGD compared to other methods, whereas MaOENAS outperforms other methods in hypervolume in most cases. In terms of test accuracy, TF-MaOENAS also completely surpasses most of the approaches in Table 6, with better accuracy and lower search costs. It indicates that TF-MaOENAS using the training-free performance metric `synflow` are more effective at transferring knowledge from one dataset to another. Besides, both penta-objective approaches TF-MaOENAS and MaOENAS give better IGD and hypervolume, respectively, than bi-objective approaches. Although the four TF-MOENAS approaches have lower computing time, the optimization result of TF-MaOENAS is a penta-objective approximation front that

	CIFAR-10 (direct)	CIFAR-100 (transfer)	ImageNet16-120 (transfer)	Search cost (hours)
		Manually designed		
ResNet [14]	93.97	70.86	43.63	-
		Weight sharing		
RSPS [16]	87.66 ± 1.69	58.33 ± 4.34	31.14 ± 3.88	2.1
DARTS [17]	54.30 ± 0.00	15.61 ± 0.00	16.32 ± 0.00	3.0
GDAS [10]	93.51 ± 0.13	70.61 ± 0.26	41.84 ± 0.90	8.0
SETN [9]	86.19 ± 4.63	56.87 ± 7.77	31.90 ± 4.07	8.6
ENAS [24]	54.30 ± 0.00	15.61 ± 0.00	16.32 ± 0.00	3.6
		Non-weight sharing		
RS [2]	93.70 ± 0.36	71.04 ± 1.07	44.57 ± 1.25	3.3
BOHB [13]	93.61 ± 0.52	70.85 ± 1.28	44.42 ± 1.49	3.3
NASWOT* [22]	93.84 ± 0.23	71.56 ± 0.78	45.67 ± 0.64	-
		Evolution		
REA [27]	93.92 ± 0.30	71.84 ± 0.99	45.54 ± 1.03	3.3
TF-MOENAS*† [7]	94.16 ± 0.22	72.75 ± 0.63	$\underline{46.61 \pm 0.46}$	2.87
MOENAS (`valacc` - FLOPs)†	94.28 ± 0.17	72.68 ± 0.71	46.50 ± 0.68	53.7
TF-MOENAS (`synflow` - FLOPs)*†	94.29 ± 0.17	73.22 ± 0.71	46.31 ± 0.40	0.7
MOENAS (`valacc` - Latency)†	94.30 ± 0.09	73.00 ± 0.32	46.35 ± 0.43	54.1
TF-MOENAS (`synflow` - Latency)*†	94.29 ± 0.14	73.17 ± 0.25	46.28 ± 0.31	1.1
MOENAS (`valacc` - #Parameters)†	94.27 ± 0.18	72.72 ± 0.69	46.31 ± 0.68	53.8
TF-MOENAS (`synflow` - #Paramters)*†	94.24 ± 0.22	72.81 ± 0.76	46.31 ± 0.32	0.8
MOENAS (`valacc` - #MACs)†	94.24 ± 0.22	72.60 ± 0.77	46.37 ± 0.74	53.8
TF-MOENAS (`synflow` - #MACs)*†	94.35 ± 0.03	73.15 ± 0.07	46.47 ± 0.00	0.8
MaOENAS†	94.27 ± 0.13	72.94 ± 0.33	46.53 ± 0.27	54.8
TF-MaOENAS*†	$\underline{94.37 \pm 0.00}$	$\underline{73.51 \pm 0.00}$	46.51 ± 0.04	2.7
Optimal	**94.37**	**73.51**	**47.31**	-

* Training-Free †Multi-Objective/Many-Objective

Table 6: Accuracy on the transfer learning task. Previous studies' results are adopted from [11, 22]. Results that are underlined indicate the best method

contains much more insightful information, which can be obtained in one run and easily projected into any lower-dimensional objective spaces for intuitive Pareto front investigations.

5 Conclusions

This paper described different multi-objective and many-objective problem formulations for NAS, i.e., MONAS and MaONAS, which can be solved by multi-objective evolutionary algorithms, such as NSGA-II. We showed that the training-free metric `synflow` can be used as a proxy metric for the network accuracy performance during NAS, without requiring any training epochs. Experimental results demonstrated the benefits of using training-free approaches, especially the many-objective TF-MaOENAS, including computational efficiency, search effectiveness and insightful decision-making capabilities. These benefits were due to the ability to obtain top-performing architectures on both direct and transfer learning tasks, and the resulting penta-objective fronts of non-dominated architectures, which provided beneficial trade-off information among the concerned objectives.

References

[1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Lukasz Dudziak, and Nicholas Donald Lane. 2021. Zero-Cost Proxies for Lightweight NAS. In *ICLR 2021*. OpenReview.net. `https://openreview.net/forum?id=0cmMMy8J5q`

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305. `https://doi.org/10.5555/2503308.2188395`

[3] Peter A. N. Bosman and Dirk Thierens. 2003. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Trans. Evol. Comput.* 7, 2 (2003), 174–188. `https://doi.org/10.1109/TEVC.2003.810761`

[4] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. 2021. Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective. In *ICLR 2021*. OpenReview.net. `https://openreview.net/forum?id=Cnon5ezMHtu`

[5] Kalyanmoy Deb. 2001. *Multi-objective optimization using evolutionary algorithms*. Wiley.

