LayerNAS: Neural Architecture Search in Polynomial Complexity

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural Architecture Search (NAS) has become a popular method for discovering effective model architectures, especially for target hardware. As such, NAS methods that find optimal architectures under constraints are essential. In our paper, we propose LayerNAS to address the challenge of multi-objective NAS by transforming it into a combinatorial optimization problem, which effectively constrains the search complexity to be polynomial.

LayerNAS rigorously derives its method from the fundamental assumption that modifications to previous layers have no impact on the subsequent layers. When dealing with search spaces containing L layers that meet this requirement, the method performs layerwise-search for each layer, selecting from a set of search options $\mathbb{S}$. LayerNAS groups model candidates based on one objective, such as model size or latency, and searches for the optimal model based on another objective, thereby splitting the cost and reward elements of the search. This approach limits the search complexity to $O(H \cdot |\mathbb{S}| \cdot L)$, where H is a constant set in LayerNAS.

Our experiments show that LayerNAS is able to consistently discover superior models across a variety of search spaces in comparison to strong baselines, including search spaces derived from NATS-Bench, MobileNetV2 and MobileNetV3.

1 Introduction

With the surge of ever-growing neural models used across all ML-based disciplines, the efficiency of neural networks is becoming a fundamental factor in their success and applicability. A carefully crafted architecture can achieve good quality while maintaining efficiency during inference. However, designing optimized architectures is a complex and time-consuming process – this is especially true when multiple objectives are involved, including the model's performance and one or more cost factors reflecting the model's size, Multiply-Adds (MAdds) and inference latency. Neural Architecture Search (NAS), is a highly effective paradigm for dealing with such complexities. NAS automates the task and discovers more intricate and complex architectures than those that can be found by humans. Additionally, recent literature shows that NAS allows to search for optimal models under specific constraints (e.g., latency), with remarkable applications on architectures such as MobileNetV3 [15], EfficientNet [34] and FBNet [36].

Most NAS algorithms encode model architectures using a list of integers, where each integer represents a selected search option for the corresponding layer. In particular, notice that for a given model with L layers, where each layer is selected from a set of search options $\mathbb{S}$, the search space contains $O(|\mathbb{S}|^L)$ candidates with different architectures. This exponential complexity presents a significant efficiency challenge for NAS algorithms.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

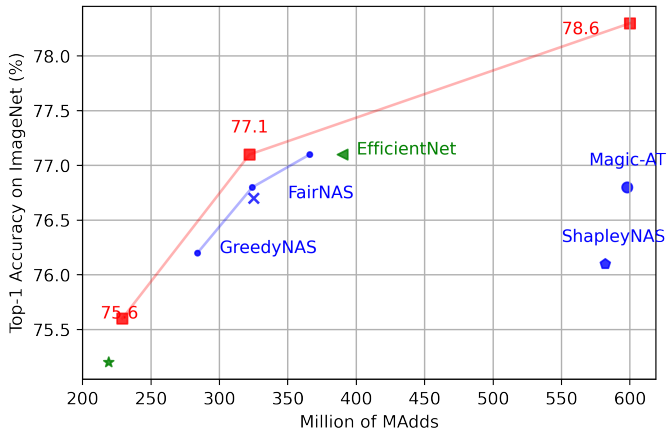

Figure 1: Comparison with baseline models and NAS methods.

In this paper, we present LayerNAS, an algorithm that addresses the problem of Neural Architecture Search (NAS) through the framework of combinatorial optimization. The proposed approach decouples the constraints of the model and the evaluation of its quality, and explores the factorized search space more effectively in a layerwise manner, reducing the search complexity from exponential to polynomial.

LayerNAS rigorously derives the method from the fundamental assumption: optimal models when searching for $layer_i$ can be constructed from one of the models in $layer_{i-1}$. For search spaces that satisfy this assumption, LayerNAS enforces a directional search process from the first layer to the last layer. The directional layerwise search makes the search complexity $O(C \cdot |\mathbb{S}| \cdot L)$, where C is the number of candidates to search per layer.

