Rapid Model Architecture Adaption for Meta-Learning

Yiren Zhao*
Imperial College London
a.zhao@imperial.ac.uk

Xitong Gao*
Shenzhen Institute of Advanced Technology, CAS
xt.gao@siat.ac.cn

Ilia Shumailov
University of Oxford
ilia.shumailov@chch.ox.ac.uk

Nicolo Fusi
Microsoft Research
fusi@microsoft.com

Robert D Mullins
University of Cambridge
robert.mullins@cl.cam.ac.uk

Abstract

Network Architecture Search (NAS) methods have recently gathered much attention. They design networks with better performance and use a much shorter search time compared to traditional manual tuning. Despite their efficiency in model deployments, most NAS algorithms target a single task on a fixed hardware system. However, real-life few-shot learning environments often cover a great number of tasks (T) and deployments on a wide variety of hardware platforms (H).

The combinatorial search complexity $T \times H$ creates a fundamental search efficiency challenge if one naively applies existing NAS methods to these scenarios. To overcome this issue, we show, for the first time, how to rapidly adapt model architectures to new tasks in a *many-task many-hardware* few-shot learning setup by integrating Model Agnostic Meta Learning (MAML) into the NAS flow. The proposed NAS method (H-Meta-NAS) is hardware-aware and performs optimisation in the MAML framework. H-Meta-NAS shows a Pareto dominance compared to a variety of NAS and manual baselines in popular few-shot learning benchmarks with various hardware platforms and constraints. In particular, on the 5-way 1-shot Mini-ImageNet classification task, the proposed method outperforms the best manual baseline by a large margin (5.21% in accuracy) using 60% less computation.

1 Introduction

Existing Network Architecture Search (NAS) methods show promising performance on image [Zoph and Le, 2016, Liu et al., 2018], language [Guo et al., 2019, So et al., 2019] and graph data [Zhao et al., 2020b]. The automation not only reduces the human effort required for architecture tuning but also produces architectures with state-of-the-art performance in domains like image classification [Zoph and Le, 2016] and language modeling [So et al., 2019]. Most NAS methods today focus on a single task with a fixed hardware system, yet real-life model deployments covering multiple tasks and various hardware platforms will significantly prolong this process. As illustrated in Figure 1, a common design flow is to re-engineer the architecture and train for the different task(T)-hardware(H) pairs with different constraints (C). The architectural engineering phase can be accomplished whether

*Correspondence to: Yiren Zhao and Xitong Gao.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

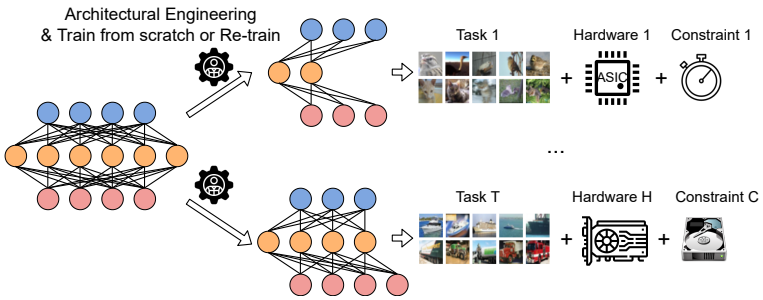


Figure 1: Deploying networks in a *many-task many-device* few-shot learning setup. This implies a large search complexity $\mathcal{O}(THC)$.

manually or by using an established NAS procedure. The challenge is designing an efficient method to overcome the quickly scaling $\mathcal{O}(THC)$ search complexity described in Figure 1.

Few-shot learning systems follow exactly this *many-task many-device* setup, when considering deployments on different user devices on key applications such as facial [Guo et al., 2020] and speech recognition [Hsu et al., 2020]. A task in few-shot learning normally takes an N-way K-shot formulation, where it contains N classes with K support samples and Q query samples in each class. Model-Agnostic Meta-Learning (MAML), incorporating the idea of learning to learn, builds a meta-model using a great number of training tasks, and then adapts the meta-model to unseen test tasks using only a very small number of gradient updates [Finn et al., 2017]. MAML then becomes a powerful and elegant approach for few-shot learning – its ability to quickly adapt to new tasks can potentially shrink the $\mathcal{O}(THC)$ complexity illustrated in Figure 1 to $\mathcal{O}(HC)$. In the meantime, hardware-aware NAS methods [Cai et al., 2019, 2018, Xu et al., 2020], *e.g.* the train-once-for-all technique [Cai et al., 2019], support deployments of searched models to fit different hardware platforms with various latency constraints. These hardware-aware NAS techniques further reduce the search complexity from $\mathcal{O}(THC)$ to $\mathcal{O}(T)$ [Cai et al., 2018].

In this paper, we propose a novel Hardware-aware Meta Network Architecture Search (H-Meta-NAS). Integration of the MAML framework into hardware-aware NAS theoretically reduces the search complexity from $\mathcal{O}(THC)$ to $\mathcal{O}(1)$, allowing for a rapid adaption of model architectures to unseen tasks on new hardware systems. However, we identified the following challenges in this integration:

- Classic NAS search space contains many over-parameterized sub-models, this makes it hard to tackle the over-fitting phenomenon in few-shot learning.
- Hardware-aware NAS profiles latency for sub-networks on each task-hardware pair, this profiling can be prolonged significantly with a great number of tasks and, more importantly, if the targeting device has scarce computation resources.

To tackle these challenges, we then propose to use Global Expansion (GE) and Adaptive Number of Layers (ANL) to allow a drastic change in model capabilities for tasks with varying difficulties. Our experiments later demonstrate that such changes alleviate over-fitting in few-shot learning and improve the accuracy significantly. We also present a novel layer-wise profiling strategy to allow the reuse of profiling information across different tasks. We make the following contributions:

- We propose a novel Hardware-aware Network Architecture Search for Meta learning (H-Meta-NAS). H-Meta-NAS quickly adapts meta-architectures to new tasks with hardware-awareness and can be conditioned with various device-specific latency constraints. The proposed NAS reduces search complexity from $\mathcal{O}(THC)$ to $\mathcal{O}(1)$ in a realistic *many-task many-device* few-shot learning setup. We extensively evaluate H-Meta-NAS on various hardware platforms (GPU, CPU, mCPU, IoT, ASIC accelerator) and constraints (latency and model size), our latency-accuracy performance generally outperforms various MAML baselines and other NAS competitors.
- We propose a **task-agnostic layer-wise profiling strategy** that reduces the profiling runtime from around 10^5 hours to 1.2 hours when targeting hardware with limited capabilities (*e.g.* IoT devices).

- MAML models are prone to over-fitting in the few-shot learning setup Antoniou et al. [2018]. We show several design options for the NAS algorithm, namely Global Expansion and Adaptively Number of Layers respectively. These methods help the NAS to overcome the over-fitting problem by optimizing in the architectural design space.
- We reveal that **popular Metric-based MAML methods are not latency-friendly**. Our empirical results suggest that Optimization-based MAML method with well-tuned architectures can achieve comparable accuracy with significantly less latency overhead ($\approx 2000\times$ on Mini-ImageNet 5-way 1-shot classification).

