HaeNAS: Hardware-Aware Efficient Neural Architecture Search via Zero-Cost Proxy

Yaodanjun Ren[†], Chen Chen[†], Zhengwei Qi[†]
[†]*Shanghai Jiao Tong University*
[§]*State Key Laboratory of Mathematical Engineering and Advanced Computing*

Abstract

The practical use of advanced DNN models is hindered by limited hardware resources and high computation demands. Neural Architecture Search (NAS) is becoming a default technique which automatically discovers architectures that are competitive with handcraft ones. However, existing methods only prioritize accuracy and overlook hardware-related factors. To address this, we introduce HaeNAS (Hardware-aware efficient NAS), which considers both accuracy and computation cost on specific hardware platforms. The search space of HaeNAS consists of several stages, each allowing different convolution kernels, layer numbers, layer widths, and operation types. We use a data-driven approach to predict the latency and energy consumption on target hardware, and we improve the zero-cost proxy based on network pruning research to speed up the NAS process. With these techniques, HaeNAS finds a target network within 160 GPU hours, which achieves 80.7% top-1 accuracy on the ImageNet, with a latency of 10.4ms and energy consumption of 931mJ.

Keywords: neural architecture search, convolutional neural network, evolution algorithm, neural network acceleration

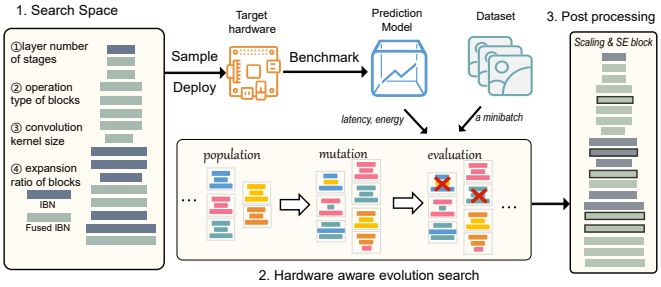

Figure 1. The framework of HaeNAS.

1 Introduction

Deep neural network models have achieved impressive results in various applications [8, 11]. However, running AI applications on specific hardware must not only achieve high accuracy but also meet latency and energy constraints. Thus, the high computation demands required by high accuracy hinder the practical application of DNN models in real-world scenarios, especially on embedded or edge devices.

Many recent studies have attempted to improve the efficiency of CNN models by utilizing software-related or hardware-related optimization methods [13]. In the hardware domain, a popular approach is to leverage the specific hardware characteristics of hardware devices such as GPUs, FPGAs, and NPUs to enhance the inference and training speed of DNNs. However, for a given DNN model, hardware optimization is constrained by the computation and storage resources inherent in the hardware itself. Optimization solutions on the software side have also received attention from researchers, such as model pruning and quantization. For example, network pruning methods usually reduce the parameter size by identifying a criterion of significance for each component(i.e., parameter, filter, layer), such as the magnitude of the weight [11]. Recently, pruning-at-initialization methods [4, 5] have attracted a lot of interest as they can reduce the training cost while achieve similar accuracy.

Neural Architecture Search (NAS) has become a popular approach for automatically discovering competitive neural network architectures, surpassing those designed by humans [2]. Existing methods [1, 12] primarily focus on accuracy while neglecting other hardware-related factors, such as latency, energy, and memory. However, the optimality of DNN models depends on both the model architecture and target devices. To address this, we propose a hardware-aware NAS method, aiming to find the Pareto-optimal architectures within the search space. There are several challenges. Firstly, different platforms require different operations, and efficiency metrics are not simple functions of MACs (multiply–accumulate operations). Therefore, it is necessary to customize the search space and strategy for a specific hardware platform. Secondly, customizing search options for hardware will inevitably increase the complexity of the search space, rendering the search cost more difficult to bear.