For multi-objective NAS problems, LayerNAS treats model constraints and model quality as separate metrics. Rather than utilizing a single objective function that combines multi-objectives, LayerNAS stores model candidates by their constraint metric value. Let $\mathcal{M}_{i,h}$ be the best model candidate for $layer_i$ with cost $= h$. LayerNAS searches for optimal models under different constraints in the next layer by adding the cost of the selected search option for next layer to the current layer, i.e., $\mathcal{M}_{i,h}$. This transforms the problem into the following combinatorial optimization problem: *for a model with L layers, what is the optimal combination of options for all layers needed to achieve the best quality under the cost constraint?* If we bucketize the potential model candidates by their cost, the search space is limited to $O(H \cdot |\mathbb{S}| \cdot L)$, where H is number of buckets per layer. In practice, capping the search at 100 buckets achieves reasonable performance. Since this holds H constant, it makes the search complexity polynomial.

Our contributions can be summarized as follows:

- We propose LayerNAS, an algorithm that transforms the multi-objective NAS problem to a Combinatorial Optimization problem. This is a novel formulation of NAS.
- LayerNAS is directly designed to tackle the search complexity of NAS, and reduce the search complexity from $O(|\mathbb{S}|^L)$ to $O(H \cdot |\mathbb{S}| \cdot L)$, where H is a constant defined in the algorithm.
- We demonstrate the effectiveness of LayerNAS by identifying high-performing model architectures under various Multiply-Adds (MAdds) constraints, by searching through search spaces derived from MobileNetV2 [30] and MobileNetV3 [15].

2 Related Work

The survey by [12] categorizes methods for Neural Architecture Search into three dimensions: search space, search strategy, and performance estimation strategy. The formulation of NAS as different

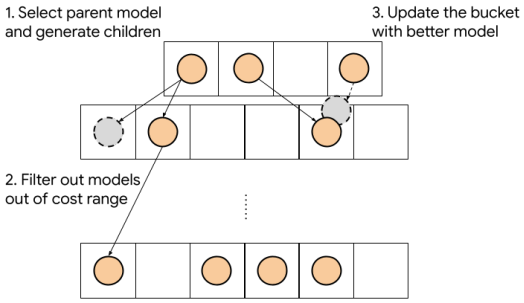

Figure 2: Illustration of the LayerNAS Algorithm described in Algorithm 1. For each layer: (1) select a model candidate from current layer and generate children candidates; (2) filter out candidates not in the target objective range; (3) update the model in the bucket if there's a candidate with better quality; and finally, move to the next layer.

problems has led to the development of a diverse array of search algorithms. Bayesian Optimization is first adopted for hyper-parameter tuning [3, 9, 13, 20]. Reinforcement Learning is utilized for training an agent to interact with a search space [45, 27, 46, 19]. Evolutionary algorithms [24, 29] have been employed by encoding model architectures to DNA and evolving the candidate pool. ProgressiveNAS [22] uses heuristic search to gradually build models by starting from simple and shallow model architectures and incrementally adding more operations to arrive at deep and complex final architectures. This is in contrast to LayerNAS, which iterates over changes in the layers of a full complex model.

Recent advancements in mobile image models, such as MobileNetV3 [15], EfficientNet [34], FBNet [36], are optimized by NAS. The search for these models is often constrained by metrics such as FLOPs, model size, latency, and others. To solve this multi-objective problem, most NAS algorithms [33, 5] design an objective function that combines these metrics into a single objective. LEMONADE [11] proposes a method to split two metrics, and searches for a Pareto front of a family of models. Once-for-All [4] proposes progressive shrinking algorithm to efficiently find optimal model architectures under different constraints.

Larger models tend to have better performance compared to smaller models. However, the increased size of models also means increased computational resource requirement. As a result, the optimization of neural architectures within constrained resources is an important and meaningful aspect of NAS problems, which can be solved as multi-objective optimization [16]. There is increasing interest in treating NAS as a compression problem [43, 42] from an over-sized model. These works indicate that compressing with different configurations on each layer leads to a model better than uniform compression. Here, NAS can be used to search for optimal configurations [14, 25, 35].

The applicability of NAS is significantly influenced by the efficiency of its search process. One-shot algorithms [23, 5, 1, 2] provide a novel approach by constructing a supernet from the search space to perform more efficient NAS. However, this approach has limit on number of branches in supernet due to the constraints of supernet size. The search cost is not only bounded by the complexity of search space, but also the cost of training under "train-and-eval" paradigm. Training-free NAS [26, 6, 44, 32] breaks this paradigm by estimating the model quality with other metrics that are fast to compute. However, the search quality heavily relies on the effectiveness of the metrics.

