HaNAS: Rapid Hardware-Adaptive Neural Architecture Search with Meta-Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Hardware-aware neural architecture search has been successfully designed neural architecture conditioned by resource constraints (e.g. inference latency, energy) for efficient deployment of a target device. They take a task-specific approach that collects a large number of samples for metric and builds metric estimation, for a target device. However, such approach is prohibitive in real-world scenarios with billions of devices as each process is time-consuming. In this work, we propose a novel sample-efficient Hardware-adaptive NAS, HaNAS, which formulates the search process of multiple devices as a meta-learning problem, such that rapid search is possible for unseen devices with few-shot samples for metric. We introduce hardware embeddings to support multiple devices with a single HaNAS regardless of differences in hardware specifications or physical structures. Our model learns meta-knowledge from already obtained devices to generalize on unseen devices and we design hardware-adaptive modulator of meta-knowledge depending on relatedness of unseen devices with obatined device pool. In the experiments, we show that the proposed model outperforms other baseline model in sample efficiency on diverse devices of extended HW-NAS-Bench Dataset (NAS-Bench-201 and FBNet) and ...

1 Introduction

Neural Architecture Search (NAS) which aims to automate neural architecture design process have been shown a great success by overcoming sub-optimality of hand-crafted approaches [43, 27, 23]. By emerging the era of billions of devices, each device has different resource constraints (e.g. inference latency, energy consumption, memory footprint) and a neural architecture should be meet such constraints in real-world deployment scenarios. Thus, designing an efficient neural architecture suitable for each target device is one of the most important problems in NAS field. Hardware-aware neural architecture search approaches have been tackled such goal, by incorporating hardware efficiency constraints (mostly latency) into the search process [31, 4, 38, 36, 7, 2, 40, 5, 37].

However, one of the main challenges is that collecting training samples (e.g. architecture-latency pairs on each target device) to build reliable metric prediction models is too time-consuming and barrier for non-hardware experts. Such existing approaches require a large number of samples (e.g. 5K) and train metric predictors from scratch independently for each device. Additionally, most works [31, 4, 38, 36, 7] are task-specific method that repeat the whole NAS process for each case, which is computationally expensive, thus, is prohibitive for real-world deployment scenarios that consider a *huge* number of hardware devices. OFA [5] and HAT [37] alleviate this by proposing high-performance supernet trained on a dataset to provide accuracy of architectures by supporting multiple devices, yet they are sub-optimal since building device-specific predictors for metric except accuracy is still needed with a large number of sample collection and training from scratch (+ obtaining a

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

large number of predictors to cover multiple devices) for each device. Additionally, application of a layer-wise predictor [5] is limited due to under-performing on graph-based search space [10]. Besides, BigNAS [41] considers FLOP as hardware-cost which is an inaccurate proxy.

To overcome such limitations, we propose a novel sample-efficient Hardware-Adaptive Neural Architecture Search, HANAS, that supports multiple devices with a **single** NAS model by **fast adapting** to new target device (unseen during training phase) with only **few-shot** training samples. We design a meta-learning framework for hardware-adaptive NAS that transfers knowledge learned from the already obtained devices and their training samples to a new device, by learning generalization well over the source device-data distribution, rather than a single task. This allows us to achieve high sample efficiency for accurate metric estimation for unseen devices. In other words, we reduce the number of sample architectures required to train the metric estimator by at least XX times on a large number of downstream devices compared with existing NAS methods, by utilizing accumulated meta-knowledge. To this end, we formulate this problem as a few-shot regression task where each task consists of {hardware, a few-shot neural architecture, a set of latency} and the proposed model learns to generalize hardware-dependent latency estimation from multiple tasks.

Further, this provide an important contribution in meta-learning domain, such that we tackle realistic problem with meta-learning while conventional meta-learning methods [35, 30, 13, 26, 19, 14] are limited in rather artificial setting such as regression of sine wave or few-shot image classification tasks. HANAS is a general model in terms of both hardware platforms and search space. HANAS can model diverse hardware platforms into standardized latent vectors without any knowledge of the physical structure or specifications. While layer-wise predictor is under-performing on non-block-based search space [10], HANAS shows consistently outperforming performances on 1) block-based search space 2) graph-based search space 3) NLP search space.