Inspired by the above observations, we propose a NAS method to find a resource-efficient network on the target hardware. As illustrated in figure 1, our hardware-aware efficient NAS (HaeNAS) consists of three steps. First, we design a flexible search space that takes into account the hardware features. Second, we employ zero-cost proxies to implement hardware-aware evolution search. Finally, post processing, such as compound scaling and adding SE block, are utilized. Our contributions are as follows:

DOI reference number: 10.18293/SEKE2023-116

- We propose a lightweight NAS methodology which can quickly explore a wide search space, considering resource constraints including latency and energy.
- We designed a flexible search space that allows for selection among different depths, widths, convolution kernels, and operation types in each stage.
- We evaluate models with direct hardware metrics such as latency and energy consumption, instead of proxy metrics such as model size and FLOPs.
- We improve the pruning-at-initialization methods which only use a minibatch of data, leading to a significant reduction in the search cost.

2 Methodology

In this section, we present the technical details of HaeNAS, which searches efficient CNN models for specific hardware platforms in a data-driven way.

2.1 Problem formulation

Our goal is to find CNN models with both high accuracy and low resource consumption. Resource consumption, such as latency and energy, during model inference is critical, as they can directly affect the user experience and the number of model executions. Previous NAS approaches often optimize for indirect metrics, such as FLOPS, MACs, and model sizes. We consider inference latency and energy consumption by running and tracking CNN models on the target devices and incorporating these hardware metrics into our objective function. The NAS problem is formulated as

$$\underset{a\in\mathcal{A}}{\text{maxmize}} \quad \text{Score}(a) \times \left[\frac{LAT(a)}{L}\right]^{\lambda} \times \left[\frac{EC(a)}{E}\right]^{\omega} \tag{1}$$

where Score(a) measures the accuracy of architecture a, λ and ω respectively represent the importance of latency and energy consumption. λ could be defined as equation 2. And ω is defined in a similar way.

$$\omega = \begin{cases} \alpha, & if\ LAT(a) \le L \\ \beta, & otherwise \end{cases} \tag{2}$$

Given an architecture search space $\mathcal{A}$, HaeNAS aims to find an optimal architecture $a \in \mathcal{A}$ that after training it can achieve the optimal accuracy and resource consumption trade-off. Therefore, in our work, we focus on three factors of the problem: (1) the architecture search space $\mathcal{A}$, (2) the objective function that considers latency and energy consumption, and (3) a lightweight search algorithm that scores architectures before training.

2.2 Search space design

The effectiveness of NAS algorithms is directly dependent on search space design. Ideally, it should encompass numerous excellent candidate architectures while avoiding unnecessary complexity. Predefined macro architecture and layer-level search strategy have been shown to reduce search cost without compromising model performance [9]. NASNet [15], for instance, improved search efficiency by 7 times and achieved higher accuracy compared with the previous work [14] by employing suitable macro architecture design.

Table 1. The macro architecture of the search space, where TBS means to be searched.

Input resolution $H_i \times W_i$	Operator O_i	Kernel $K_i \times K_i$	Expansion e_i	Layer L_i
$224 \times 224 \times 3$	$Conv3 \times 3$	-	-	1
$112 \times 112 \times 16$	TBS	{3, 5, 7}	{3, 4, 6}	1
$112 \times 112 \times 16$	TBS	{3, 5, 7}	{3, 4, 6}	{2, 3, 4}
$56 \times 56 \times 24$	TBS	{3, 5, 7}	{3, 4, 6}	{2, 3, 4}
$28 \times 28 \times 40$	TBS	{3, 5, 7}	{3, 4, 6}	{2, 3, 4}
$14 \times 14 \times 80$	TBS	{3, 5, 7}	{3, 4, 6}	{2, 3, 4}
$14 \times 14 \times 112$	TBS	{3, 5, 7}	{3, 4, 6}	{2, 3, 4}
$7 \times 7 \times 160$	TBS	{3, 5, 7}	{3, 4, 6}	1
$7 \times 7 \times 960$	$Conv1 \times 1\&Pooling\&FC$	-	-	-

HaeNAS adopts a macro architecture based on the MobileNets [12] and searches in the level of stage. As shown in Table 1, HaeNAS's search space is divided to 9 stages, where the first and last stages are fixed. The first stage contains a regular convolution layer with a 3x3 kernel and stride of 2. The last stage contains an average pooling layer and two convolution layers, which is proposed by MobileNetv3 [12] to save cost without sacrificing accuracy. The middle 7 stages are searchable, while the input resolution and stride of each stage is fixed. HaeNAS searches the setting of stages, including operation type, kernel size, stage width, and stage depth.