3 Problem Definition

Most NAS algorithms do not differentiate the various types of NAS problems. Rather, they employ a single encoding of the search space with a general solution for the search process. However, the unique characteristics of NAS problems can be leveraged to design a tailored approach. We categorize NAS problems into three major types:

- Topology search: the search space defines a graph with multiple nodes. The objective is to identify an optimal topology for connecting nodes with different operations. This task allows for the exploration of novel architectures.
- Size search or compression search: the search occurs on a predefined model architecture with multiple layers. Each layer can be selected from as a set of search options. Empirically, the best-performing model is normally the one with the most parameters per layer. Therefore, in practice, we aim to search for the optimal model under certain constraints. NATSBench size search [10] provides a public dataset for this type of task. MobilNetV3 [15], EfficientNet [34], FBNet [36] also establish the search space in this manner. This problem can also be viewed as a compression problem [14], as reducing the layer size serves as a means of compression by decreasing the model size, FLOPs and latency.
- Scale search: model architectures are uniformly configured by hyper-parameters, such as number of layers or size of fully-connected layers. This task views the model as a holistic entity and uniformly scales it up or down, rather than adjusting individual layers or components.

This taxonomy illustrates the significant variation among NAS problems. Rather than proposing a general solution to address all of them, we propose to tackle with search spaces in a layerwise manner. Specifically, we aim to find a model with L layers. For each $layer_i$, we select from a set of search options $\mathbb{S}_i$. A model candidate $\mathcal{M}$ can be represented as a tuple with size L: $(s_1, s_2, ..., s_L)$. $s_i \in \mathbb{S}_i$ is a selected search option on $layer_i$. The objective is to find an optimal model architecture $\mathcal{M} = (s_1, s_2, ..., s_L)$ with the highest accuracy:

$$\underset{(s_1,s_2,...,s_L)}{\operatorname{argmax}} \; Accuracy(\mathcal{M}) \tag{1}$$

4 Method

We propose LayerNAS as an algorithm that leverages layerwise attributes. When searching models $\mathcal{M}_i$ on $layer_i$, we are searching for architectures in the form of $(s_{1..i-1}, x_i, o_{i+1..L})$. $s_{1..i-1}$ are the selected options for $layer_{1..i-1}$, and $o_{i+1..L}$ are the default, predefined options. x_i is the search option selected for $layer_i$, which is the current layer in the search. In this formulation, only $layer_i$ can be changed, all preceding layers are fixed, and all succeeding layers are using the default option. In topology search, the default option is usually no-op; in size search, the default option can be the option with most computation.

LayerNAS operates on a search space that meets the following assumption, which has been implicitly utilized by past chain-structured NAS techniques [22, 33, 15].

Assumption 4.1. The optimal model $\mathcal{M}_i$ on $layer_i$ can be constructed from a model $m \in \mathbb{M}_{i-1}$, where $\mathbb{M}_{i-1}$ is a set of model candidates on $layer_{i-1}$.

This assumption implies:

- Enforcing a sequential search process is possible when exploring $layer_i$ because improvements to the model cannot be achieved by modifying $layer_{i-1}$.
- The information for finding an optimal model architecture on $layer_i$ was collected when searching for model architectures on $layer_{i-1}$.
- Search spaces that are constructed in a layerwise manner, such as those in size search problems discussed in Section 3, can usually meet this assumption. Each search option can completely define how to construct a succeeding layer, and does not depend on the search options in previous layers.
- It's worth noting that not all search spaces can meet the assumption. Succeeding layers may be coupled with or affect preceding layers in some cases. In practice, we transform the search space in Section 5 to ensure that it meets the assumption.

4.1 LayerNAS for Topology Search

The LayerNAS algorithm is described by the pseudo code in Algorithm 1. $\mathbb{M}_l$ is a set of model candidates on $layer_l$. $\mathbb{M}_{l,h}$ is the model on $layer_l$ mapped to $h \in \mathbb{H}$, a lower dimensional representation. $\mathbb{H}$ is usually a finite integer set, so that we can index and store models.