2 Background

2.1 Few-shot learning in the MAML framework

Inspired by human's ability to learn from only a few tasks and generalize the knowledge to unseen problems, a meta learner is trained over a distribution of tasks with the hope of generalizing its learned knowledge to new tasks Finn et al. [2017].

$$\arg\min_{\theta}(\mathbb{E}_{\mathcal{T}\in\mathbb{T}}[\mathcal{L}_{\theta}(\mathcal{T})]) \tag{1}$$

Equation (1) captures the optimization objective of meta-learning, where optimal parameters are obtained through optimizing on a set of *meta-training* tasks. Current approaches of using meta-learning to solve few-shot learning problems can be roughly categorized into three types: Memory-based, Metric-based and Optimization-based.

Memory-based method utilizes a memory-augmented neural network [Munkhdalai and Yu, 2017, Gidaris and Komodakis, 2018] to memorize meta-knowledge for a fast adaption to new tasks. Metric-based methods aim to meta-learn a high-dimensional feature representation of samples, and then apply certain metrics to distinguish them. For instance, Meta-Baseline utilizes the cosine nearest-centroid metric [Chen et al., 2020] and DeepEMD applies the Wasserstein distance [Zhang et al., 2020]. Optimization-based method, on the other hand, focuses on learning a good parameter initialization (also known as *meta-parameters* or *meta-weights*) from a great number of training tasks, such that these meta-parameters adapt to new few-shot tasks within a few gradient updates. The most well-established Optimization-based method is Model-Agnostic Meta-Learning (MAML) [Finn et al., 2017]. MAML is a powerful yet simple method to tackle the few-shot learning problem, since its adaption relies solely on gradient updates. Antoniou et al. later demonstrate MAML++, a series of modifications that improved MAML's performance and stability. Baik et al. introduce an additional network for generating adaptive parameters for the inner-loop optimization.

Despite the rise in popularity of the meta-learning framework applied to few-shot learning, little attention has been paid to the runtime efficiency of these approaches. In this work, we reveal that *the current mainstream Metic-based methods all suffer from a severe latency overhead*, since Metric-based approaches rely on metric comparisons and have to use multiple inference runs to generate these metrics (at least two) for a single image classification. With the help of H-Meta-NAS, we show how Optimization-based methods can achieve a similar level of accuracy but using significantly less computation at inference time.

2.2 Network architecture search

Architecture engineering is a tedious and complex process requiring a lot of effort from human experts. Network Architecture Search (NAS) focuses on reducing the amount of manual tuning in this design space. Early NAS methods use evolutionary algorithms and reinforcement learning to traverse the search space [Zoph and Le, 2016, Real et al., 2017]. These early methods require scoring architectures trained to a certain convergence and thus use a huge number of GPU hours. Two major directions of NAS methods, Gradient-based and Evolution-based methods, are then explored in parallel in order to make the search cost more affordable. Gradient-based NAS methods use Stochastic Gradient Descent (SGD) to optimize a set of probabilistic priors that are associated with architectural choices [Liu et al., 2018, Casale et al., 2019]. Although these probabilistic priors can be made latency-aware [Wu et al., 2019, Xu et al., 2020], it is challenging to make them follow a hard latency constraint. Evolution-based NAS, on the other hand, operates on top of a pre-trained super-net and use evolutionary algorithms or reinforcement learning to pick best-suited sub-networks

[Cai et al., 2018, 2019], making it easier to be constrained by certain hardware metrics. For instance, Once-for-all (OFA) is an Evolution-based NAS method and its searched networks are not only optimized for a specific hardware target but also constrained by a pre-defined latency budget [Cai et al., 2019]. Our proposed H-Meta-NAS shares certain similarities to Once-for-all, since this method offers a chance to reduce the hardware search complexity from $\mathcal{O}(HC)$ to $\mathcal{O}(1)$. Latency is a key metric in mobile applications, and obviously can be also improved through other techniques such as quantization [Zhao et al., 2019a, Hönig et al., 2022] and sparsification [Gao et al., 2018, Wang et al., 2019] in various learning setups, however, many of these optimizations might be ineffectual [Nikolić et al., 2019] unless the underlying hardware has special support for it [Su et al., 2018, Zhao et al., 2019b, Parashar et al., 2017]. NAS algorithm is a promising approach to address this problem, since it can directly optimize the network topology; recent NAS methods have also been extended to Transformers [So et al., 2019] and Graph Neural Networks [Zhao et al., 2020b,a].

Several NAS methods are proposed under the MAML framework [Kim et al., 2018, Shaw et al., 2018, Lian et al., 2019], these methods successfully reduce the search complexity from $\mathcal{O}(T)$ to $\mathcal{O}(1)$. However, some of these methods do not show significant performance improvements compared to carefully designed MAML methods (*e.g.* MAML++) [Kim et al., 2018, Shaw et al., 2018]. In the meantime, some of these MAML-based NAS methods follow the Gradient-based approach and operate on complicated cell-based structures [Lian et al., 2019]. We illustrate later how cell-based NAS causes an undesirable effect on latency, and also meets fundamental scalability challenges when trying to deploy in a *many-task many-device* few-shot learning setup.

3 Method

Problem formulation In the MAML setup, we consider a set of tasks $\mathbb{T}$ and each task $\mathcal{T}_i \in \mathbb{T}$ contains a support set $\mathcal{D}_i^s$ and a query set $\mathcal{D}_i^q$. The support set is used for task-level learning while the query set is in charge of evaluating the meta-model. All tasks are divided into three sets, namely meta-training ($\mathbb{T}_{train}$), meta-validation ($\mathbb{T}_{val}$) and meta-testing ($\mathbb{T}_{test}$) sets.

Equation (2) formally states the objective of the pre-training stage illustrated in Figure 3a. The objective of this process is to optimize the parameters θ of the super-net for various sub-networks sampled from the architecture set $\mathbb{A}$. This will ensure the proposed H-Meta-NAS to have both the meta-parameters and meta-architectures ready for the adaption to new tasks.

$$\arg\min_{\theta} \mathbb{E}_{\alpha \sim p(\mathbb{A})}[\mathbb{E}_{\mathcal{T} \in \mathbb{T}_{train}}[\mathcal{L}_\theta(\mathcal{T}, \alpha)]] \tag{2}$$

Equation (3) describes how H-Meta-NAS adapts network architectures to a particular task $\mathcal{T}$ with a given hardware constraint C_h. In practice, using the support set data $\mathcal{D}_i^s$ from a target task $\mathcal{T}_i$, we apply a genetic algorithm for finding the optimal architectures α^*. We discuss further how this process in detail in later sections.

$$\begin{aligned} \alpha^* = \min_{\alpha} & \sum_{\alpha \in \mathbb{A}} \mathcal{L}_\theta(\mathcal{D}_i^s, \alpha) \\ \text{s.t.} \quad & \mathcal{C}(\alpha) \leq C_h \end{aligned} \tag{3}$$

Architecture search space H-Meta-NAS considers a search space composed of different kernel sizes, the number of channels, and activation types. We mostly consider a VGG9-based NAS backbone, that is a 5-layer CNN model with the last layer being a fully connected layer. We chose this NAS backbone because both MAML Finn et al. [2017] and MAML++ Antoniou et al. [2018] used a VGG9 model architecture. The details of this backbone are in Appendix A.

We allow kernel sizes to be picked from $\{1, 3, 5\}$, channels to be expanded with a set of scaling factors $\{0.25, 0.5, 0.75, 1, 1.5, 2, 2.25\}$ and also six different activation functions (details in Appendix). For a single layer, there are $3 \times 7 \times 6 = 126$ search options. H-Meta-NAS also contains an Adaptive Number of Layers strategy, the network is allowed to use a subset of the total layers in the supernet with maximum usage of 4 layers. The whole VGG9-based backbone then gives us in total $126^4 \times 4 \approx 10^9$ possible neural network architectures.