The inverted bottleneck (IBN) block with DWConv, which is proposed by MobileNets [12], is widely. It has fewer parameters and MACs, making it more suitable for mobile devices. However, the theoretical computational complexity (e.g. MACs or FLOPs) does not often necessarily correspond to inference speed. The inference latency may be proportional to the MACs of the model on one platform, while being proportional to the memory access on another platform. Therefore, when designing a model, it is crucial to select a model that aligns with the platform's features.

Many current NAS method, such as Once-for-all and MnasNet, only consider IBN block when designing search spaces [1]. This severely limits the flexibility of the resulting models and fails to provide models that are better suited to hardware platform. To address this, we propose searching for convolution operation type in addition to kernel size, stage depth, and width. As shown in Fig 2, besides IBN block, the search space of HaeNAS also includes fused inverted bottleneck layers (Fused-IBN). The Fused-IBN block replaces the combination of DWConv and pointwise Conv layers in IBN block with a regular Conv. Although this increases the computation cost, it reduces the I/O overhead of intermediate results.

2.3 Hardware aware evolution search

HaeNAS improves the aging evolution (AE) algorithm proposed by AmoebaNet [7], and incorporates hardware awareness in fitness evaluation. The procedure of Hardware-Aware Aging Evolution (HAE) algorithm is shown in Algorithm 1.

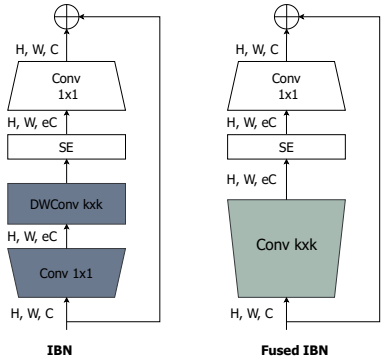

Figure 2. The structure of IBN block and fused-IBN block.

We use queue to save population participating in evolution and use set to save all excellent individuals in the history. HAE algorithm adds young individuals from the right and removes old individuals from the left of the queue based on the FIFO principle. The *history* set only stores the model architecture without saving weights, thereby avoiding occupying too much memory. HAE inherits the "age" concept proposed by AE [7] to ensure a regular elimination and update of the population. Individuals with the longest existence time, rather than those with the worst performance, will be eliminated. This approach avoids the situation where elite individuals produce a large number of offspring, which would compromise the diversity of the population. Furthermore, the HAE algorithm prevents being trapped in local optima by assigning a probability for poor individuals to generate offspring through random selecting candidate parents. The HAE algorithm also maintains a faster convergence rate by preserving a subset of superior individuals in the population.

Algorithm 1: Hardware-aware aging evolution

Output: candidate with the highest fitness
population ← empty queue; ▷ current population
history ← ∅ ; ▷ save all the excellent models
while | *population* |< P **do**
  model.arch ← RandArch() ;
  model.fit ← HaeFitness(model.arch) ;
  add model to *population* queue ;
  add model to *history* set ;
end
while | *history* |< C **do**
  sample ← ∅ ; ▷ candidate parents
  while | *sample* |< S **do**
    candidate ← random member from *population*;
    add *candidate* to *sample*;
  end
  parent ← highest-fitness member in *sample*;
  children.arch ← HaeMutation(parent.arch);
  children.fit ← HaeFitness(children.arch);
  add highest-fitness child to *history*;
  add highest-fitness child to *population*;
  remove the oldest from *population*;
end

Some work [1, 13] achieves hardware awareness by including latency as part of the loss function, while HaeNAS achieves it more directly by incorporating hardware metrics into the fitness function of the evolution algorithm. As shown in equation 1, we adopt a custom weighted product approach to evaluate fitness by aligning accuracy, latency, and energy consumption metrics.