Algorithm 1 LayerNAS algorithm

Inputs: L (num layers), R (num searches per layer), T (num models to generate in next layer)
l = 1
$\mathbb{M}_1 = \{\forall \mathcal{M}_1\}$
repeat
  for i = 1 to R **do**
    $\mathcal{M}_l$ = select($\mathbb{M}_l$)
    for j = 1 to T **do**
      $\mathcal{M}_{l+1}$ = apply_search_option($\mathcal{M}_l$, $\mathbb{S}_{l+1}$)
      $h = \varphi(\mathcal{M}_{l+1})$
      $accuracy$ = train_and_eval($\mathcal{M}_{l+1}$)
      if $accuracy >$ Accuracy($\mathbb{M}_{l+1,h}$) **then**
        $\mathbb{M}_{l+1,h} = \mathcal{M}_{l+1}$
      end if
    end for
  end for
  $l = l + 1$
  if $l == L$ **then**
    l = 1
  end if
until no available candidates

Some functions can be customized for different NAS problems with a-priori knowledge:

- select: samples a model from a set of model candidates in the previous layer $\mathbb{M}_l$. It could be a mechanism of evolutionary algorithm [29] or a trained predictor [22] to select most promising candidates. The method will also filter out model architectures that we do not need to search, usually a model architecture that is invalid or known not to generate better candidates. This can significantly reduce the number of candidates to search.

- apply_search_option: applies a search option from $\mathbb{S}_{l+1}$ on $\mathcal{M}_l$ to generate $\mathcal{M}_{l+1}$. We currently use random selection in the implementation, though, other methods could lead to improved search option.

- $\varphi : \mathbb{M} \rightarrow \mathbb{H}$ maps model architecture in $\mathbb{M}$ to a lower dimensional representation $\mathbb{H}$. φ could be an encoded index of model architecture, or other identifiers that group similar model architectures. We discuss this further in 4.2. When there is a unique id for each model, LayerNAS will store all model candidates in $\mathbb{M}$.

In this algorithm, the total number of model candidates we need to search is $\sum_{i=1}^{L} |\mathbb{M}_i| \cdot |\mathbb{S}_i|$. It has a polynomial form, but $|\mathbb{M}_L| = O(|\mathbb{S}|^L)$ if we set $\varphi(\mathcal{M})$ as unique id of models. This does not limit the order of $|\mathbb{M}_L|$ to search. For topology search, we can design a sophisticated φ to group similar model candidates. In the following discussion, we will demonstrate how to lower the order of $|\mathbb{M}_L|$ in multi-objective NAS.

4.2 LayerNAS for Multi-objective NAS

LayerNAS is aimed at designing an efficient algorithm for size search or compression search problems. As discussed in Section 3, such problems satisfy Assumption 4.1 by nature. Multi-objective NAS usually searches for an optimal model under some constraints, such as model size, inference latency, hardware-specific FLOPs or energy consumption. We use "cost" as a general term to refer to these constraints. These "cost" metrics are easy to calculate and can be determined when the model architecture is fixed. This is in contrast to calculating accuracy, which requires completing the model training. Because the model is constructed in a layer-wise manner, the cost of the model can be estimated by summing the costs of all layers.

Hence, we can express the multi-objective NAS problem as,

$$\begin{aligned} \underset{(s_1,s_2,\ldots,s_L)}{\text{argmax}} \quad & Accuracy(\mathcal{M}_L) \\ \text{s.t.} \quad & \sum_{i=1}^{L} Cost(s_i) \leq \text{target} \end{aligned} \tag{2}$$

where $Cost(s_i)$ is the cost of applying option s_i on $layer_i$.

We introduce an additional assumption by considering the cost in Assumption 4.1:

Assumption 4.2. The optimal model $\mathcal{M}_i$ with cost $= C$ when searching for $layer_i$ can be constructed from the optimal model $\mathcal{M}_{i-1}$ with cost $= C - Cost(s_i)$ from $\mathbb{M}_{i-1}$.

In this assumption, we only keep one optimal model out of a set of models with similar costs. Suppose we have two models with the same cost, but $\mathcal{M}_i$ has better quality than $\mathcal{M}'_i$. The assumption will be satisfied if any changes on following layers to $\mathcal{M}_i$ will generate a better model than making the same change to $\mathcal{M}'_i$.