In addition, to demonstrate the ability of H-Meta-NAS on a more complex NAS backbone. We also studied an alternative ResNet12-based NAS backbone, that has approximately 2×10^{24} possible sub-networks.

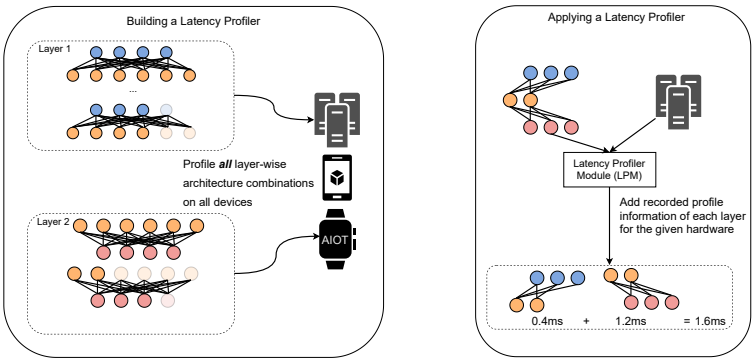
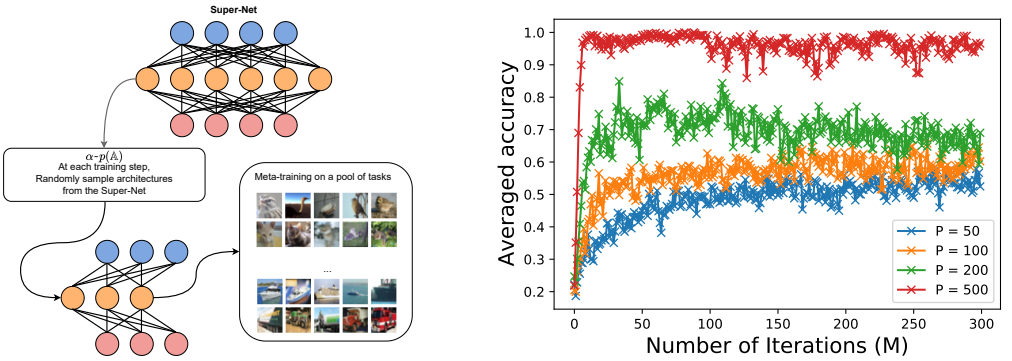


Figure 2: An illustration of how Latency Profiler works. The left side shows how the profiler is built by recording the run-time of ***all configurations of each layer***. The right side shows this profiler then can sum the layer-wise information to provide a hardware-aware runtime for a whole network.



(a) Super-net meta-training.

(b) The effect of different pool size (P) and different number of iterations (M).

Figure 3: Figure 3a is an overview of the super-net meta-training. Figure 3b shows how different parameters in the Adaption strategy can affect accuracy.

Layer-wise profiling Hardware-aware NAS needs the run-time of sub-networks on the targeting hardware to guide the search process Cai et al. [2019], Xu et al. [2020]. However, the profiling stage can be time-consuming if given a low-end hardware as the profiling target and the search space is large. For instance, running a single network inference of VGG9 on the Raspberry Pi Zero with a 1GHz single-core ARMv6 CPU takes around 2.365 seconds to finish. If we assume this is the average time needed for profiling a sub-network, given that the entire search space includes around 10^9 sub-networks, a naive traverse will take a formidable amount of time which is approximately 6×10^5 hours. More importantly, the amount of profiling time scales with the number of hardware devices ($\mathcal{O}(H)$). Existing hardware-aware NAS schemes build predictive methods to estimate the run-time of sub-networks Cai et al. [2019], Xu et al. [2020] and have a relatively significant error. An illustration of a profiler is shown in Figure 2, and the latency predictor's illustration is in Figure 8.

Adaption strategy The adaption strategy uses a genetic algorithm Whitley [1994] to pick the best-suited sub-network with respective to a given hardware constraint, the full algorithm is detailed in Appendix D. In general, the adaption algorithm randomly samples a set of tasks from $\mathbb{T}_{val}$, and uses the averaged loss value and satisfaction to the hardware constraints as indicators for the genetic algorithm. The genetic algorithm has a pool size P and number of iterations M.

We then run a hyper-parameter analysis in Figure 3b to determine the values of P and M. The full adaption algorithm makes use of these hyper-parameters is in our Appendix. The horizontal axis shows the number of iterations and the vertical axis shows the averaged accuracy on the sampled tasks for all architectures in the pool. Figure 3b shows that the accuracy convergence is reached after

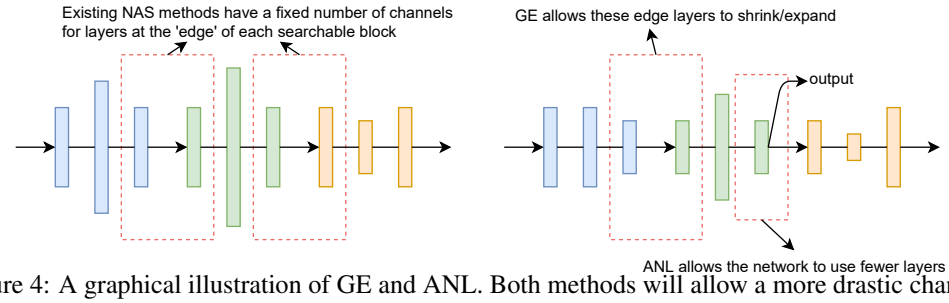


Figure 4: A graphical illustration of GE and ANL. Both methods will allow a more drastic change in model capabilities, allowing the searched model to deal with tasks with varying difficulties.

around 150 iterations, and running for additional iterations only provides marginal accuracy gains. For this reason, we picked the number of iterations to be 200 for a balance between accuracy and run-time. In the meantime, we notice in general a higher pool size will give better-adapted accuracy. However, this does not mean the final searched accuracy is affected to the same degree. The final re-trained accuracy of searched architectures show an accuracy gap of 0.21% between $P = 100$ and $P = 200$ and 0.32% between $P = 100$ and $P = 500$. An increase in pool size can prolong the run-time significantly, we thus picked a pool size of 100 since it offers the best balance between accuracy and run-time.

NAS backbone design, Global Expansion (GE) and Adaptive Number of Layers (ANL) One particular problem in few-shot learning is that models are prone to over-fitting. This is because only a small number of training samples are available for each task and the network normally iterate on these samples many times Antoniou et al. [2018]. We would like to explore the architectural space to help models overcome over-fitting and conduct a case study for different design options available for the backbone network. We identify the following key changes to the NAS backbone to help the models to have high accuracy in few-shot learning:

- $n \times n$ pooling: Pooling that applied to the final convolutional operation, $n \times n$ indicates the height and width of feature maps after pooling.
- Global Expansion (GE): Allowing the NAS to globally expand or shrink the number of channels of all layers.
- Adaptive Number of Layers (ANL): Allowing the NAS to use an arbitrary number of layers, the network then is able to early stop using only a fewer number of layers.

Figure 4 further illustrates how GE and ANL can allow a much smaller model compared to existing NAS backbones. We also discuss in Section 3 how the pooling strategy can join the NAS process.