To evaluate the fitness of candidate individuals, it is necessary to obtain their accuracy scores. In AmoebaNet [7], candidate models are trained for 50 epochs under the same setting to obtain their accuracy on CIFAR-10. However, it is very expensive to train a model on ImageNet, which is the target dataset of our HaeNAS. To reduce the training cost, some NAS algorithms, such as ProxylessNAS [1], adopt methods like training supernet and inheriting supernet weights for subnet. However, these method still require significant computation cost. Inspired by pruning-at-initialization methods[4], HaeNAS proposes to rank candidate networks using zero-cost metrics instead of accuracy. While accuracy is obtained after tens of epoches of training, the score of the zero-cost metric is obtained by a single forward/backward propagation, thus saving the search cost.

We improve the following pruning-at-initialization methods, extending them to scoring the entire candidate network. These metrics were previously used at the granularity of a single neuron (e.g. a parameter or a channel), now we adapt them by sum up every neuron's score to get the score of entire model. Here, $\mathcal{L}$ represents the loss function of a candidate model with weight parameters θ. $\odot$ represents the Hadamard product, which refers to the element-wise multiplication of two matrices with the same dimensions. The *grad_norm* means we sum the Euclidean norm (L2) of the gradients after a single minibatch. The *snip* [4] is proposed by Lee et al, which measures the change in loss function when a specific operation or parameter is removed. The *grasp* [10] is similar to *snip*, while it measures the change in gradient norm instead of loss function. The *jacob_conv* [5] is proposed by Mellor et al, which captures the correlations between activations in the network when presented with different inputs. A lower correlation is indicative of better performance in distinguishing between different input classes.

$$\mathcal{S}_{grad\_norm} = \sqrt{\sum \left(\frac{\partial \mathcal{L}}{\partial \theta}\right)^2} \tag{3}$$

$$\mathcal{S}_{snip}(\theta) = \left|\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right| \tag{4}$$

$$\mathcal{S}_{grasp}(\theta) = -\left(H\frac{\partial \mathcal{L}}{\partial \theta}\right) \odot \theta \tag{5}$$

$$S_{jacob\_conv} = -\sum_{i=1}^{N} \left[\log\left(\sigma_{J,i} + k\right) + \left(\sigma_{J,i} + k\right)^{-1}\right] \tag{6}$$

Another issue is the value of λ, ω, L (i.e. target inference latency), E (i.e. target energy consumption) in the fitness function. Figure 3 presents the fitness curve under two different sets of (α, β) values. For illustration purposes, the inference latency T is set to 80ms and energy consumption is not considered. The upper curve applies a soft constraint on the target inference latency with $(\alpha = -0.05, \beta = -0.05)$, while the lower curve imposes a hard constraint with $(\alpha = -0, \beta = -0.7)$. A hard constraint can prevent the model from violating the inference latency constraint by rapidly decreasing the fitness value, but the fitness score would depend solely on the accuracy when the latency constraint is not exceeded. A smoother adjustment of the fitness function value is more beneficial for balancing accuracy and hardware metrics. Therefore, the HaeNAS algorithm adopts a soft-constrained fitness function.

An empirical rule for choosing λ and ω is to ensure that the Pareto-optimal solutions have similar fitness value. In the EfficientNets [6], doubling the latency typically comes with a 3% relative accuracy improvement. Specifically, given two models m_1 and m_2 with the same energy cost, and considering only inference latency and accuracy, if model m_1 has an inference latency of l and an accuracy score of s, while model m_2 has an inference latency of $2l$ and an accuracy score of $(1+3\%)s$, the fitness value for them should be roughly equal. Thus, the value of α is approximately -0.05. The situation is similar for energy cost. Therefore, in the subsequent experiments, unless otherwise specified, the HaeNAS algorithm sets the scaling factors for latency and energy to -0.05.

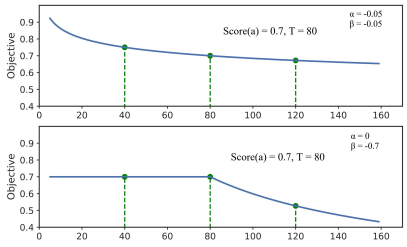

Figure 3. Objective function defined by equation 1

2.4 Post processing

Modern series models such as MobileNets [12] and EfficientNets [9] typically include multiple models that vary in size and accuracy to meet different resource requirements. HaeNAS intentionally imposes stricter constraints on the target inference latency T and target energy consumption E in the fitness function. We employ compound scaling [9] to simultaneously increase the model's depth, width, and input resolution. Compound scaling achieves higher precision return than scaling a single factor alone by allocating scaling coefficients for each factor in a balanced way. We perform a grid search to find scaling factors for depth, width, and input resolution under different multiples of computational cost. Additionally, the proposed method employs manual adjustments. Firstly, it restricts the maximum input resolution to 380 to avoid excessive training and inference costs. Secondly, more network layers are added in the later stages, as aggressive expansion of depth is unnecessary in the early stages.