We first validate the proposed model on NAS-Bench-201 space with various devices from different hardware platform which we collected. Further, we utilize latency dataset in [20] for extensive device pool. In this experiment, our meta-learned predictor on meta-training device pool successfully generalize to require only 100 magnitude orders of less architecture samples on various downstream unseen target devices compared with baseline latency estimator. Further, we combine the proposed predictor with NAS method [5, 37] and shows generally works well on various tasks to search for CNN-based and transformer-based neural architecture for vision and NLP tasks, respectively.

In sum, the main contributions of this paper are as follows:

- We consider a novel problem of NAS that is a sample-efficient hardware-cost estimation over diverse hardware devices, by formulating it as a few-shot regression problem for multiple devices.
- To this end, we propose a novel Meta-NAS framework that encodes hardware identifiers and modulates meta-knowledge for unseen devices. Our meta-learned model accurately predicts hardware-cost (e.g. latency) on an unseen device by fast adapting with a few collected sample data.
- Our meta-learned predictor successfully generalizes on multiple unseen devices outperforming all other baseline models in a prediction problem and shows that 10 samples are enough practically with our latency prediction model for search process achieving X $\times$ faster than base NAS models in wall-clock building time.

2 Related Work

Hardware-aware neural architecture search By automating neural architecture design process, Neural Architecture Search (NAS) aim to optimize neural architecture for a given task. Since earlier NAS approaches [44, 45] require at least thousands of GPU days, many NAS approaches such as one-shot NAS [27, 3], differentiable NAS [23, 8, 6, 39, 12], and surrogate scheme-based NAS [25, 24, 42, 32, 1] have been proposed to reduce such huge computational cost. Recently, hardware-aware NAS [31, 4, 5, 38, 36, 15, 37, 28], which aims to design neural architectures that achieves a good trade-off between accuracy and latency for efficient deployment on target devices, is an active topic of research. MNasNet [31], ProxylessNAS [4], and FBNet [38] incorporate latency into main objective to consider both latency and accuracy, HAT [37] designs architecture of transformer [34] for Natural Language Processing (NLP) tasks by guiding latency-predictor. While

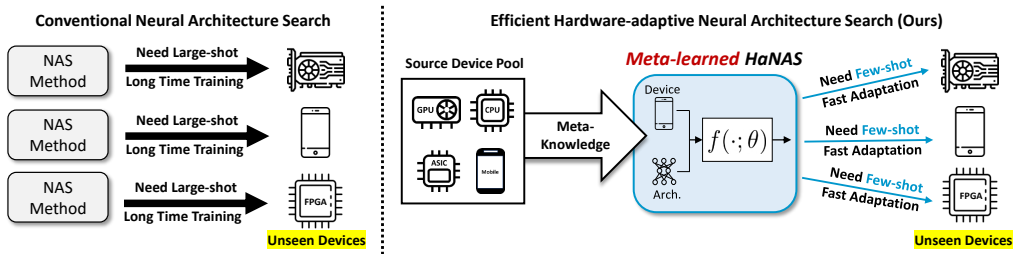

Figure 1: **Left** Conventional NAS methods require time-consuming a large number of architecture-latency pairs collection process and train NAS model independently for each device, which is inefficient. **Right** The proposed HANAS supports multiple unseen devices with a single NAS model by fast adapting to each device with a few-shot pair from meta-knowledge accumulated from the source device pool.

ChamNet [7] introduces Bayesian Optimization for sampling efficiency, it is limitedly applied to accuracy predictor and LUT is still used to estimate latency of architectures. Most of hardware-aware NAS approaches have been rely on latency look up table (LUT) [31, 4, 38, 36, 7, 5], or latency predictor [37] to estimate latency of architecture candidates during search process. Although to build LUT or latency predictor requires quite heavy cost since we port and measure at least thousands of architectures on target devices, such building costs have been overlooked by conventional NAS methods. The proposed meta-predictor can be combined with existing NAS methods by replacing conventional LUT and latency predictor and attains sample efficiency by at least 10 orders of magnitude less computation regardless of the number of target devices after one-time building cost.