Super-net meta-training strategy As illustrated by prior work Cai et al. [2019], progressively shrinking the super-net during meta-training can reduce the interference between sub-networks. We observe the same phenomenon and then use a similar progressive shrinking strategy in H-Meta-NAS, the sampling process $\alpha \sim p(\mathbb{A})$ will pick the largest network with a probability of p, and randomly pick other sub-networks with a probability of $1 - p$. We apply an exponential decay strategy to p:

$$p = p_e + (p_i - p_e) \times exp(-\alpha \times \frac{e - e_s}{e_m - e_s})) \tag{4}$$

p_e and p_i are the end and initial probabilities. e is the current number of epochs, and e_s and e_m are the starting and end epochs of applying this decaying process. α determines how fast the decay is. A graphical illustration is shown in Figure 3a. In our experiment, we pick $p_i = 1.0$ and $e_s = 30$, because the super-net reaches a relatively stable training accuracy at that point.

We then start the decaying process, and the value $\alpha = 5$ is determined through a hyper-parameter study shown in our Appendix B.

Table 1: Comparing latency predictor with our proposed profiling. MSE Error is the error between estimated and measured latency, Time is the total time taken to collect and build the estimator.

Hardware	Metric	Latency Predictor	Layer-wise Profiling
2080 Ti GPU	MSE Error	0.0188	0.00690
	Time	16.09 mins	6.216 secs
Intel i9 CPU	MSE Error	0.165	0.0119
	Time	21.92 mins	16.41 secs
Pi Zero	MSE Error	N/A	0.00742
	Time	N/A (Approx. 220 hours)	82.41 mins

4 Evaluation

We evaluate H-Meta-NAS on a range of popular few-shot learning benchmarks. For each dataset, we search for the meta-architecture and meta-parameters. We then adapt the meta-architecture with respect to a target hardware-constraint pair. In the evaluation stage, we then re-train the obtained hardware-aware task-specific architecture to convergence and report the final accuracy. We consider three popular datasets in the few-shot learning community: Omniglot, Mini-ImageNet and Few-shot CIFAR100. We use the PytorchMeta framework to handle the datasets [Deleu et al., 2019].

Omniglot is a handwritten digits recognition task, containing 1623 samples [Lake et al., 2015]. We use the meta train/validation/test splits originally used Vinyals *et al.* Vinyals et al. [2016]. These splits are over 1028/172/423 classes (characters).

Mini-ImageNet is first introduced by Vinyals *et al.*. This dataset contains images of 100 different classes from the ILSVRC-12 dataset [Deng et al., 2009], the splits are taken from Ravi *et al.*[Ravi and Larochelle, 2016].

Table 10 in our appendix details the systems and representative devices considered. We use the ScaleSIM cycle-accurate simulator Samajdar et al. [2018] for the Eyeriss Chen et al. [2016] accelerator. This simulation and more datasets and search configurations information are in Appendix F.

Evaluating layer-wise profiling We re-implemented the latency predictor in OFA Cai et al. [2019] as a baseline and compare it to our layer-wise profiling. We pick 16K training samples and 10K validation samples to train and test the latency predictor, which is the same setup used in OFA. We use another 10K testing samples to evaluate the performance of OFA-based latency predictor against our layer-wise profiling on different hardware systems in terms of MSE (measuring the latency estimation quality) and Time (measuring the efficiency). As illustrated in Table 1, layer-wise profiling saves not only time but also has a smaller MSE error compared to a predictor-based strategy that is very popular in today's evolutionary-based NAS frameworks [Cai et al., 2019, 2018]. In addition, layer-wise profiling shows orders of magnitude better run-time when targeting hardware devices with scarce computational resources. If we consider an IoT class device as a target (i.e the Raspberry Pi Zero), it requires an unreasonably large amount of time to generate training samples for latency predictors, making them an infeasible approach in real life. For instance, the total time consumed by the latency predictor is infeasible to execute on Pi Zero (last row in Table 1). Of course, in reality, there is also a great number of IoT devices using more low-end CPUs compared to Pi Zero (ARMV5 or ARMV4), making the latency predictor even harder to be deployed on these devices. Also in a many-hardware setup considered in this paper, this profiling is executed $\mathcal{O}(H)$ times.

Most existing layer-wise profile/look-up approaches consider at most mobile systems as targeting platforms [Xu et al., 2020, Yang et al., 2018]. These systems are in general more capable than a great range of IoT devices. In this paper, we demonstrate the effectiveness of this approach on more low-end systems (Raspberry Pi and Pi Zeros), illustrating this is the more scalable approach for hardware-aware NAS operating on constrained hardware systems.

Evaluating GE and ANL Our results in Table 2 suggest that a correct pooling strategy, GE and ANL can change the NAS backbone to allow the search space to reach much smaller models and thus provide better accuracy. In addition, Table 2 also illustrates that 5×5 pooling is necessary for higher accuracy. We hypothesize this is because a relatively large fully-connected layer after the

Table 2: A case study of different design options for the NAS backbone network. Experiments are executed with a model size constraint of $70K$ on the Mini-ImageNet 5-way 1-shot classification task.

Design options	Accuracy
MAML	48.70%
MAML++	52.15%
H-Meta-NAS + 1×1 Pool	42.28%
H-Meta-NAS + 5×5 Pool	46.13%
H-Meta-NAS + 5×5 Pool + GE	53.09%
H-Meta-NAS + 5×5 Pool + GE + ANL	56.35%

pooling is required for the network to achieve a good accuracy in this few-learning setup. We then demonstrate using a case study in our evaluation of how a combination of these techniques can help H-Meta-NAS: the final searched model can have an up to 14.28% accuracy increase on the 5-way 1-shot Mini-ImageNet classification if using these optimization tricks. Table 2 also showed us that a well-tuned architecture can help MAML models overcome the over-fitting phenomenon in the few-shot learning setup.

Table 3: Results of Omniglot 20-way few-shot classification. We keep two decimal places for our experiments, and keep the decimal places as it was reported for other cited work. * reports a MAML replication implemented by Antoniou *et al.*.

Method	Size	MACs	Accuracy 1-shot	Accuracy 5-shot
Siamese Nets Koch et al. [2015]	$35.96M$	$1.36G$	99.2%	97.0%
Matching Nets Vinyals et al. [2016]	$225.91K$	$20.29M$	93.8%	98.5%
Meta-SGD Li et al. [2017]	$419.86K$	$46.21M$	$95.93\% \pm 0.38\%$	$98.97\% \pm 0.19\%$
Meta-NAS Elsken et al. [2020]	$100.00K$	-	$96.20\% \pm 0.16\%$	$99.20\% \pm 0.07\%$
MAML Finn et al. [2017]	$113.21K$	$10.07M$	$95.8\% \pm 0.3\%$	$98.9\% \pm 0.2\%$
MAML* (Replication from Antoniou et al. [2018])	$113.21K$	$10.07M$	$91.27\% \pm 1.07\%$	98.78%
MAML++ * Antoniou et al. [2018]	$113.21K$	$10.07M$	$\mathbf{97.65\% \pm 0.05\%}$	$\mathbf{99.33\% \pm 0.03\%}$
MAML++ (Local Replication)	$113.21K$	$10.07M$	$96.60\% \pm 0.28\%$	$99.00\% \pm 0.07\%$
H-Meta-NAS	$\mathbf{110.73K}$	$\mathbf{4.95M}$	$97.61 \pm 0.03\%$	$99.11\% \pm 0.09\%$

Table 4: Results of Mini-ImageNet 5-way classification. We use two decimal places for our experiments, and keep the decimal places of cited work as they were originally reported. T-NAS uses the complicated DARTS cell Lian et al. [2019], it has a smaller size but a large MACs usage.