To enhance the accuracy, HaeNAS further incorporates a lightweight attention module: squeeze-and-excitation(SE) block [3]. It applies attention to channels. Given an input feature map, the SE block first performs global average pooling for each channel. Then, the squeeze operation captures channel correlations through two cascaded fully connected layers. Finally, channel-wise multiplication is performed between the activation values and the input feature map. To avoid introducing too much computation cost, we selectively remove unimportant SE blocks. The standard deviation of the activation values across different images on each channel is calculated. If the standard deviation is small, it indicates that the SE block does not help distinguish which channel is more important, so the SE block can be removed.

3 Evaluation

3.1 Experimental setups

Datasets & Evaluation Metrics. We evaluate HaeNAS on image classification task. We use ImageNet-1K dataset, which consists of 1.2 million training images and 50,000 validation images, covering 1000 categories. In addition to the classification accuracy, HaeNAS evaluates the inference latency on Xeon CPU and GTX 3080 GPU, and the inference energy consumption on GTX 3080 GPU and Raspberry Pi 4.

Hardware Platforms. We conduct experiments on three different hardware platforms. The first one is a Xeon E5-2650 CPU, which is a server CPU. As it is in a stable power supply environment, we only focus on its inference latency. The second one is a GTX 3080 GPU, which is a high-performance GPU for both training and inference. We use the nvidia-smi utility to measure its energy consumption. The third one is a Raspberry Pi 4, which is a widely used edge development board. We use an FNIRSI-FNB58 power meter to track its voltage and current, enabling us to obtain power consumption information.

Latency and Energy Prediction Models. HaeNAS employs multi-layer perceptron (MLP) models to predict inference latency and energy consumption. To train the prediction models, 1000 random models are sampled from the search space. To avoid the impact of the DNN model's computation startup and the randomness in the computation process, we adopt the approach of warm-up and multiple runs to obtain an average value. Meanwhile, different devices have different computing parallelism, and appropriate loads need to be set when measuring hardware metrics, which is achieved by selecting an appropriate batch size.

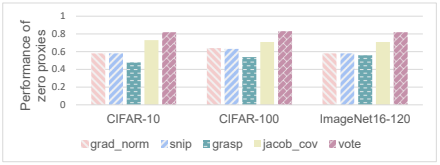

Figure 4. Performance of zero-cost proxies.

3.2 Main results

Train-free proxy for lightweight NAS. The effectiveness of adopting zero-cost metrics as the evaluation strategy of HaeNAS needs to be confirmed. We use NAS-Bench-201 as the benchmark model set and evaluate the effectiveness with Spearman's rank correlation coefficient. The Spearman's rank correlation coefficient measures the order relationship between variables, rather than the numerical relationship.

The NAS-Bench-201 provides accuracy information for three datasets: CIFAR-10, CIFAR-100, and ImageNet-16-120. Figure 4 presents the performance results of zero-cost proxies. In terms of a single criterion, jacob_conv achieves the highest score. It captures the model's ability to differentiate between different inputs, i.e., classification, based on activation values, and thus has considerable potential. Following closely behind are grad_norm and snip, both of which involve gradient values, with Spearman's correlation coefficients around 0.6. The calculation of loss and the update of parameters based on the loss are called gradient propagation. Thus, the gradient value to some extent reflects the model's learning direction and learning ability.

Due to the fact that the selected evaluation metrics have different focuses, HaeNAS utilizes a majority voting scheme to decide which individuals to keep during each evolution round. Specifically, jacob_conv, grad_norm, and snip are jointly considered to determine the candidates to be preserved. Moreover, for recording fitness scores and selecting potential parents, the value of jacob_conv is used.