Meta-learing Meta-learning (learning to learn) [33] aims to learn a model generalized over multiple tasks rather than a single task, such that the model meta-learned across the task distribution can be applied fast to unseen task. While conventional meta-learning methods [35, 30, 13, 26, 19, 14] have been meta-learned initial parameters to adapt rapidly on a novel few-shot classification task, recently, Meta-NAS approaches which combine meta-learning to neural architecture search for reducing computation requirements have been actively explored by many researchers. Gradient-based Meta-NAS methods [11, 22, 29] learn both initial parameters and operator-weighting parameters of differential NAS over multiple few-shot classification tasks to search for architecture and parameters through a few gradient steps on novel tasks. NAS with amortized meta-learning [18], which utilizes a set encoder to perform learning at the task-level, considering each task as a data instance and minimizing loss over it, cut down search time for a novel task within just a few second by utilizing meta-knowledge learned across multiple tasks of model zoo. Inspired by [18], we introduce amortized meta-learning to learn the proposed meta-predictor for device-aware neural architecture search. However, note that while [18] learns a cross-modal latent space between datasets and architectures to search for a simply high-performing architectures, the proposed method reduces the number of required latency samples measured on target devices by learning meta-knowledge from fluent pool of various type of hardware devices and adapt it to novel task with target devices, which is more challenging.

3 Method

Our goal is to design a metric prediction model that utilizes already obtained resources to rapidly adapt to a new task with a small number of resources of the task to ease a burden of a search process on a variety of tasks. While metrics can be any metrics such as accuracy, latency, and energy, etc, we focus on latency of diverse devices in this paper.

3.1 Problem Definition

Assume a task $\tau = \{h^\tau, \mathbf{X}^\tau, \mathbf{Y}^\tau\}$ where $h^\tau \in \mathcal{H}$ is a hardware device, $\mathbf{X}^\tau \subset \mathcal{X}$ is a set of neural architectures, and $\mathbf{Y}^\tau \subset \mathcal{Y}$ is a set of latencies of $\mathbf{X}^\tau$ directly measured on the hardware device h^τ. Learning conventional task-specific prediction model $f(x;\boldsymbol{\theta}) : \mathcal{X} \rightarrow \mathbb{R}$ of NAS parameterized by $\boldsymbol{\theta}$ is a regression problem that estimates latency $y \in \mathbf{Y}^\tau$ of a neural architecture $x \in \mathbf{X}^\tau$ for a given hardware device h^τ by minimizing empirical loss $\mathcal{L}$ (e.g. mean squared error) between predicted

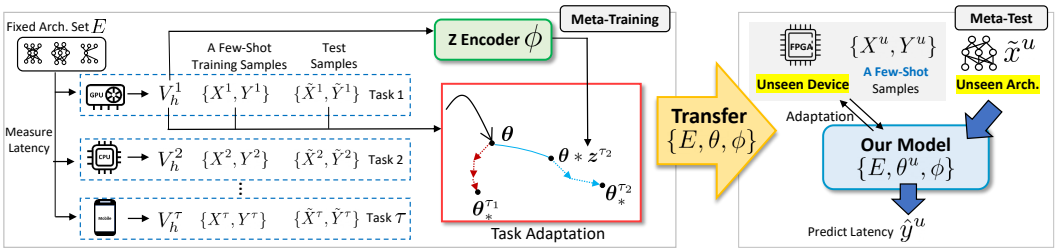

Figure 2: **Overview** For hardware-adaptive search, we introduce a latency-based hardware embedding and $\boldsymbol{z}$ modulator of initial parameters. By formulating sample-efficient NAS as a few-shot regression problem of meta-learning framework, our models successfully learn meta-knowledge $\boldsymbol{\theta}$ from source device pool. The meta-learned model achieves high sample efficiency on several unseen devices.

values $f(\mathbf{X}^\tau; \boldsymbol{\theta})$ and $\mathbf{Y}^\tau$ as follows:

$$\min_{\boldsymbol{\theta}} \mathcal{L}\big(f^\tau(\mathbf{X}^\tau; \boldsymbol{\theta}), \mathbf{Y}^\tau\big) \tag{1}$$

While such models have been work well on a single task, the model has several limitations considering real-world scenarios to support billions of devices:

1. Task-specific prediction models should repeatedly collect *a large number of* neural architecture-latency sample pairs (e.g. 2K [37], 5K [25] samples) for each device to achieve reliable prediction performance while they has difficulty utilizing already collected sample pairs of other devices, which is highly inefficient.
2. Task-specific prediction models repeat the whole NAS process including sample collection and training NAS model for each device, which results in prohibitive time-consuming for the total set of multiple devices.
3. We should keep multiple prediction models $\{f^\tau(\cdot; \boldsymbol{\theta}^\tau)\}_{\tau=1}^N$ for N multiple devices. As the number of target devices increases, the number of predictors also increases, which is heavy cost in the real world with billions of devices.