By applying Assumption 4.2 to Equation (2), we can formulate the problem as combinatorial optimization:

$$\begin{aligned} \underset{x_i}{\text{argmax}} \quad & Accuracy(\mathcal{M}_i) \\ \text{s.t.} \quad & \sum_{j=1}^{i-1} Cost(s_{1..i-1}, x_i, o_{i+1,L}) \leq \text{target} \\ \text{where} \quad & \mathcal{M}_i = (s_{1..i-1}, x_i, o_{i+1..L}), \mathcal{M}_{i-1} = (s_{1..i-1}, o_{i..L}) \in \mathbb{M}_{i-1,h'} \end{aligned} \tag{3}$$

This formulation decouples cost from reward, so there is no need to manually design an objective function to combine these metrics into a single value, and we can avoid tuning hyper-parameters of such an objective. Formulating the problem as combinatorial optimization allows solving it efficiently using dynamic programming. $\mathbb{M}_{l,h}$ can be considered as a memorial table to record best models on $layer_l$ at cost h. For $layer_l$, $\mathcal{M}_l$ generates the $\mathcal{M}_{l+1}$ by applying different options selected from $\mathbb{S}_{l+1}$ on $layer_{l+1}$. The search complexity is $O(H \cdot |\mathbb{S}| \cdot L)$.

We do not need to store all $\mathbb{M}_{l,h}$ candidates, but rather group them with the following transformation:

$$\varphi(\mathcal{M}_i) = \left\lfloor \frac{Cost(\mathcal{M}_i) - \min Cost(\mathbb{M}_i)}{\max Cost(\mathbb{M}_i) - \min Cost(\mathbb{M}_i)} \times H \right\rfloor \tag{4}$$

where H is the desired number of buckets to keep. Each bucket contains model candidates with costs in a specific range. In practice, we can set $H = 100$, meaning we store optimal model candidates within 1% of the cost range.

Equation (4) limits $|\mathbb{M}_i|$ to be a constant value since H is a constant. $\min Cost(\mathbb{M}_i)$ and $\max Cost(\mathbb{M}_i)$ can be easily calculated when we know how to select the search option from $\mathbb{S}_{i+1}..\mathbb{S}_L$ in order to maximize or minimize the model cost. This can be achieved by defining the order within $\mathbb{S}_i$. Let $s_i = 1$ represent the option with the maximal cost on $layer_i$, and $s_i = |\mathbb{S}|$ represent the option with the minimal cost on $layer_i$. This approach for constructing the search space facilitates an efficient calculation of maximal and minimal costs.

The optimization applied above leads to achieving polynomial search complexity $O(H \cdot |\mathbb{S}| \cdot L)$. $O(|\mathbb{M}|) = H$ is upper bound of the number of model candidates in each layer, and becomes a constant after applying Equation (4). $|\mathbb{S}|$ is the number of search options on each layer.

LayerNAS for Multi-objective NAS does not change the implementation of Algorithm 1. Instead, we configure methods to perform dynamic programming with the same framework:

- φ: groups $\mathbb{M}_i$ by their costs with Equation (4)
- select: filters out $\mathcal{M}_l$ if all $\mathcal{M}_{l+1}$ constructed from it are out of the range of target cost. This significantly reduces the number of candidates to search.

- In practice, Assumption 4.2 is not always true because accuracy may vary in each training trial. The algorithm may store a lucky model candidate that happens to get a better accuracy due to variation. We store multiple candidates for each h to reduce the problem from training accuracy variation.

5 Experiments

5.1 Search on ImageNet

Search Space: we construct several search spaces based on MobileNetV2, MobileNetV2 (width multiplier=1.4), MobileNetV3-Small and MobileNetV3-Large. For each search space, we set similar backbone of the base model. For each layer, we consider kernel sizes from {3, 5, 7}, base filters and expanded filters from a set of integers, and a fixed strides. The objective is to find better models with similar MAdds of the base model.

To avoid coupling between preceding and succeeding layers, we first search the shared base filters in each block to create residual shortcuts, and search for kernel sizes and expanded filters subsequently. This ensures the search space satisfy Assumption 4.1.