Method	Size	MACs	Accuracy 1-shot	Accuracy 5-shot
Matching Nets Vinyals et al. [2016]	$228.23K$	$200.31M$	$43.44 \pm 0.77\%$	$55.31 \pm 0.73\%$
CompareNets Sung et al. [2018]	$337.95K$	$318.38M$	$50.44 \pm 0.82\%$	$65.32 \pm 0.70\%$
MAML Finn et al. [2017]	$\mathbf{70.09K}$	$57.38M$	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
MAML++ Antoniou et al. [2018]	$\mathbf{70.09K}$	$57.38M$	$52.15 \pm 0.26\%$	$68.32 \pm 0.44\%$
ALFA + MAML + L2F Baik et al. [2020]	$\mathbf{70.09K}$	$57.38M$	$52.76 \pm 0.52\%$	$71.44 \pm 0.45\%$
OFA Cai et al. [2019] (Local Replication) + MAML++	82.20K	33.11M	$51.32 \pm 0.07\%$	$68.22 \pm 0.12\%$
Auto-Meta Kim et al. [2018]	98.70K	-	$51.16 \pm 0.17\%$	$69.18 \pm 0.14\%$
BASE (Softmax) Shaw et al. [2018]	$1200K$	-	-	$65.4 \pm 0.7\%$
BASE (Gumbel) Shaw et al. [2018]	$1200K$	-	-	$66.2 \pm 0.7\%$
Meta-NAS * Elsken et al. [2020]	$100K$	-	$53.2 \pm 0.4\%$	$67.8 \pm 0.7\%$
T-NAS * Lian et al. [2019]	$24.3/26.5K$	$37.96/52.63M$	$52.84 \pm 1.41\%$	$67.88 \pm 0.92\%$
T-NAS++ * Lian et al. [2019]	$24.3/26.5K$	$37.96/52.63M$	$54.11 \pm 1.35\%$	$69.59 \pm 0.85\%$
H-Meta-NAS	$70.28K$	$\mathbf{24.09M}$	$\mathbf{57.36 \pm 1.11\%}$	$\mathbf{77.53 \pm 0.77\%}$

Evaluating H-Meta-NAS searched architectures Table 3 displays the results of H-Meta-NAS on the Omniglot 20-way 1-shot and 5-shot classification tasks. We match the size of H-Meta-NAS to MAML and MAML++ for a fair comparison. H-Meta-NAS outperforms all competing methods apart from the original MAML++. MAML++ uses a special evaluation strategy, it creates an ensemble of models with best validation-set performance. MAML++ then picks the best model from the ensemble based on support set loss and report accuracy on the query set. We then locally replicated MAML++ without this trick, and show that H-Meta-NAS outperforms it by a significant margin ($+1.01\%$ on 1-shot and $+0.11\%$ on 5-shot) with around half of the MACs ($4.95M$ compared to $10.07M$).

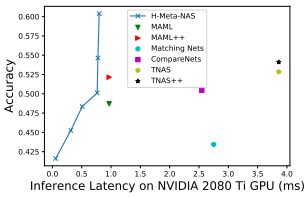


Figure 5: Target a GPU

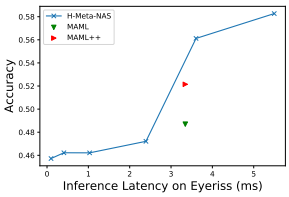


Figure 6: Target an ASIC

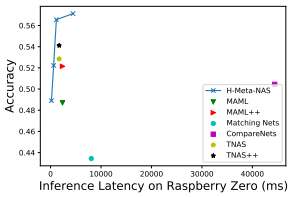


Figure 7: Target an IoT device

Table 4 shows the results of running the 5-way 1-shot and 5-shot Mini-ImageNet tasks, similar to the previous results, we match the size of searched networks to MAML, MAML++ and ALFA+MAML+L2F. Table 4 not only displays results on MAML methods with fixed-architectures, it also shows the performance of searched networks including Auto-Meta Kim et al. [2018], BASE Shaw et al. [2018] and T-NAS Lian et al. [2019]. H-Meta-NAS shows interesting results when compared to T-NAS and T-NAS++. H-Meta-NAS has much higher accuracy ($+3.26\%$ in 1-shot and 7.94% in 5-shot) and a smaller MAC count, but uses a greater amount of parameters. T-NAS and T-NAS++ use DARTS cells Liu et al. [2018]. This NAS cell contains a complex routing of computational blocks, making it not suitable for latency critical applications. We will demonstrate later how this design choice gives a worse on-device latency performance.

H-Meta-NAS for diverse hardware platforms and constraints In addition to using the model sizes as a constraint for H-Meta-NAS, we use various latency targets on various hardware platforms as the optimization target. Figure 5, Figure 6 and Figure 7 show how GPU, ASIC and IoT device latency can be used as constraints. The smaller model sizes of T-NAS do not provide a better run-time on GPU devices (Figure 5), in fact, T-NAS based models have the worst run-time on GPU devices due to the complicated dependency of DARTS cells. We only compare to MAML and MAML++ when running on Eyeriss due to the limitations of the ScaleSIM simulator Samajdar et al. [2018]. In Appendix J, we provide more latency *vs.* accuracy plots using various hardware platforms' latency as constraints and observe the same pareto dominance.

Table 5: Applying H-Meta-NAS to different NAS backbones/algorithms for the Mini-ImageNet 5-way 1-shot classification.

Method	Network Backbone	Inference Style	Size	MACs	Accuracy
MAML Finn et al. [2017]	VGG-based	Single Pass	$70.09K$	$57.38M$	$48.70 \pm 1.84\%$
MAML++ Antoniou et al. [2018]	VGG-based	Single Pass	$70.09K$	$57.38M$	$52.15 \pm 0.26\%$
Meta-Baseline Chen et al. [2020]	ResNet-based	Multi Pass	$12.44M$	$56.48G$	$63.17 \pm 0.23\%$
DeepEMD Zhang et al. [2020]	ResNet-based	Multi Pass	$12.44M$	$56.38G$	$65.91 \pm 0.82\%$
MetaNAS Elsken et al. [2020]	DARTS-cell-based	Single Pass	$1M$	-	$61.8 \pm 0.1\%$
H-Meta-NAS	VGG-based	Single Pass	$70.28K$	$24.09M$	$57.36 \pm 1.11\%$
H-Meta-NAS	ResNet-based	Single Pass	$70.62K$	$28.19M$	$64.67 \pm 2.03\%$

Table 6: Comparing the NAS search complexity with N tasks, H hardware platforms and C constraints. Search time is estimated for a deployment scenario with 500 tasks and 10 hardware-constraint pairs, estimation details are discussed in Appendix.