To overcome such limitations, we propose a **single** predictor $f(\cdot; \boldsymbol{\theta})$ which can cover N multiple devices by **fast** adapting to a new target device (unseen during training phase) only collecting a **small number** of architecture-latency pairs ($X^\tau \ll \mathbf{X}^\tau$, $Y^\tau \ll \mathbf{Y}^\tau$), by transferring knowledge obtained from the retained device pool $p(\tau)$.

3.2 Hardware-Conditioned Prediction Model

While the measured latency $y \leftarrow (x, h)$ is dependent on both device type h and architecture x, the structure of existing prediction model $f(x; \boldsymbol{\theta})$ ignores device information which results in poor performance when performing metric estimation on multi-devices with a single prediction model. Thus, we propose hardware-conditioned prediction model parameterized with $\boldsymbol{\theta}$:

$$f(x, h; \boldsymbol{\theta}) : \mathcal{X} \times \mathcal{H} \rightarrow \mathbb{R} \tag{2}$$

that can predict different latency dependent on device type h even for the same architecture x. The main challenge of our hardware-conditioned prediction model is to model hardware identifier as input embedding for $f(\cdot, h; \boldsymbol{\theta})$ for all devices regardless of hardware platform types, which is not trivial since the physical structure of devices is very different. Thus, we design a novel latency-based latent vector V_h as hardware condition, which contains representative tendency of latency values of a set of fixed architectures, instead of directly modelling hardware devices as follows:

$$V_h = [y_1(x_1, h), y_2(x_2, h), ..., y_d(x_d, h)] : E \times \mathcal{H} \rightarrow \mathbb{R}^d \tag{3}$$

where E is a set of fixed neural architectures $\{x_1, x_2, ..., x_d\} \subset \mathcal{X}$ measured on device h, E is fixed across of all tasks from both meta-training and meta-test, and d can be any small number, in our case, we set $d = 10$. The proposed latency-based hardware embedding allows us to predict metric values for multi-devices without considering hardware specification or structure.

3.3 Episodic Meta-Training

To tackle a few-shot regression problem for multiple devices by utilizing retained data pool $p(\tau)$, we adopt the gradient-based bi-level optimization framework [13, 21] that meta-learn the initial model parameter $\boldsymbol{\theta}$ as a meta-knowledge to generalize over the task distribution $p(\tau)$ rather than a single task τ. By doing so, we can easily obtain the task-specific predictor $\boldsymbol{\theta}^\tau$ in a few gradient steps from the initial parameter $\boldsymbol{\theta}$. During meta-training phase, we take episodic training strategy which simulates a few-shot regression task for each iteration by randomly sampling task τ from the retained device pool $p(\tau)$ and spliting τ as training set $\mathcal{D} = \{h^\tau, X^\tau, Y^\tau\}$ and test set $\tilde{\mathcal{D}} = \{h^\tau, \tilde{X}^\tau, \tilde{Y}^\tau\}$ where $X^\tau \subset \mathbf{X}$ and $\tilde{X}^\tau \subset \mathbf{X}$ denote sets of neural architecture samples, X^τ is few-shot samples $X^\tau \ll \mathbf{X}$, there is no overlap between them $X^\tau \cap \tilde{X}^\tau = \emptyset$ and $Y^\tau \subset \mathbf{Y}^\tau$ and $\tilde{Y}^\tau \subset \mathbf{Y}^\tau$ denote sets of corresponding latency values of neural architectures X^τ and $\tilde{X}^\tau$, respectively measured on device h^τ. For each task, we obtain the hardware-adaptive prediction model $\boldsymbol{\theta}^\tau = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(X^\tau, V_h^\tau; \boldsymbol{\theta}), Y^\tau))$ by conducting inner gradient updates on few-shot samples $\mathcal{D} = \{h^\tau, X^\tau, Y^\tau\}$. This model is trained to minimize the test loss $\mathcal{L}(\cdot; \tilde{\mathcal{D}}^\tau)$ by optimizing following gradient-based meta-learning objective:

$$\min_{\boldsymbol{\theta}} \sum_{\tau \sim p(\tau)} \mathcal{L}\big(f\big(\tilde{X}^\tau, V_h^\tau; \boldsymbol{\theta}^\tau\big), \tilde{Y}^\tau\big) \tag{4}$$