We estimate and compare the number of unique model candidates defined by the search space and the maximal number of searches in Table 1. In the experiments, we set $H = 100$, and store 3 best models with same h-value. Note that the maximal number of searches does not mean actual searches conducted in the experiments, but rather an upper bound defined by the algorithm.

A comprehensive description of the search spaces and discovered model architectures in this experiment can be found in the Appendix for further reference.

Table 1: Comparison of model candidates in the search spaces

Search Space	Target MAdds	# Unique Models	# Max Trials
MobileNetV3-Small	60M	$5.0e+20$	$1.2e+5$
MobileNetV3-Large	220M	$4.8e+26$	$1.5e+5$
MobileNetV2	300M	$5.3e+30$	$1.4e+5$
MobileNetV2 1.4x	600M	$1.6e+39$	$2.0e+6$

Search, train and evaluation:

During the search process, we train the model candidates for 5 epochs, and use the top-1 accuracy on ImageNet as a proxy metrics. Following the search process, we select several model architectures with best accuracy on 5 epochs, train and evaluate them on 4x4 TPU with 4096 batch size (128 images per core). We use RMSPropOptimizer with 0.9 momentum, train for 500 epochs. Initial learning rate is 2.64, with 12.5 warmup epochs, then decay with cosine schedule.

Results

We list the best models discovered by LayerNAS, and compare them with baseline models and results from recent NAS works in Table 2. For all targeted MAdds, the models discovered by LayerNAS achieve better performance: 69.0% top-1 accuracy on ImageNet for 61M MAdds, a 1.6% improvement over MobileNetV3-Small; 75.6% for 229M MAdds, a 0.4% improvement over MobileNetV3-Large; 77.1% accuracy for 322M MAdds, a 5.1% improvement over MobileNetV2; and finally, 78.6% accuracy for 627M MAdds, a 3.9% improvement over MobileNetV2 1.4x.

Note that for all of these models, we include squeeze-and-excitation blocks [17] and use Swish activation [28], in order to to achieve the best performance. Some recent works on NAS algorithms, as well as the original MobileNetV2, do not use these techniques. For a fair comparison, we also list the model performance after removing squeeze-and-excitation and replacing Swish activation with ReLU. The results show that the relative improvement from LayerNAS is present even after removing these components.

Table 2: Comparison of models on ImageNet

Model	Top1 Acc.	Params	MAdds
MobileNetV3-Small† [15]	67.4	2.5M	56M
MNasSmall [33]	64.9	1.9M	65M
LayerNAS (Ours)†	**69.0**	3.7M	61M
MobileNetV3-Large† [15]	75.2	5.4M	219M
LayerNAS (Ours) †	**75.6**	5.1M	229M
MobileNetV2⋆ [30]	72.0	3.5M	300M
ProxylessNas-mobile⋆ [5]	74.6	4.1M	320M
MNasNet-A1 [33]	75.2	3.9M	315M
FairNAS-C⋆ [8]	74.7	5.6M	325M
LayerNAS-no-SE(Ours)⋆	75.5	3.5M	319M
EfficientNet-B0 [34]	77.1	5.3M	390M
SGNAS-B [18]	76.8	-	326M
FairNAS-C† [8]	76.7	5.6M	325M
GreedyNAS-B† [41]	76.8	5.2M	324M
LayerNAS (Ours)†	**77.1**	5.2M	322M
MobileNetV2 1.4x⋆ [30]	74.7	6.9M	585M
ProgressiveNAS⋆ [22]	74.2	5.1M	588M
Shapley-NAS⋆ [37]	76.1	5.4M	582M
MAGIC-AT⋆ [38]	76.8	6M	598M
LayerNAS-no-SE (Ours)⋆	77.1	7.6M	598M
LayerNAS (Ours) †	**78.6**	9.7M	627M

⋆ Without squeeze-and-excitation blocks.
† With squeeze-and-excitation blocks.

5.2 NATS-Bench

The following experiments compare LayerNAS with others NAS algorithms on NATS-Bench [10]. We evaluate NAS algorithms from these three perspectives:

- Candidate quality: the quality of the best candidate found by the algorithm, as can be indicated by the peak value in the chart.
- Stability: the ability to find the best candidate, after running multiple searches and analyzing the average value and range of variation.
- Efficiency: The training time required to find the best candidate. The sooner the peak accuracy candidate is reached, the more efficient the algorithm.