[6] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197. `https://doi.org/10.1109/4235.996017`

[7] Tu Do and Ngoc Hoang Luong. 2021. Training-Free Multi-objective Evolutionary Neural Architecture Search via Neural Tangent Kernel and Number of Linear Regions. In *ICONIP 2021 (Lecture Notes in Computer Science, Vol. 13109)*, Teddy Mantoro, Minho Lee, Media Anugerah Ayu, Kok Wai Wong, and Achmad Nizar Hidayanto (Eds.). Springer, 335–347. `https://doi.org/10.1007/978-3-030-92270-2_29`

[8] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. 2022. NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. *IEEE*

Trans. Pattern Anal. Mach. Intell. 44, 7 (2022), 3634–3646. https://doi.org/10.1109/TPAMI.2021.3054824

[9] Xuanyi Dong and Yi Yang. 2019. One-Shot Neural Architecture Search via Self-Evaluated Template Network. In *ICCV 2019*. IEEE, 3680–3689. https://doi.org/10.1109/ICCV.2019.00378

[10] Xuanyi Dong and Yi Yang. 2019. Searching for a Robust Neural Architecture in Four GPU Hours. In *CVPR 2019*. Computer Vision Foundation / IEEE, 1761–1770. https://doi.org/10.1109/CVPR.2019.00186

[11] Xuanyi Dong and Yi Yang. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR 2020*. OpenReview.net. https://openreview.net/forum?id=HJxyZkBKDr

[12] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.* 20 (2019), 55:1–55:21. http://jmlr.org/papers/v20/18-598.html

[13] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *ICML 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 1436–1445. http://proceedings.mlr.press/v80/falkner18a.html

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[15] Zhilin He. 2023. Improved Genetic Algorithm in Multi-objective Cargo Logistics Loading and Distribution. *Informatica* 47, 2 (2023). https://doi.org/10.31449/inf.v47i2.3958

[16] Liam Li and Ameet Talwalkar. 2019. Random Search and Reproducibility for Neural Architecture Search. In *UAI 2019 (Proceedings of Machine Learning Research, Vol. 115)*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 367–377. http://proceedings.mlr.press/v115/li20c.html

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *ICLR 2019*. OpenReview.net. https://openreview.net/forum?id=S1eYHoC5FX

[18] Zhichao Lu, Ran Cheng, Yaochu Jin, Kay Chen Tan, and Kalyanmoy Deb. 2022. Neural Architecture Search as Multiobjective Optimization Benchmarks: Problem Formulation and Performance Assessment. *IEEE Transactions on Evolutionary Computation* (2022). https://doi.org/10.1109/TEVC.2022.3233364

[19] Zhichao Lu, Kalyanmoy Deb, Erik D. Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. 2020. NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search. In *ECCV 2020 (Lecture Notes in Computer Science, Vol. 12346)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer, 35–51. https://doi.org/10.1007/978-3-030-58452-8_3

[20] Zhichao Lu, Ian Whalen, Yashesh D. Dhebar, Kalyanmoy Deb, Erik D. Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. 2020. NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm (Extended Abstract). In *IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, 4750–4754. https://doi.org/10.24963/ijcai.2020/659

[21] Hoang N. Luong and Peter A. N. Bosman. 2012. Elitist Archiving for Multi-Objective Evolutionary Algorithms: To Adapt or Not to Adapt. In *PPSN XII (Lecture Notes in Computer Science, Vol. 7492)*, Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone (Eds.). Springer, 72–81. https://doi.org/10.1007/978-3-642-32964-7_8

[22] Joseph Charles Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. 2020. Neural Architecture Search without Training. *CoRR* abs/2006.04647 (2020). arXiv:2006.04647 https://arxiv.org/abs/2006.04647

[23] Sarat Mishra and Sudhansu Kumar Mishra. 2020. Performance Assessment of a set of Multi-Objective Optimization Algorithms for Solution of Economic Emission Dispatch Problem. *Informatica* 44, 3 (2020), 349–360. https://doi.org/10.31449/inf.v44i3.1969

[24] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *ICML 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 4092–4101. http://proceedings.mlr.press/v80/pham18a.html

[25] Quan Minh Phan and Ngoc Hoang Luong. 2021. Efficiency Enhancement of Evolutionary Neural Architecture Search via Training-Free Initialization. In *(NICS) 2021*. 138–143. https://doi.org/10.1109/NICS54270.2021.9701573

[26] Quan Minh Phan and Ngoc Hoang Luong. 2023. Enhancing multi-objective evolutionary neural architecture search with training-free Pareto local search.

Appl. Intell. 53, 8 (2023), 8654–8672. https://doi.org/10.1007/s10489-022-04032-y

[27] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI 2019*. AAAI Press, 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780

[28] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS 2020*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/46a4378f835dc8040c8057beb6a2da52-Abstract.html

[29] An Vo, Tan Ngoc Pham, Van Bich Nguyen, and Ngoc Hoang Luong. 2022. Training-Free Multi-Objective and Many-Objective Evolutionary Neural Architecture Search with Synaptic Flow. In *SoICT 2022*. ACM, 1–8. https://doi.org/10.1145/3568562.3568569

[30] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR 2017*. OpenReview.net. https://openreview.net/forum?id=r1Ue8Hcxg