Rapid Model Architecture Adaption for Meta-Learning

Yiren Zhao
yaz21@cam.ac.uk
University of Cambridge

Xitong Gao
xt.gao@siat.ac.cn
Shenzhen Institutes of Advanced

Ilia Shumailov
is410@cam.ac.uk
University of Cambridge

Nicolo Fusi
fusi@microsoft.com
Microsoft Research

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

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 [21, 37], language [13, 28] and graph data [36]. 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 [37] and language modeling [28]. 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 different task(T)-hardware(H) pairs with different constraints (C). The architectural engineering phase can be accomplished whether manually or by using an established NAS procedure. The major challenge is designing an efficient algorithmic method to overcome the quickly scaling $O(THC)$ 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 [12] and speech recognition [14]. 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 [10]. MAML then becomes a powerful and elegant approach for few-shot learning – its ability to quickly adapt to new tasks can potentially shrink the $O(THC)$ complexity illustrated in Figure 1 to $O(HC)$. In the meantime, hardware-aware NAS methods [3, 4, 33], *e.g.* the train-once-for-all technique [3], support deployments of searched models to fit to different hardware platforms with various latency constraints. These hardware-aware NAS techniques further reduce the search complexity from $O(THC)$ to $O(T)$ [4].

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 $O(THC)$ to $O(1)$, 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-parameterised 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 reuse of profiling information across different tasks.

In this paper, we make the following contributions:

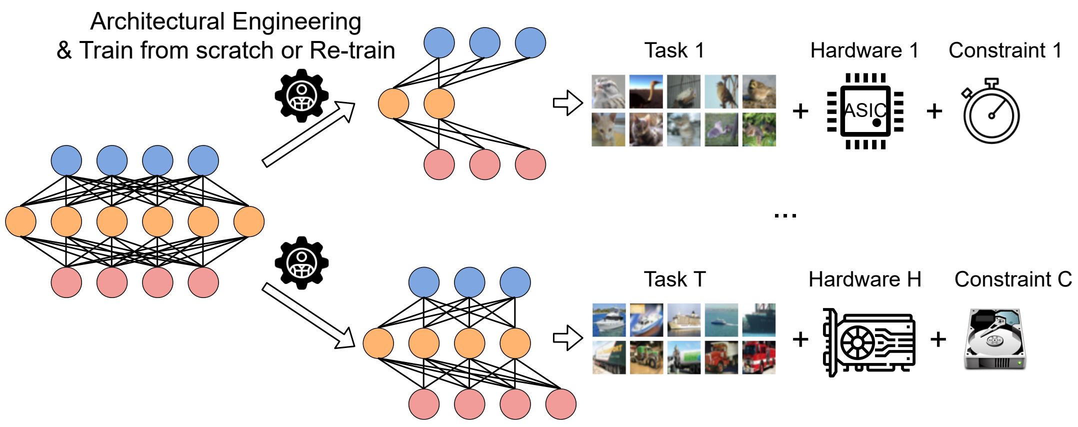

Figure 1. Deploying networks in a *many-task many-device* few-shot learning setup. This implies a large search complexity $O(THC)$.

- 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 $O(THC)$ to $O(1)$ 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 efficiency constraints (latency and model size), our latency-accuracy performance curve demonstrates a pareto dominance.
- We propose a task-agnostic layer-wise profiling strategy for the NAS. This profiling reduces the profiling run-time from around 10^5 hours to 1.2 hours when targeting hardware with limited capabilities (*e.g.* IoT devices).
- We show several tricks for the NAS algorithm, named Global Expansion and Adaptively Number of Layers respectively. These methods help the NAS to overcome the over-fitting problem in few-shot learning from the architectural perspective.

2 Related work

Few-shot learning in the MAML framework. Inspired by human's ability to learn from only a few tasks and generalise the knowledge to unseen problems, a meta learner is trained over a distribution of tasks with the hope of generalising its learned knowledge to new tasks [10].

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

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

Memory-based method utilises a memory-augmented neural network [11, 22] to memorise 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 utilises the cosine nearest-controid metric [6] and DeepEMD applies the Wasserstein distance [35]. Optimisation-based method, on the other hand, focuses on learning a good parameter initialisation (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 Optimisation-based method is Model-Agnostic Meta-Learning (MAML) [10]. 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 optimisation.