NATS-Bench topology search

NATS-Bench topology search defines a search space on 6 ops that connect 4 tensors, each op has 5 options (conv1x1, conv3x3, maxpool3x3, no-op, skip). It contains 15625 candidates with their number of parameters, FLOPs, accuracy on Cifar-10, Cifar-100 [21], ImageNet16-120 [7].

In Table 3, we compare with recent state-of-the-art methods. Although training-free NAS has advantage of lower search cost, LayerNAS can achieve much better results.

NATS-Bench size search

NATS-Bench size search defines a search space on a 5-layer CNN model, each layer has 8 options on different number of channels, from 8 to 64. The search space contains 32768 model candidates. The one with the highest accuracy has 64 channels for all layers, we can refer this candidate as "the largest model". Instead of searching for the best model, we set the goal to search for the optimal model with 50% FLOPs of the largest model.

Under this constraints for size search, we implement popular NAS algorithms for comparison, which are also used in the original benchmark papers [40, 10]: random search, proximal policy optimization (PPO) [31] and regularized evolution (RE) [29]. We conduct 5 runs for each algorithm, and record the best accuracy at different training costs.

LayerNAS treats this as a compression problem. The base model, which is the largest model, has 64 channels on all layers. By applying search options with fewer channels, the model becomes smaller,

faster and less accurate. The search process is to find the optimal model with expected FLOPs. By filtering out candidates that do not produce architectures falling within the expected FLOPs range, we can significantly reduce the number of candidates that need to be searched.

Table 3: Comparison on NATS-Bench topology search. Mean and deviation of test accuracy on 5 runs.

	Cifar10	Cifar100	ImageNet16-120	Search cost (sec)
RS	92.39±0.06	63.54±0.24	42.71±0.34	1e+5
RE [29]	94.13±0.18	71.40±0.50	44.76±0.64	1e+5
PPO [31]	94.02±0.13	71.68±0.65	44.95±0.52	1e+5
KNAS [39]	93.05	68.91	34.11	4200
TE-NAS [6]	93.90±0.47	71.24±0.56	42.38±0.46	1558
EigenNas [44]	93.46±0.02	71.42±0.63	45.54±0.04	-
NASI [32]	93.55±0.10	71.20±0.14	44.84±1.41	120
FairNAS [8]	93.23±0.18	71.00±1.46	42.19±0.31	1e+5
SGNAS [18]	93.53±0.12	70.31±1.09	44.98±2.10	9e+4
LayerNAS	**94.34±0.12**	**73.01±0.63**	**46.58±0.59**	1e+5
Optimal test accuracy	94.37	73.51	47.31	

Table 4: Comparison on NATS-Bench size search. Average on 5 runs.

	Cifar10		Cifar100		ImageNet16-120	
Training time (sec)	2e+5		4e+5		6e+5	
Target mFLOPs	140		140		35	
	Validation	Test	Validation	Test	Validation	Test
RS	0.8399	0.9265	0.5947	0.6935	0.3638	0.4381
RE [29]	**0.8440**	0.9282	0.6057	0.6962	0.3770	0.4476
PPO [31]	0.8432	0.9283	0.6033	0.6957	0.3723	0.4438
LayerNAS	**0.8440**	**0.9320**	**0.6067**	**0.7064**	**0.3812**	**0.4537**
Optimal validation	0.8452	0.9264	0.6060	0.6922	0.3843	0.4500
Optimal test	0.8356	0.9334	0.5870	0.7086	0.3530	0.4553

6 Conclusion and Future Work

In this research, we propose LayerNAS that formulates Multi-objective Neural Architecture Search to Combinatorial Optimization. By decoupling multi-objectives into cost and accuracy, and leverages layerwise attributes, we are able to reduce the search complexity from $O(|\mathbb{S}|^L)$ to $O(H \cdot |\mathbb{S}| \cdot L)$.