Method	Style	Hardware-aware	Search complexity	Search time (GPU hrs)
DARTS Liu et al. [2018]	Gradient-based, single task	No	$\mathcal{O}(THC)$	$\approx 10^6$
Once-for-all Cai et al. [2019]	Evolution-based, single task	Yes	$\mathcal{O}(N)$	$\approx 10^4$
TNAS & TNAS++ Lian et al. [2019]	Gradient-based, multi task	No	$\mathcal{O}(HC)$	$\approx 10^3$
H-Meta-NAS	Evolution-based, multi task	Yes	$\mathcal{O}(1)$	40

A more complex NAS backbone and a comparison to Metric-based MAML Table 5 shows how H-Meta-NAS performs with a more complicated NAS backbone. In previous experiments, we build the NAS on top of a VGG9 backbone since it is the architecture utilized in the MAML++ algorithm. To have a fair comparison, we did not manually pick a complex NAS backbone. However, in this section, we additionally show that **H-Meta-NAS can be applied with a more complicated backbone** and it shows better final accuracy as expected.

We compare the proposed approach with state-of-the-art Metric-based meta-learning methods Zhang et al. [2020], Chen et al. [2020]. There is a current trend that Metric-based MAML approaches are

becoming the mainstream in few-shot learning. Chen et al. shows that a simple cosine-similarity measurement can be used as a metric and outperforms standard Optimization-based approaches by a significant margin. In this experiment, we demonstrate that ***Metric-based methods suffer from a significant runtime overhead***. Optimization methods, on the other hand, have a significantly less MACs usage. Although using only a single inference pass (our method does not conduct inference runs on the support set when deployed), H-Meta-NAS shows competitive results with SOTA ***Metric-based methods while having a much smaller MACs usage (around*** $2000\times$***)***, showing that a well-tuned network architecture can help Optimization methods to close the accuracy gap. ***We hope this finding will encourage researchers in this field to look back into Optimization-based MAML***.

Search complexity and search time In Table 6, we show a comparison between H-Meta-NAS and various NAS schemes in the many-task many-device setup. Specifically, we consider a scenario with 500 tasks and 10 different hardware-constraint pairs. Our results in Table 6 suggest that H-Meta-NAS is the most efficient search method because of its low search complexity.

5 Conclusion

In this paper, we show H-Meta-NAS, a NAS method focusing on fast adaption of not only model weights but also model architectures in a many-task many-device few-shot learning setup. H-Meta-NAS outperforms a wide range of MAML baselines on a set of few-shot learning tasks. We study the effectiveness of H-Meta-NAS on different hardware systems and constraints, and demonstrate its superior performance on real devices using an orders of magnitude shorter search time compared to existing NAS methods.

Acknowledgements

Xitong Gao is supported in part by Shenzhen Science and Technology Innovation Commission (No. JCYJ20190812160003719).

References

A. Antoniou, H. Edwards, and A. Storkey. How to train your MAML. *arXiv preprint arXiv:1810.09502*, 2018.

S. Baik, M. Choi, J. Choi, H. Kim, and K. M. Lee. Meta-learning with adaptive hyperparameters. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20755–20765. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/ee89223a2b625b5152132ed77abbcc79-Paper.pdf`.

H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

F. P. Casale, J. Gordon, and N. Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.

T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, et al. TVM: An automated End-to-End optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.

Y. Chen, X. Wang, Z. Liu, H. Xu, and T. Darrell. A new meta-baseline for few-shot learning. *arXiv preprint arXiv:2003.04390*, 2020.

Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

T. Deleu, T. Würfl, M. Samiei, J. P. Cohen, and Y. Bengio. Torchmeta: A Meta-Learning library for PyTorch, 2019. URL `https://arxiv.org/abs/1909.06576`. Available at: https://github.com/tristandeleu/pytorch-meta.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

T. Elsken, B. Staffler, J. H. Metzen, and F. Hutter. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12365–12375, 2020.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

J. Guo, X. Zhu, C. Zhao, D. Cao, Z. Lei, and S. Z. Li. Learning meta face recognition in unseen domains. *CoRR*, abs/2003.07733, 2020. URL `https://arxiv.org/abs/2003.07733`.

Y. Guo, Y. Zheng, M. Tan, Q. Chen, J. Chen, P. Zhao, and J. Huang. NAT: Neural architecture transformer for accurate and compact architectures. *arXiv preprint arXiv:1910.14488*, 2019.

R. Hönig, Y. Zhao, and R. Mullins. DAdaQuant: Doubly-adaptive quantization for communication-efficient federated learning. In *International Conference on Machine Learning*, pages 8852–8866. PMLR, 2022.

J.-Y. Hsu, Y.-J. Chen, and H.-y. Lee. Meta learning for end-to-end low-resource speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7844–7848. IEEE, 2020.

J. Kim, S. Lee, S. Kim, M. Cha, J. K. Lee, Y. Choi, Y. Choi, D.-Y. Cho, and J. Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018.

G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Z. Li, F. Zhou, F. Chen, and H. Li. Meta-SGD: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

D. Lian, Y. Zheng, Y. Xu, Y. Lu, L. Lin, P. Zhao, J. Huang, and S. Gao. Towards fast adaptation of neural architectures with meta learning. In *International Conference on Learning Representations*, 2019.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

T. Munkhdalai and H. Yu. Meta networks. In *International Conference on Machine Learning*, pages 2554–2563. PMLR, 2017.

M. Nikolić, M. Mahmoud, A. Moshovos, Y. Zhao, and R. Mullins. Characterizing sources of ineffectual computations in deep learning networks. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 165–176. IEEE, 2019.

B. N. Oreshkin, P. Rodriguez, and A. Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123*, 2018.

A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH computer architecture news*, 45(2):27–40, 2017.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. 2016.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

A. Shaw, W. Wei, W. Liu, L. Song, and B. Dai. Meta architecture search. *arXiv preprint arXiv:1812.09584*, 2018.

D. So, Q. Le, and C. Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

J. Su, J. Faraone, J. Liu, Y. Zhao, D. B. Thomas, P. H. Leong, and P. Y. Cheung. Redundancy-reduced MobileNet acceleration on reconfigurable logic for ImageNet classification. In *International Symposium on Applied Reconfigurable Computing*, pages 16–28. Springer, 2018.

F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*, 2016.

K. Wang, X. Gao, Y. Zhao, X. Li, D. Dou, and C.-Z. Xu. Pay attention to features, transfer learn faster CNNs. In *International conference on learning representations*, 2019.

D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *CVPR*, 2019.

Y. Xu, L. Xie, X. Zhang, X. Chen, B. Shi, Q. Tian, and H. Xiong. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392*, 2020.

T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

C. Zhang, Y. Cai, G. Lin, and C. Shen. Deepemd: Few-shot image classification with differentiable earth mover's distance and structured classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12203–12213, 2020.

Y. Zhao, X. Gao, D. Bates, R. Mullins, and C.-Z. Xu. Focused quantization for sparse CNNs. *Advances in Neural Information Processing Systems*, 32, 2019a.

Y. Zhao, X. Gao, X. Guo, J. Liu, E. Wang, R. Mullins, P. Y. Cheung, G. Constantinides, and C.-Z. Xu. Automatic generation of multi-precision multi-arithmetic cnn accelerators for FPGAs. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 45–53. IEEE, 2019b.

Y. Zhao, D. Wang, D. Bates, R. Mullins, M. Jamnik, and P. Lio. Learned low precision graph neural networks. *arXiv preprint arXiv:2009.09232*, 2020a.