Results under different computational resource constraints. The searching results of HaeNAS for base models on different platforms are shown in Figures 5. The letters denote the corresponding platform, with "G" representing GPU, "C" representing CPU, and "E" representing edge device Raspberry Pi 4. Different colors indicate different convolution kernel sizes, and F-IBN and IBN are used to distinguish operation types. The number inside the rectangle represents the expansion ratio, and the number outside the rectangle represents the stage depth.

Different hardware platforms result in architectures with distinct characteristics when searched by HaeNAS. For instance, HaeNAS-G tends to choose less layers and F-IBN block, with a preference for large kernel and big expansion ratio in the early stages. HaeNAS-C tends to choose more layers and smaller expansion ratio. Meanwhile, HaeNAS-E tends to select IBN block, which result in less computation cost. Overall, all three models prefer larger kernel sizes and retain SE block in the later stages.

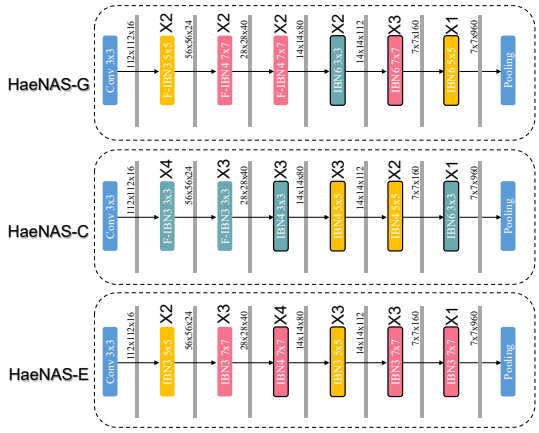

Figure 5. Visualization of the searched architectures.

Table 2. The comparison of HaeNAS and other models

Model Name	Top-1	Top-5	AutoML	Latency Aware	Energy Aware	Search cost (GPU hour)
ResNet-50	76.1%	92.09	Handcraft	N	N	-
MobileNetv2	72.0%	91.0%	Handcraft	N	N	-
MobileNetv3-L	75.2%	92.4%	Handcraft	N	N	-
NasNet-A	74.7%	91.97%	RL	N	N	48,000
MnasNet	75.6%	92.6%	RL	Y	N	40,000
ProxylessNAS	75.1%	92.5%	Gradient	Y	N	200
HaeNAS-G	78.3%	93.3%	Evolution	Y	Y	≈160
HaeNAS-C	78.1%	93.2%	Evolution	Y	Y	≈160
HaeNAS-E	77.6%	93.0%	Evolution	Y	Y	≈160

Accuracy and search cost. Table 2 presents the accuracy result of HaeNAS and other mainstream models, both handcraft or automaticallly designed. Compared to MobileNetv2, HaeNAS models have improved accuracy on the ImageNet dataset by 4.6% to 6.3% while maintaining similar inference time.

Compared to other autoML methods, the HaeNAS based on zero-cost metrics has a significant advantage in search cost. Compared to reinforcement learning methods, the required search cost of HaeNAS is reduced by nearly 300 times. Even compared to the ProxylessNAS based on gradient and super network parameter sharing, HaeNAS achieves 1.25 times computation cost savings. These benefits are mainly due to the use of zero-cost proxy metrics that reduce unnecessary additional training.

Hardware metrics. HaeNAS applies compound scaling to the base models obtained by the hardware-aware evolutionary algorithm. Figures 6 shows the accuracy-latency comparisons of the HaeNAS-C and HaeNAS-G models after scaling, with M/L denoting the scaled models. HaeNAS-C-L can achieve 80.1% Top-1 accuracy at a speed of 137ms on the CPU, and the latency is 86% of that of EfficientNet at the same accuracy. On 3080 GPU, the inference latency of HaeNAS-G models are only 44% to 49% of that of EfficientNets at the same accuracy. We believe this is due to the inclusion of F-IBN block in the search space, which is more conducive to the parallel performance of GPU-like hardware.