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. Meta-learning has been explored in key applications such as facial and speech recognition [12, 14] for mobile deices. Real-life deployments on these devices resemble a *many-task many-device* scenario, where learning on each user's data is a few-shot learning task and different hardware platforms represent different types of under-deployment devices. Memory-based and Metric-based meta-learning methods are then challenged by the hardware or latency constraints: Memory-based methods need additional storage space (at least double) and Metric-based approaches use multiple inference runs (at least two) for a single image classification. In this work, we then focus solely on an Optimisation-based approach because of the runtime concern outlined above. The proposed NAS method utilises the simple yet effective MAML++ framework: after adapting the model to new tasks, MAML++ executes exactly one inference run for a single test sample without additional memory usage.

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 [25, 37]. 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 optimise a set of probabilistic priors that are associated with architectural choices [5, 21]. Although these probabilistic priors can be made latency-aware [32, 33], 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 [3, 4], 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 optimised for a specific hardware target but also constrained by a pre-defined latency budget [3]. 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 $O(HC)$ to $O(1)$.

Several NAS methods are proposed under the MAML framework [15, 20, 27], these methods successfully reduce the search complexity from $O(T)$ to $O(1)$. However, some of these methods do not show significant performance improvements compared to carefully designed MAML methods (*e.g.* MAML++) [15, 27]. In the meantime, some of these MAML-based NAS methods follow the Gradient-based approach and operate on complicated cell-based structures [20]. 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 2 Phase 1. The objective of this process is to optimise 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 (Phase 3 in Figure 2). 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 details in later sections.

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

Architecture space. H-Meta-NAS considers a search space composed of different kernel sizes, 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 [10] and MAML++ [1] used a VGG9 model architecture. The details of this backbone are in Appendix.

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 is $3 \times 7 \times 6 = 126$ 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 a 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 more complex NAS backbone. We also studied an alternative ResNet12-based NAS backbone, that has approximately 2×10^{24} possible sub-networks.

Super-net meta-training strategy. As illustrated by prior work [3], 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 architectural 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 exponentially decay strategy to p:

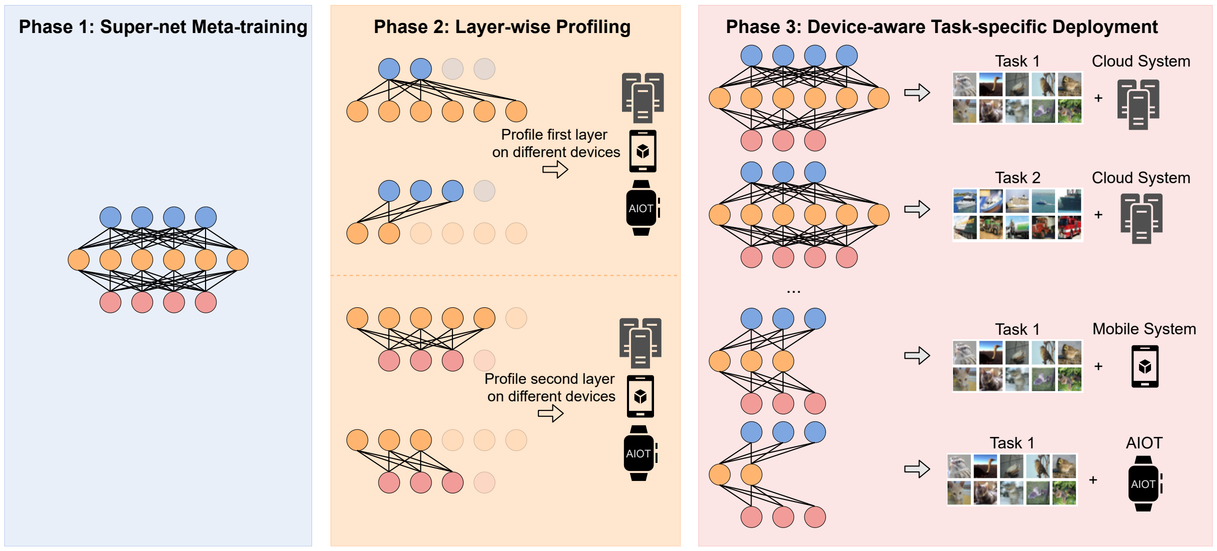
Phase 1: Super-net Meta-training
Phase 2: Layer-wise Profiling
Profile first layer on different devices
AIOT
Profile second layer on different devices
AIOT
Phase 3: Device-aware Task-specific Deployment
Task 1
Cloud System
Task 2
Cloud System
...
Task 1
Mobile System
Task 1
AIOT
AIOT

Figure 2. An overview of the three main stages of the proposed H-Meta-NAS algorithm.