where α denotes multi-dimensional global learning rate vector [21]. Furthermore, when we handle a very new device, which has less relatedness of distribution of obtained device pool, it is helpful to move largely task-specific parameter from the meta-knowledge compared with the case of other in-distribution devices [17]. We introduce hardware-adaptive modulator $\mathbf{z}^\tau = g(V_h^\tau; \boldsymbol{\phi}) : \mathbb{R}^d \rightarrow \mathbb{R}^{d_{\boldsymbol{\theta}}}$ with $\mathbf{z}$ encoder $g(\cdot; \boldsymbol{\phi})$ parameterized by ϕ to modulate initial parameter as $\boldsymbol{\theta}_0 = \boldsymbol{\theta} * \mathbf{z}^\tau$ where $\boldsymbol{\theta}_0$ is new initialization point for hardware h^τ. Following [17], we concretely set $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta} \circ \mathbf{z}^\tau$ for weights and $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta} + \mathbf{z}^\tau$ for biases with multiplication operator $\circ$. Final meta-learning framework for sample-efficient prediction-based NAS model as follows:

$$\boldsymbol{\theta}_0^\tau = \boldsymbol{\theta} * \mathbf{z}^\tau \tag{5}$$

$$\boldsymbol{\theta}_k^\tau = \boldsymbol{\theta}_{k-1}^\tau - \alpha \nabla_{\boldsymbol{\theta}_{k-1}^\tau} \mathcal{L}(f(X^\tau, V_h^\tau; \boldsymbol{\theta}_{k-1}^\tau), Y^\tau)) \quad \text{for } k = 1, \ldots, K \tag{6}$$

where K is the number of inner gradient steps and the resultant final objective as follows:

$$\min_{\boldsymbol{\theta}, \boldsymbol{\phi}} \sum_{\tau \sim p(\tau)} \mathcal{L}\big(f\big(\tilde{X}^\tau, V_h^\tau; \boldsymbol{\theta}_K^\tau\big), \tilde{Y}^\tau\big) \tag{7}$$

Note that the existing meta-learning methods [13, 21] have dealt with the few-shot regression problem, but, it has a limitation in that it is artificial settings such as sine wave. On the other hand, we tackle the real-world problem and, to our best knowledge, our work is the first to target such latency estimation of multi-devices with few-shot data samples. Although Neural process [16] can be an alternative framework for the few-shot latency prediction model, we observe that it underperforms compared with our method as shown in Section X.

3.4 Transferring to New Devices (Meta-Test)

Equipped with meta-training components described in the previous subsections, we now describe how to use the prediction model with meta-learned initial parameters $f(\cdot; \boldsymbol{\theta})$ on a new device that is unseen during meta-training. Different with task-specific predictors [37, 5, 25] that collect a large amount of neural architecture-latency pairs for the unseen device $h^\upsilon \notin \mathbf{H}$, we measure a few-shot latency values Y^υ of randomly sampled neural architectures X^υ to construct query task $\upsilon = \{h^\upsilon, X^\upsilon, Y^\upsilon\}$. Then, we can get the task-specific parameters $\boldsymbol{\theta}^\upsilon = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(X^\upsilon, V_h^\upsilon; \boldsymbol{\theta}), Y^\upsilon)$ such that we can predict latency $f(\tilde{x}^\upsilon; \boldsymbol{\theta}^\upsilon)$ for a neural architecture $\tilde{x}^\upsilon \notin X^\upsilon$. We can easily extend the Eq. (4), such that we obtain $\boldsymbol{\theta}^\upsilon$ with more than one inner-gradient steps from the initial $\boldsymbol{\theta}$.

4 Experiment

In this section, we will first analyze the behavior of our meta-latency predictor in NAS-Bench-201 [9] space, which is small and controlled space. Afterwards, we will search competitive architecture on diverse hardware devices using our predictor in hardware friendly architecture space, OFA [5] search space.