Y. Zhao, D. Wang, X. Gao, R. Mullins, P. Lio, and M. Jamnik. Probabilistic dual network architecture search on graphs. *arXiv preprint arXiv:2003.09676*, 2020b.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A Details of VGG9 and ResNet12 backbones

Table 7 and Table 8 show the NAS backbones of H-Meta-NAS. Clearly the ResNet-based NAS backbone is significantly more complicated. The kernel size search space is $\{1, 3, 5\}$. The channel expansion search space is $\{0.25, 0.5, 0.75, 1, 1.5, 2, 2.25\}$ for the VGG-based NAS backbone but $\{0.25, 0.5, 1, 1.5, 1.75, 2\}$ for the ResNet-based backbone. The reason for the modification in search space is because the GPU RAM limitation does not support an expansion size of 2.25 on the ResNet-based backbone. The activation search space contains $\{[' relu',' elu',' selu',' sigmoid',' relu6',' leakyrelu'\}$.

Table 7: Details of the VGG9 NAS backbone

Layer Name	Base channel counts	Stride
Layer0	64	2
Layer1	64	2
Layer2	64	2
Layer3	64	2

Table 8: Details of the ResNet NAS backbone

Layer Name	Base channel counts	Stride
Block0_Layer0	32	2
Block0_Layer1	32	1
Block0_Layer2	32	1
Block1_Layer0	64	2
Block1_Layer1	64	1
Block1_Layer2	64	1
Block2_Layer0	128	2
Block2_Layer1	128	1
Block2_Layer2	128	1
Block3_Layer0	256	2
Block3_Layer1	256	1
Block3_Layer2	256	1

B Tuning the decay process in pre-training strategy

As mentioned in Section 3.3 in the paper, we apply a progressive shrinking strategy to pre-training. We decay the probability of picking the largest sub-network gradually. Recall that the architectural sampling process $\alpha \sim p(\mathbb{A})$ will pick the largest network with a probability of p, and randomly pick a sub-network with a probability of $1 - p$. We apply an exponentially decay strategy to p:

$$p = p_e + (p_i - p_e) \times exp(-\alpha \times \frac{e - e_s}{e_m - e_s})) \tag{5}$$

p_e and p_i are the end and initial probabilities. e is the current number of epochs, and e_s is the starting epoch of applying this decaying process. α determines how fast the decay is. In our experiment, we pick $p_i = 1.0$ and $e_s = 30$, because the super-net reaches a relatively stable training accuracy at that point. We then start the decaying process, and evaluate different values of α in Table 9. The averaged accuracy is averaged across 100 randomly picked sub-networks on the $\mathcal{T}_{val}$ tasks. Based on these results, we picked $\alpha = 5$ for our later experiments.

C Latency predictor: an explanation

Figure 8 explains how the latency predictor work. The predictor builds on N profiled networks, and this style of sub-network profiling is infeasible on IoT devices.

Table 9: Tuning the decay factor α for pre-training on Mini-ImageNet 5-way 1-shot classification. Accuracy is averaged across 100 randomly picked sub-networks.

α	0.1	0.5	5	10	50
AVG ACCURACY	0.424	0.4145	0.5464	0.5323	0.4423

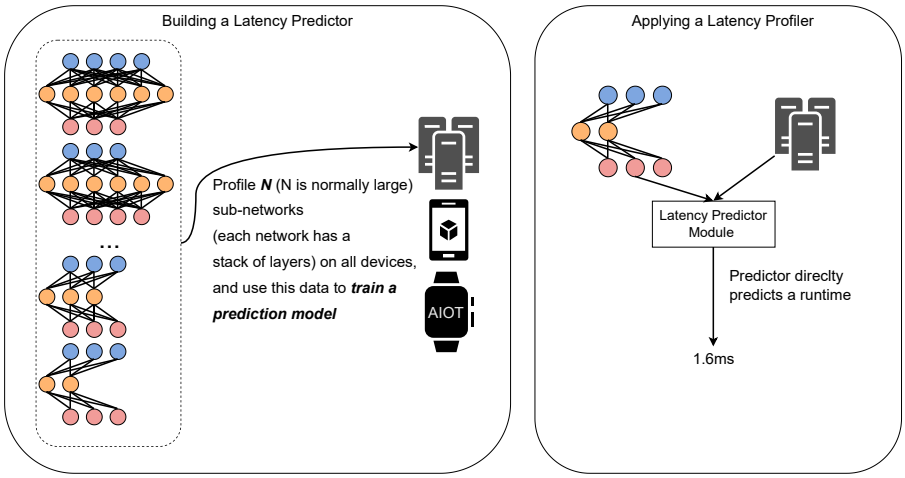


Figure 8: An illustration of how Latency Predictor works. The left side shows how the predictor is built by training on the run-time of ***a bunch of profiled sub-networks***. The right side shows this predictor then can predict a hardware-aware runtime for a whole network.

D Adaption algorithm and the hyper-parameter choices

Algorithm 1 details the adaption algorithm. In the $Mutate$ function, each architecture is ranked with the averaged loss across all sampled tasks, and 10% of the architectures with the lowest loss values are then used to perform a classic genetic algorithm mutation Whitley [1994]. The mutation will allow the top-performing architectures to have two randomly picked architectural choices being modified to another choice that is not the original one. The mutation function considers the original pool of architectures ($\mathbb{A}$) and their averaged loss values (L_a). The cost of each architecture can be computed by the pre-build hardware-specific hash-table $H_t(\mathcal{A})$. We then only mutate the subset in $\mathbb{A}$ that their hardware cost has satisfied the constraints $\{\mathcal{A}|\mathcal{A} \in \mathbb{A} \wedge H_t(\mathcal{A}) \leq C\}$. The mutation is to randomly pick two options in the entire architectural space and change them to other choices that are different from the original.

Algorithm 1 The adaption algorithm

Input: M, P, C, H_t
$\mathbb{A} = \text{Init}(P)$ ▷ Initialise a set of architectures with a size of P
for $i = 0$ **to** $M - 1$ **do**
  $L_a = \emptyset$
  $\mathbb{T}_s \sim p(\mathbb{T}_{val})$ ▷ Obtain a subset from the validation task set
  for $\mathcal{A} \in \mathbb{A}$ **do**
    $L_t = \emptyset$
    for $\mathcal{T} \in \mathbb{T}_s$ **do**
      $l = \mathcal{L}(\mathcal{T}, \mathcal{A})$ ▷ Compute loss
      $L_t = L_t \bigcup \{l\}$
    end for
    $L_a = L_a \bigcup \{mean(L_t)\}$ ▷ Collect averaged loss values across all tasks
  end for
  $\mathbb{A} = Mutate(\mathbb{A}, L_a, H_t, C)$ ▷ Mutate the architectures based on hardware constraints
end for

Table 10: Details of hardware systems experimented.

System	Device
Cloud	Nvidia GeForce RTX 2080 Ti
Mid-end CPU	Intel CPU
Mobile CPU	Raspberry Pi 4B
IoT	Raspberry Pi Zero
ASIC	Eyeriss Chen et al. [2016]

E Hardware devices

To demonstrate the hardware-awareness of our NAS algorithm, Table 10 summarises the hardware platform we used for our experiments. We deployed our networks to Raspberry Pi models using TVM Chen et al. [2018].