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

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. 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.

Layer-wise profiling. Hardware-aware NAS needs the run-time of sub-networks on the targeting hardware to guide the search process [3, 33]. 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 averaged 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 ($O(H)$). Existing hardware-aware NAS schemes build predictive methods to estimate the run-time of sub-networks [3, 33] and have a relatively significant error. We show in our evaluation, performing an exact profiling can be done with a low cost if allowing a per-layer profiling strategy.

Adaption strategy. The adaption strategy uses a genetic algorithm [31] to pick the best suited sub-network with respective to a given hardware constraint, the full algorithm is detailed in Appendix. 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 the for the genetic algorithm. The genetic algorithm has a pool size P and number of iterations M, we demonstrate the optimal values are $P = 100, M = 200$ in our evaluation.

NAS backbone design. 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 sample many times. We would like to explore on the architectural space to help models to 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 3 further illustrate that GE and ANL can allow a much smaller model compared to existing NAS backbones. We then demonstrate using a case study in our evaluation 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 optimisation tricks.

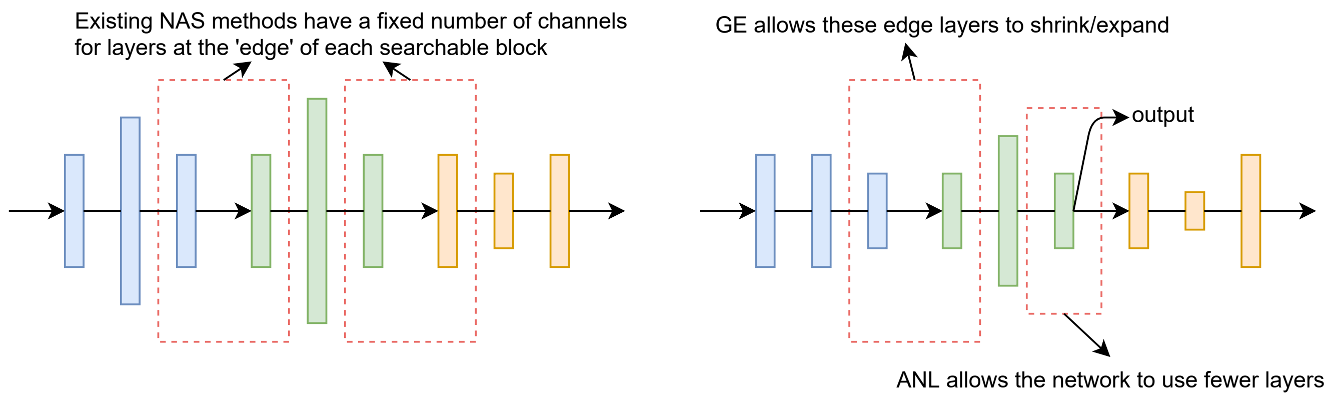
Existing NAS methods have a fixed number of channels for layers at the 'edge' of each searchable block
GE allows these edge layers to shrink/expand
output
ANL allows the network to use fewer layers

Figure 3. 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.

Table 1. Details of hardware systems experimented with H-Meta-NAS.

System	Device
Cloud	Nvidia GeForce RTX 2080 Ti
Mid-end CPU	Intel CPU
Mobile CPU	Raspberry Pi 4B
IoT	Raspberry Pi Zero
ASIC	Eyeriss [7]

4 Evaluation

We evaluate H-Meta-NAS in a few-shot learning setup. 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 [8].

Omniglot is a handwritten digits recognition task, containing 1623 samples [18]. We use the meta train/validation/test splits used Vinyals *et al.* [30]. 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 [9], the splits are taken from Ravi *et al.*[24].