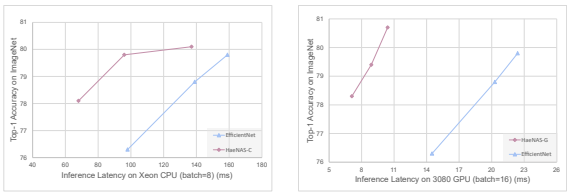

Figure 6. The latency comparison of HaeNAS models and EfficientNets on CPU and GPU.

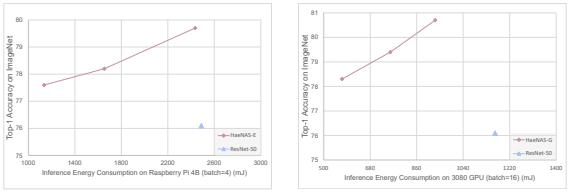

Figure 7. The energy consumption comparison of HaeNAS models and ResNet-50 on Raspberry Pi 4 and GPU.

Figure 7 presents a comparison of the energy efficiency between HaeNAS and ResNet-50 on edge devices and GPU, where the vertical axis represents the Top-1 accuracy and the horizontal axis represents the average inference energy consumption measured in millijoules (mJ). On Raspberry Pi 4, the inference latency of HaeNAS-E is only 67% of ResNet-50, and the inference energy consumption is only 45% of ResNet-50. This is due to the use of IBN block in HaeNAS-E, which effectively reduces computation cost. On 3080 GPU, HaeNAS achieved an average inference power reduction of 80% to 89% compared to ResNet-50, but an energy consumption reduction of 49% to 80%. The energy optimization mainly comes from the improvement in inference latency. The GPU frequency adjustment strategy is more conservative, resulting in a strong positive correlation between energy consumption and latency. On the CPU platform, the frequency adjustment is more aggressive, which weakens the correlation between energy consumption and latency, leading to a trade-off between latency and energy consumption for better latency performance.

4 Conclusion

We present HaeNAS, a hardware-aware efficient neural architecture search framework. It includes three steps. First, it designs the search space based on hardware characteristics and predefine the macro architecture of models. Second, in order to save search cost, HaeNAS proposes to use zero cost proxies and hardware-aware evolution algorithm to search candidates for target devices. The actual target device latency and energy consumption of sampled networks are used to train prediction models. Third, compound scaling and selective SE block are adopted to find models that strike a balance between accuracy and hardware metrics. The accuracy of the models found by HaeNAS on ImageNet can be comparable to mainstream models, both handcraft and automatically designed. The latency and energy consumption comparison on all target hardware platforms (Xeon CPU, 3080 GPU, and Raspberry Pi 4) confirms the effectiveness of the proposed HaeNAS methodology.

Acknowledgments

This work was supported by the Open Project Program of the State Key Laboratory of the Mathematical Engineering and Advanced Computing (2020A10), the National NSF of China (NO. 62141218), and Shanghai Key Laboratory of Scalable Computing and Systems.

References

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

[2] Krishna Teja Chitty-Venkata and Arun K Somani. 2022. Neural architecture search survey: A hardware perspective. *Comput. Surveys* 55, 4 (2022), 1–36.

[3] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.

[4] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).

[5] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. 2021. Neural architecture search without training. In *International Conference on Machine Learning*. PMLR, 7588–7598.

[6] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*. PMLR, 4095–4104.

[7] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 33. 4780–4789.

[8] Sergei Shcherban, Peng Liang, Zengyang Li, and Chen Yang. 2021. Multiclass classification of four types of UML diagrams from images using deep learning. In *Proc. of the 33rd International Conference on Software Engineering & Knowledge Engineering (SEKE). KSI.* 57–62.

[9] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[10] Chaoqi Wang, Guodong Zhang, and Roger Grosse. 2020. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376* (2020).

[11] Xiao-Jie Wang, WenBin Yao, and Huiyuan Fu. 2019. A Convolutional Neural Network Pruning Method Based On Attention Mechanism.. In *SEKE*. 343–452.

[12] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. 2021. Mobiledets: Searching for object detection architectures for mobile accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3825–3834.

[13] Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, et al. 2020. SkyNet: a hardware-efficient method for object detection and tracking on embedded systems. *Proceedings of Machine Learning and Systems* 2 (2020), 216–229.

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

[15] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.