F H-Meta-NAS search configurations and hardware simulation

We mostly follow the experiment setup in MAML++ Antoniou et al. [2018]. In the pre-training stage, we train for 100 epochs, each epoch consists of 500 iterations. We also pick 600 tasks to be validation tasks. In the adaption stage, we randomly sample from the validation set, and pick 16 tasks to build a data slice for the architectures to traverse. In the final re-training stage of a searched architecture, we follow the strategy used in MAML++ Antoniou et al. [2018]. We then introduce the detailed special configurations for the datasets:

- Omniglot: We randomly split 1200 characters for training, and the rest is used for testing. The images are augmented with randomised rotation of multiples of 90 degrees.
- Mini-ImageNet: All images are down-sampled to 84×84.

We use the ScaleSim framework Samajdar et al. [2018] for simulating the Eyeriss Chen et al. [2016] accelerator. ScaleSim is an open-source cycle-accurate CNN simulator. The simulator has certain limitations with respect to the DRAM simulation, it could be advanced with an external DRAM simulator but will cause a large run-time. So we kept the original setup and the DRAM simulation would report a read/write bandwidth requirements. For simplicity, we assume these DRAM requirements are met. In addition, it is a well-known fact that cycle-accurate simulators are slow to execute. Due to this reason, we only launched the MAML and MAML++ networks in the ScaleSim simulator.

G T-NAS baseline results

We notice the model sizes of some baseline models (*e.g.* MAML and MAML++) reported in the original TNAS paper Lian et al. [2019] are different from our results in Table 3. We calculated the model sizes of these baselines using their official open-sourced implementations. T-NAS did not provide an implementation of their mentioned baselines in their official repository, so we cannot replicate their model size numbers. We have contacted the T-NAS authors regarding this issue.

H Additional results on FC100

FC100 is introduced by Oreshkin et al. and has 100 different classes from the CIFAR100 dataset Krizhevsky [2009].

In Table 11, we further demonstrate the effectiveness of the proposed H-Meta-NAS on the FC100 dataset. T-NAS did not report their model sizes on this task, and our results suggest that H-Meta-NAS achieves the best accuracy on both the 1-shot and 5-shot setups.

Table 11: Results of FC100 5-way few-shot classification. We keep two decimal places for our experiments, and keep the decimal places of cited work as they were originally reported.

METHOD	SIZE	ACCURACY 1-SHOT	ACCURACY 5-SHOT
MAML	$70.09K$	$38.1 \pm 1.7\%$	$50.4 \pm 1.0\%$
MAML++	$70.09K$	$38.7 \pm 0.4\%$	$52.9 \pm 0.4\%$
T-NAS	-	$39.7 \pm 1.4\%$	$53.1 \pm 1.0\%$
T-NAS++	-	$40.4 \pm 1.2\%$	$54.6 \pm 0.9\%$
H-META-NAS	**55.52K**	$\mathbf{43.29 \pm 1.22\%}$	$\mathbf{56.86 \pm 0.76\%}$

I Search time estimation

Due to the limited computing facilities available, we estimate the search time of DARTS Liu et al. [2018], Once-for-all Cai et al. [2019] and T-NAS Lian et al. [2019] in a multi-task multi-device setup. We take the search time reported in the original publications and multiply them by the appropriate scaling factors. For DARTs, we take the search time (4 GPU days = 96 GPU hours) and multiply it by $H \times T = 5000$. We additionally assume a linear scaling relationship between search time and input image sizes, so we multiply the total search time by $\frac{84 \times 84}{32 \times 32}$, this gives us in total a search time of around 10^6. We perform the same estimation for Once-for-all Cai et al. [2019] and T-NAS Lian et al. [2019].

J Latency-aware optimisation on more hardware platforms

In addition to using the model sizes as a constraint for H-Meta-NAS, we use various latency targets on various hardware platforms as the optimisation target. Figure 9 shows how GPU latencies can be used as constraints. T-NAS and T-NAS++ show a better performance on the size-accuracy plot in Figure 9. The smaller model sizes of T-NAS do not provide a better run-time on GPU devices (Figure 5), in fact, T-NAS based models have the worst run-time on GPU devices due to the complicated dependency of DARTS cells. We only compare to MAML and MAML++ when running on Eyeriss due to the limitations of the ScaleSIM simulator Samajdar et al. [2018].

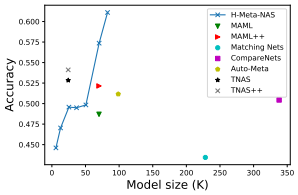


Figure 9: Target model sizes

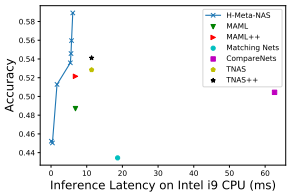


Figure 10: Target a mid-end CPU

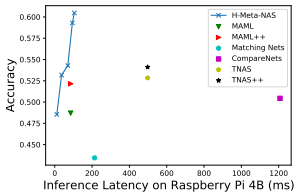


Figure 11: Target a low-end CPU

K Searched Pooling mechanism

In this section, we discuss the effect of automatically search for the pooling strategy in the proposed algorithm. The H-Meta-NAS algorithm used a fixed pooling strategy that is manually tuned, this is the same setup in other NAS algorithms (*e.g.* T-NAS Lian et al. [2019]). We add an additional search space of pooling (1×1, 3×3, 5×5 and 7×7).

Table 12 shows the results of searched pooling. Unsurprisingly, the network picked 5×5 pooling for both datasets, which is the same as pooling strategy obtained from our manual tunning. The automatic search for the pooling strategy should be more beneficial if our downstream tasks are of different image resolutions. Unfortunately, most of the standard benchmarks today in few-shot learning do not consider this scenario. If given the resolutions are the same for the meta-tasks and downstream tasks, an automatic search for the pooling design might not be necessary, since the number of pooling options are normally small.

Table 12: Searched pooling strategy. Experiments are executed with a model size constraint of $70K$ and $120K$ on the Mini-ImageNet 5-way 1-shot classification task and Omniglot 20-way 1-shot classification respectively.

METHOD	DATASET	SIZE	1-SHOT ACCURACY
H-META-NAS + 5×5 POOL + GE + ANL	MINI-IMAGENET	$67.2K$	56.35 ± 0.73
H-META-NAS + SEARCHED POOL + GE + ANL	MINI-IMAGENET	$69.3K$	56.42 ± 1.02
H-META-NAS + 5×5 POOL + GE + ANL	OMNIGLOT	$110.73K$	97.61 ± 0.03
H-META-NAS + SEARCHED POOL + GE + ANL	OMNIGLOT	$103.23K$	97.22 ± 0.92

L License of the assets

In our work, we utilized the following datasets/library/code listed in Table 13.

Table 13: Licenses of used assets.

DATASET/ALGORITHM/LIB NAMES	LICENSE
THE OMNIGLOT DATASET	MIT LICENSE
THE MINI-IMAGENET DATASET	MIT LICENSE
THE FC100 DATASET	APACHE V2 LICENSE
PYTORCH-META	MIT LICENSE
MAML++	MIT LICENSE

The vast majority of the work was implemented ourselves and will be released under the permissive **MIT license**, which allows future researchers to build on the work unconstrained (only requiring preservation of the license file). All dependencies of my library are similarly released under OSI[2]-approved licenses, allowing them all to be easily compiled and installed.

M Computation resources

Apart from the hardware devices used for profiling in the main paper. All experiments complete in $<$ 30 GPU-days on a four NVIDIA GeForce GTX 1080 Ti system with an Intel(R) Xeon(R) CPU E5-2620 v4 at 2.10GHz.

[2]https://opensource.org/licenses