FC100 is introduced by Oreshkin et al., the datasets has 100 different classes from the CIFAR100 dataset [17].

Table 1 details the systems and representative devices considered. Our Appendix contains a more detailed explanation of the specs of each hardware device. We use the ScaleSIM cycle-accurate simulator [26] for the Eyeriss [7] accelerator. Details about this simulation and more information with respect to the datasets and search configurations are in our Appendix.

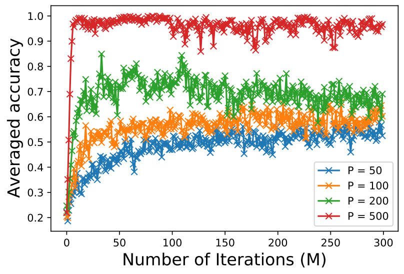

Figure 4. The effect of different pool size (P) and different number of iterations (M).

The effect of pool sizes. We identify the following two hyper-parameters that can potentially affect the performance in the adaption stage, namely the number of iterations M and the pool size P, and then run an hyper-parameter analysis in Figure 4. The full adaption algorithm making 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 4 shows that the accuracy convergence is reached after 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 accuracies 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.

Evaluating pooling, GE and ANL. Our results in Table 2 suggest that a correct pooling strategy, GE and ANL can

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%

Table 3. 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

change the NAS backbone to allow the search space to reach much smaller models and thus provide a better accuracy. In addition, Table 2 also illustrates that 5×5 pooling is necessary for a higher accuracy. We hypothesize this is because a relatively large fully-connected layer after the pooling is required for the network to achieve a good accuracy in this few-learning setup.

Latency predictor vs. layer-wise latency profiling. We re-implemented the latency predictor in OFA [3] to illustrate how a layer-wise profiling and look-up method is a perfect match in our learning scenario. We pick 16K training samples and 10K validation samples to train and test the latency predictor, which is the same as 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 3, 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 [3, 4]. 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,

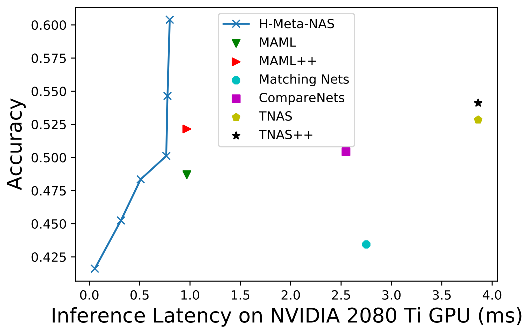

Figure 5. Targeting a GPU

Figure 6. Applying H-Meta-NAS with GPU latency as optimisation targets.

making them an infeasible approach in real life. For instance, the total time consumed by latency predictor is infeasible to execute on Pi Zero (last row in Table 3). 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 O(H) times.

Most existing layer-wise look-up approaches consider at most mobile systems as targeting platforms [33, 34]. 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 for constrained hardware systems.

Evaluating H-Meta-NAS searched architectures. 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 [15], BASE [27] and T-NAS [20]. H-Meta-NAS shows interesting results when compared to T-NAS and T-NAS++. H-Meta-NAS has a 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 [21]. 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. We also show how H-Meta-NAS work with FC100 and Omniglot in Appendix.

H-Meta-NAS for diverse hardware platforms and constraints. In addition to using the model sizes as a constraint

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 [20], it has a smaller size but a large MACs usage.

Method	Size	MACs	Accuracy 1-shot	Accuracy 5-shot
Matching Nets [30]	228.23K	200.31M	43.44 ± 0.77%	55.31 ± 0.73%
CompareNets [29]	337.95K	318.38M	50.44 ± 0.82%	65.32 ± 0.70%
MAML [10]	**70.09K**	57.38M	48.70 ± 1.84%	63.11 ± 0.92%
MAML++ [1]	**70.09K**	57.38M	52.15 ± 0.26%	68.32 ± 0.44%
ALFA + MAML + L2F [2]	**70.09K**	57.38M	52.76 ± 0.52%	71.44 ± 0.45%
OFA [3] (Local Replication) + MAML++	82.20K	33.11M	51.32 ± 0.07%	68.22 ± 0.12%
Auto-Meta [15]	98.70K	-	51.16 ± 0.17%	69.18 ± 0.14%
BASE (Softmax) [27]	1200K	-	-	65.4 ± 0.7%
BASE (Gumbel) [27]	1200K	-	-	66.2 ± 0.7%
T-NAS * [20]	24.3/26.5K	37.96/52.63M	52.84 ± 1.41%	67.88 ± 0.92%
T-NAS++ * [20]	24.3/26.5K	37.96/52.63M	54.11 ± 1.35%	69.59 ± 0.85%
H-Meta-NAS	70.28K	**24.09M**	**57.36 ± 1.11%**	**77.53 ± 0.77%**