Table 1: **Correlation on Unseen Devices with Few-shot Samples (Meta-Test)**

Method	# of Sample (N)	Correlation						
		RTX Titan	Xeon Gold	Pixel3	Raspi4	Eyeriss	FPGA	Mean
Task-Specific Predictor	10	0.266	0.318	0.366	0.303	0.315	0.346	0.319
MAML [13]	10	0.738	0.544	0.576	0.568	0.602	0.541	0.595
Meta-SGD [21]	10	0.924	0.831	0.786	0.844	0.831	0.882	0.850
ANP [16]	10	0.888	0.853	0.745	0.801	0.657	0.884	0.805
HaNAS (Ours)	10	**0.964**	**0.925**	**0.894**	**0.885**	**0.942**	**0.889**	**0.916**

4.1 Meta-Test on Unseen Devices

We first validate our model **Baselines**

1-1) Task-Specific Predictor

1-2) MAML [13]

1-3) Meta-SGD [21]

1-4) ANP [16]

1-5) Ours

4.2 NAS-Bench-201

NAS-Bench-201 [9] search space contains cell-based neural architectures, where each cell is represented as directed acyclic graph (DAG) consisting of the 4 nodes and the 6 edge connections. For each edge connection, NAS models select one of 5 operation candidates such as zerorize, skip connection, 1-by-1 convolution, 3-by-3 convolution, and 3-by-3 average pooling, leading to total 15625 unique architectures.

We practically collect latency data of all NAS-Bench-201 architecture on multiple devices, such as GPU, CPU, mobile phone (NVIDIA 1080ti, Titan X, Titan XP, 2080ti, RTX Titan GPUs, Intel CPUs, Google Pixel2, Pixel3, Samsung A50) using CIFAR-10 dataset. For GPU, two batch sizes are considered (1, 256) and for all other hardware, batch size is set to 1. Note that different batch size is considered different hardware even if the same GPU. Moreover, we use data of ASIC, Jetson, Raspi4, FPGA from hw-nas-bench to increase diversity of device pool. Device pool is split as meta-train deivce : 1080ti, titan X, 1 CPU, 1Mobile phone, and meta-test device : rtx titan, cpu, mobile phone, jetson, ASIC, rapsi 4 to consider both cases, whether there is device type similar to target device in training pool.

Implementation Details We utilize GCN to encode graph structure of cell in NAS-Bench-201 space. Note that any architecture encoding scheme is compatible with our latency predictor (we use MLP to encode architecture in OFA search space). Then architecture embedding is concatenated with hardware embedding vector and pass MLP to predict latency of the architecture on the specific hardware.

4.3 OFA space

OFA search space is a linear structured architecture space, each block is composed of an inverted bottleneck convolution. They support diversity of four dimension (depth, width, kernel size, and resolution) The input image size ranges from 128 to 224 with a stride 4; the depth of each stage is chosen from .{2, 3, 4.}; the expansion ratio of each block is chosen from 3, 4, 6; the kernel size is chosen from .{3, 5, 7.}. With total 5 stage, roughly 2*10 19 unique architectures can be established and each of them can be used under 25 different input resolution.

Hardware Devices and Dataset

1) Meta-training

2) Meta-test

Search Space

Table 2: **Ablation Study**

Method	# of Sample (N)	Correlation						
		RTX Titan	Xeon Gold	Pixel3	Raspi4	Eyeriss	FPGA	Mean
Amortized meta-learning	10	0.738	0.546	0.577	0.568	0.604	0.539	0.596
+ Hardware-Condition	10	0.957	0.920	0.861	0.853	0.904	0.861	0.893
+ Few-shot Adaptation	10	0.964	**0.927**	0.865	0.872	0.913	0.866	0.901
+ z Modulator (HaNAS)	10	**0.964**	0.925	**0.894**	**0.885**	**0.942**	**0.889**	**0.916**

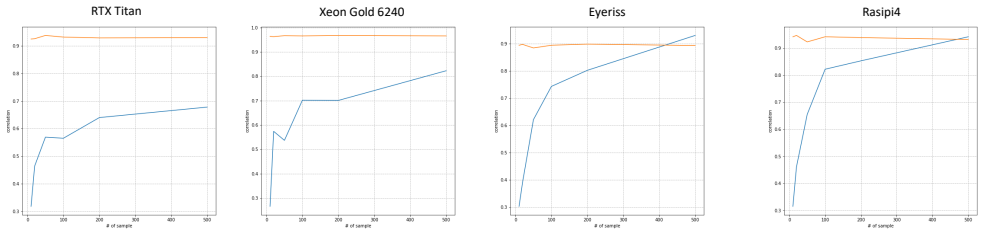

Figure 3: **Uncertainty-based Sample Efficiency**

1) MobileNetV3

2) NAS-Bench-201

3) HAT (NLP)