Our experiment results demonstrate the effectiveness of LayerNAS in discovering models that achieve superior performance compared to both baseline models and models discovered by other NAS algorithms under various constraints of MAdds. Specifically, models discovered through LayerNAS achieve top-1 accuracy on ImageNet of 69% for 61M MAdds, 75.6% for 229M MAdds, 77.1% for 322M MAdds, 78.6% for 627M MAdds. Furthermore, our analysis reveals that LayerNAS outperforms other NAS algorithms on NATS-Bench in all aspects including best model quality, stability and efficiency.

While the current implementation of LayerNAS has shown promising results, several current limitations that can be addressed by future work:

- LayerNAS is not designed to solve scale search problems mentioned in Section 3, because many hyper-parameters of model architecture are interdependent in scale search problem, which contradicts the statement in Assumption 4.1.
- One-shot NAS algorithms have been shown to be more efficient. We aim to investigate the potential of applying LayerNAS to One-shot NAS algorithms.

References

[1] Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR.

[2] Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.-J., and Le, Q. V. (2020). Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14323–14332.

[3] Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR.

[4] Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (2020). Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

[5] Cai, H., Zhu, L., and Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*.

[6] Chen, W., Gong, X., and Wang, Z. (2021). Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*.

[7] Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

[8] Chu, X., Zhang, B., and Xu, R. (2021). Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12239–12248.

[9] Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*.

[10] Dong, X., Liu, L., Musial, K., and Gabrys, B. (2021). Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE transactions on pattern analysis and machine intelligence*.

[11] Elsken, T., Metzen, J. H., and Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*.

[12] Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017.

[13] Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR.

[14] He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. (2018). Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800.

[15] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324.

[16] Hsu, C.-H., Chang, S.-H., Liang, J.-H., Chou, H.-P., Liu, C.-H., Chang, S.-C., Pan, J.-Y., Chen, Y.-T., Wei, W., and Juan, D.-C. (2018). Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332*.

[17] Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

[18] Huang, S.-Y. and Chu, W.-T. (2021). Searching by generating: Flexible and efficient one-shot nas with architecture generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 983–992.

[19] Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. (2019). Reinforcement learning for neural architecture search: A review. *Image and Vision Computing*, 89:57–66.

[20] Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., and Xing, E. P. (2018). Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31.

[21] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

[22] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018a). Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34.

[23] Liu, H., Simonyan, K., and Yang, Y. (2018b). Darts: Differentiable architecture search. In *International Conference on Learning Representations*.

[24] Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Tan, K. C. (2021). A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*.

[25] Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., and Sun, J. (2019). Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305.

[26] Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2021). Neural architecture search without training. In *International Conference on Machine Learning*, pages 7588–7598. PMLR.

[27] Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR.

[28] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

[29] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

[30] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

[31] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[32] Shu, Y., Cai, S., Dai, Z., Ooi, B. C., and Low, B. K. H. (2021). Nasi: Label-and data-agnostic neural architecture search at initialization. *arXiv preprint arXiv:2109.00817*.

[33] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828.

[34] Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

[35] Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. (2019). Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620.

[36] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742.

[37] Xiao, H., Wang, Z., Zhu, Z., Zhou, J., and Lu, J. (2022). Shapley-nas: Discovering operation contribution for neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11892–11901.

[38] Xu, J., Tan, X., Song, K., Luo, R., Leng, Y., Qin, T., Liu, T.-Y., and Li, J. (2022). Analyzing and mitigating interference in neural architecture search. In *International Conference on Machine Learning*, pages 24646–24662. PMLR.

[39] Xu, J., Zhao, L., Lin, J., Gao, R., Sun, X., and Yang, H. (2021). Knas: green neural architecture search. In *International Conference on Machine Learning*, pages 11613–11625. PMLR.

[40] Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR.

[41] You, S., Huang, T., Yang, M., Wang, F., Qian, C., and Zhang, C. (2020). Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008.

[42] Yu, J. and Huang, T. (2019). Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*.

[43] Zhou, H., Yang, M., Wang, J., and Pan, W. (2019). Bayesnas: A bayesian approach for neural architecture search. In *International conference on machine learning*, pages 7603–7613. PMLR.

[44] Zhu, Z., Liu, F., Chrysos, G. G., and Cevher, V. (2022). Generalization properties of nas under activation and skip connection search. *arXiv preprint arXiv:2209.07238*.

[45] Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

[46] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710. IEEE Computer Society.