for H-Meta-NAS, we use various latency targets on various hardware platforms as the optimisation target. Figure 6 shows how GPU latencies can be used as constraints. The smaller model sizes of T-NAS do not provide a better run-time on GPU devices (Figure 7), 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 [26]. In our Appendix, we provide more latency *vs.* accuracy plots using various hardware platforms' latency as constraints and observe the same pareto dominance shown in Figure 6.

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 shows a Pareto dominance when compared to a wide range of MAML baselines and other NAS results. We study the effectiveness of H-Meta-NAS on a wide variety of hardware systems and constraints, and demonstrate its superior performance on real-hardware devices using an orders of magnitude shorter search time compared to existing NAS methods.

References

[1] Antreas Antoniou, Harrison Edwards, and Amos Storkey. 2018. How to train your maml. *arXiv preprint arXiv:1810.09502* (2018).

[2] Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. 2020. Meta-Learning with Adaptive Hyperparameters. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 20755–20765. https://proceedings.neurips.cc/paper/2020/file/ee89223a2b625b5152132ed77abbcc79-Paper.pdf

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

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

[5] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. 2019. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116* (2019).

[6] Yinbo Chen, Xiaolong Wang, Zhuang Liu, Huijuan Xu, and Trevor Darrell. 2020. A new meta-baseline for few-shot learning. *arXiv preprint arXiv:2003.04390* (2020).

[7] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.

[8] Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. 2019. Torchmeta: A Meta-Learning library for PyTorch. https://arxiv.org/abs/1909.06576 Available at: https://github.com/tristandeleu/pytorch-meta.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*. PMLR, 1126–1135.

[11] Spyros Gidaris and Nikos Komodakis. 2018. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4367–4375.

[12] Jianzhu Guo, Xiangyu Zhu, Chenxu Zhao, Dong Cao, Zhen Lei, and Stan Z. Li. 2020. Learning Meta Face Recognition in Unseen Domains. *CoRR* abs/2003.07733 (2020). arXiv:2003.07733 https://arxiv.org/abs/2003.07733

[13] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. 2019. Nat: Neural architecture transformer for accurate and compact architectures. *arXiv preprint arXiv:1910.14488* (2019).

[14] Jui-Yang Hsu, Yuan-Jui Chen, and Hung-yi Lee. 2020. Meta learning for end-to-end low-resource speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 7844–7848.

[15] Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. 2018. Auto-meta: Automated gradient based meta learner search.

arXiv preprint arXiv:1806.06927 (2018).

[16] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2. Lille.

[17] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[18] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338.

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

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

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[22] Tsendsuren Munkhdalai and Hong Yu. 2017. Meta networks. In *International Conference on Machine Learning*. PMLR, 2554–2563.

[23] Boris N Oreshkin, Pau Rodriguez, and Alexandre Lacoste. 2018. Tadam: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123* (2018).

[24] Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning. (2016).

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

[26] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[27] Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. 2018. Meta architecture search. *arXiv preprint arXiv:1812.09584* (2018).

[28] David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International Conference on Machine Learning*. PMLR, 5877–5886.

[29] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. 2018. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1199–1208.

[30] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080* (2016).

[31] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.

[32] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*.

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

[34] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 285–300.

[35] Chi Zhang, Yujun Cai, Guosheng Lin, and Chunhua Shen. 2020. 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*. 12203–12213.

[36] Yiren Zhao, Duo Wang, Xitong Gao, Robert Mullins, Pietro Lio, and Mateja Jamnik. 2020. Probabilistic dual network architecture search on graphs. *arXiv preprint arXiv:2003.09676* (2020).