Implementation Details

Baselines and NAS Model

1-1) LUT

1-2) Predictor (rand init)

1-3) MAML [13]

1-4) Meta-SGD [21]

1-5) ANP [16]

1-6) Ours

2-1) OFA [5]

2-2) BRP-NAS [10]

2-3) HAT [37]

2-4) Ours

4.4 Latency Prediction with Few-shot Architecture Samples

Corr results 2

4.5 Ablation Study of Our Meta-Predictor

4.6 Effectiveness of The Number of Samples

Effectiveness of sampling 3

Table 3: **NAS Results on Unseen Devices (Meta-Test)**

Search Space	NAS-Bench-201							
Model	BRP-NAS [10]				HaNAS (Ours)			
	Sample	Time (hours)	Latency (ms)	Accuracy (%)	Sample	Time (hours)	Latency (ms)	Accuracy (%)
Desktop GPU	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
Desktop CPU	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
Galaxy S21	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
Google Pixel5	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x

Table 4: **NAS Results on Unseen Devices (Meta-Test)**

Search Space	MobileNetV3							
Model	OFA [5]				HaNAS (Ours)			
	Sample	Time (hours)	Latency (ms)	Accuracy (%)	Sample	Time (hours)	Latency (ms)	Accuracy (%)
Desktop GPU	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
Desktop CPU	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
Galaxy S21	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
Google Pixel5	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x

4.7 Effectiveness of Meta-Device Pool

4.7.1 How Many Devices Do We Need to Construct Device Pool?

4.7.2 Which Devices Do We Need to Construct Device Pool?

4.8 Combining Existing NAS Methods

NAS results **??**

4.9 Toward Generic Meta-predictor

5 Conclusion

References

[1] Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. Zero-cost proxies for lightweight nas. *International Conference on Learning Representations, 2021*, 2021.

[2] Maxim Berman, Leonid Pishchulin, Ning Xu, Matthew B Blaschko, and Gérard Medioni. Aows: Adaptive and optimal network width search with latency constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11217–11226, 2020.

[3] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

[6] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021.

[7] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019.

[8] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition (CVPR)*, 2019.

[9] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

[10] Łukasz Dudziak, Thomas Chau, Mohamed S Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D Lane. Brp-nas: Prediction-based nas using gcns. *Advances in neural information processing systems (NeurIPS)*, 2020.

[11] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2020.

[12] Jiemin Fang, Yuzhu Sun, Kangjian Peng, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Fast neural network adaptation via parameter remapping and architecture search. *arXiv preprint arXiv:2001.02525*, 2020.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[14] Ruibing Hou, Hong Chang, MA Bingpeng, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[16] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

[17] Hae Beom Lee, Hayeon Lee, Donghyun Na, Saehoon Kim, Minseop Park, Eunho Yang, and Sung Ju Hwang. Learning to balance: Bayesian meta-learning for imbalanced and out-of-distribution tasks. In *International Conference on Learning Representations (ICLR)*, 2020.

[18] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021.

[19] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[20] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021.

[21] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[22] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *International Conference on Learning Representations (ICLR)*, 2019.

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

[24] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision*, pages 35–51. Springer, 2020.

[25] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems (NeurIPS)*, 2018.

[26] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[27] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018.

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

[29] Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[30] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems (NIPS)*, 2017.

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

[32] Yehui Tang, Yunhe Wang, Yixing Xu, Hanting Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. A semi-supervised assessor of neural architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1810–1819, 2020.

[33] Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0-7923-8047-9.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[35] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems (NIPS)*, 2016.

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

[37] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[38] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 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, 2019.

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

[40] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Bowen Shi, Qi Tian, and Hongkai Xiong. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392*, 2020.

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

[42] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11396–11404, 2020.

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

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

[45] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? **[TODO]**
   (b) Did you describe the limitations of your work? **[TODO]**
   (c) Did you discuss any potential negative societal impacts of your work? **[TODO]**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? **[TODO]**

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? **[TODO]**
   (b) Did you include complete proofs of all theoretical results? **[TODO]**
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? **[TODO]**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? **[TODO]**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? **[TODO]**
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? **[TODO]**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? **[TODO]**
   (b) Did you mention the license of the assets? **[TODO]**
   (c) Did you include any new assets either in the supplemental material or as a URL? **[TODO]**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? **[TODO]**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? **[TODO]**
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? **[TODO]**
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? **[TODO]**
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? **[TODO]**