[37] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

A A more complex NAS backbone

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 utilised in the MAML++ algorithm. For the purpose of having a fair comparison, we did not manually pick a complex NAS backbone. However, we demonstrate, in this section, that H-Meta-NAS can be applied with a more complicated backbone and it shows better final accuracy as expected. The trained accuracy of searched networks using ResNet12 reaches a 7.31% higher accuracy compared the original VGG9 backbone. In addition, we compare the proposed approach with state-of-the-art Metric-based meta-learning methods [6, 35]. 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 20×).

B 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 paris. Our results in Table 6 suggest that H-Meta-NAS is the most efficient search method because of its low search complexity.

C 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 $\{['relu', 'elu', 'selu', 'sigmoid', 'relu6', 'leakyrelu'\}$.

D 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:

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

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.

E 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 [31]. 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 $\{\mathcal{A} | \mathcal{A} \in \mathbb{A} \land H_t(\mathcal{A}) \leq C\}$. 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: $M, P, C, H_t$
$\mathbb{A} = Init(P)$ {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 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 [10]	VGG-based	Single Pass	$70.09K$	$57.38M$	$48.70 \pm 1.84\%$
MAML++ [1]	VGG-based	Single Pass	$70.09K$	$57.38M$	$52.15 \pm 0.26\%$
Meta-Baseline [6]	ResNet-based	Multi Pass	$12.44M$	$56.48G$	$63.17 \pm 0.23\%$
DeepEMD [35]	ResNet-based	Multi Pass	$12.44M$	$56.38G$	$65.91 \pm 0.82\%$
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 [21]	Gradient-based, single task	No	$O(THC)$	$\approx 10^6$
Once-for-all [3]	Evolution-based, single task	Yes	$O(N)$	$\approx 10^4$
TNAS & TNAS++ [20]	Gradient-based, multi task	No	$O(HC)$	$\approx 10^3$
H-Meta-NAS	Evolution-based, multi task	Yes	$O(1)$	40

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 ResNet12 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

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

F H-Meta-NAS search configurations and hardware simulation

We mostly follow the experiment setup in MAML++ [1]. 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++ [1]. 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 [26] for simulating the Eyeriss [7] 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 Additional results on Omniglot

Table 10 displays the results of H-Meta-NAS on the Omniglot 20-way 1-shot and 5-shot classification tasks. We match the

Table 10. 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.*[16].

Method	Size	MACs	Accuracy 1-shot	Accuracy 5-shot
Siamese Nets [16]	35.96M	1.36G	99.2%	97.0%
Matching Nets [30]	225.91K	20.29M	93.8%	98.5%
Meta-SGD [19]	419.86K	46.21M	95.93% ± 0.38%	98.97% ± 0.19%
MAML [10]	113.21K	10.07M	95.8% ± 0.3%	98.9% ± 0.2%
MAML* (Replication from [1])	113.21K	10.07M	91.27% ± 1.07%	98.78%
MAML++ * [1]	113.21K	10.07M	**97.65% ± 0.05%**	**99.33% ± 0.03%**
MAML++ (Local Replication)	113.21K	10.07M	96.60% ± 0.28%	99.00% ± 0.07%
H-Meta-NAS	**110.73K**	**4.95M**	97.61 ± 0.03%	99.11% ± 0.09%

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 (4.95G compared to 10.07G).

H Additional results on FC100

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 ± 1.7%	50.4 ± 1.0%
MAML++	70.09K	38.7 ± 0.4%	52.9 ± 0.4%
T-NAS	-	39.7 ± 1.4%	53.1 ± 1.0%
T-NAS++	-	40.4 ± 1.2%	54.6 ± 0.9%
H-META-NAS	55.52K	43.29 ± 1.22%	56.86 ± 0.76%

I T-NAS baseline results

We notice the model sizes of some baseline models (*e.g.* MAML and MAML++) reported in the original TNAS paper [20] 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.

J Search time estimation

Due to the limited computing facilities available, we estimate the search time of DARTS [21], Once-for-all [3] and T-NAS [20] 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 $\frac{84\times84}{32\times32}$, this gives us in total a search time of around 10^6. We perform the same estimation for Once-for-all [3] and T-NAS [20].

K 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 8 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 8. The smaller model sizes of T-NAS do not provide a better run-time on GPU devices (Figure 7), in fact, T-NAS based models have the worst run-time on GPU devices due to the complicated dependency of DARTS cells. Figure 13 illustrates the performance of H-Meta-NAS on different CPU devices and an ASIC hardware. The details of these hardware are described in the main paper. In Figure 13, H-Meta-NAS shows a better Pareto-frontier performance compared to a range of baselines and searched models. We only compare to MAML and MAML++ when running on Eyeriss due to the limitations of the ScaleSIM simulator [26]. In our Appendix, we provide more latency *vs.* accuracy plots using various hardware platforms' latency as constraints and observe the same pareto dominance shown in Figure 13. Our

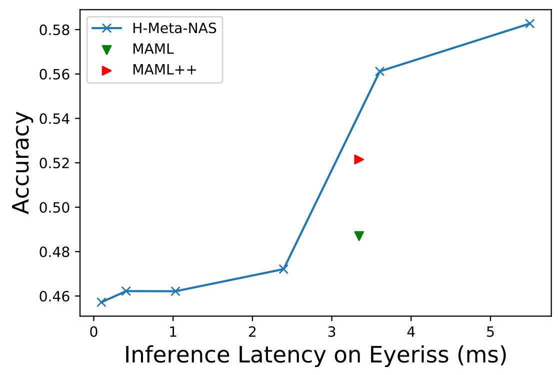

Figure 9. Targeting an ASIC accelerator

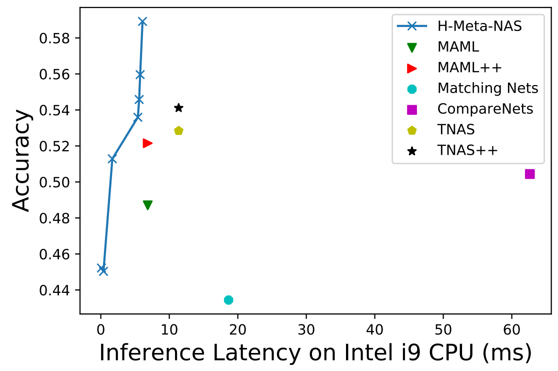

Figure 10. Targeting a mid-end CPU

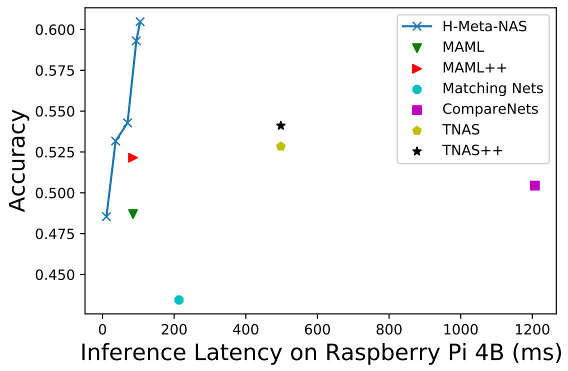

Figure 11. Targeting a low-end CPU

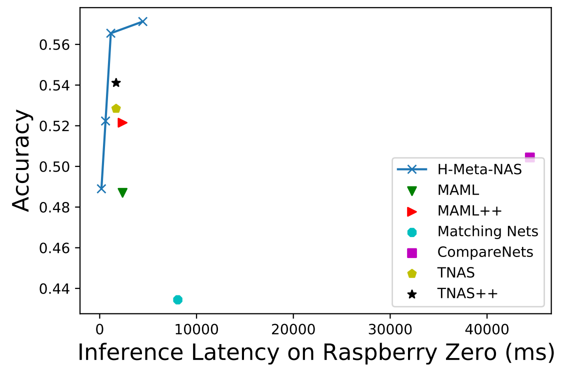

Figure 12. Targeting an IoT device

results in Figure 13 demonstrate that H-Meta-NAS consistently generates more efficient models compared to various MAML-based methods.

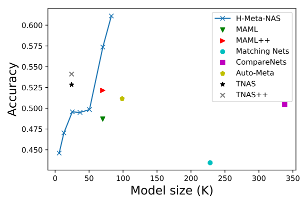

Figure 7. Targeting model sizes

Figure 8. Applying H-Meta-NAS with model sizes as optimisation targets.

L License of the assets

In our work, we utilised the following datasets/library/code:

Table 12. 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