Double-Win NAS: Towards Deep-to-Shallow Transformable Neural Architecture Search for Intelligent Embedded Systems

Xiangzhong Luo[1], Di Liu[2], Hao Kong[1], Shuo Huai[1], and Weichen Liu[1]*
[1]Nanyang Technological University [2]Norwegian University of Science and Technology

ABSTRACT

Thanks to the evolving network depth, convolutional neural networks (CNNs) have achieved impressive performance across various intelligent embedded scenarios towards embedded intelligence. Nonetheless, this trend also leads to degraded hardware efficiency as the network evolves deeper and deeper. In contrast, shallow networks exhibit superior hardware efficiency, which, unfortunately, suffer from inferior accuracy. To tackle this dilemma, we establish the first deep-to-shallow transformable neural architecture search (NAS) paradigm, namely Double-Win NAS (DW-NAS), which is dedicated to automatically exploring deep-to-shallow transformable networks to marry the best of both worlds. Extensive experiments on two NVIDIA Jetson intelligent embedded systems clearly show the superiority of DW-NAS over previous state-of-the-art methods.

1 INTRODUCTION

Convolutional neural networks (CNNs) have empowered a myriad of intelligent embedded scenarios towards embedded intelligence [1], such as on-device object detection/tracking [2] and immersive AR/VR [3]. In the era of deep learning, "*deeper + wider = better*" has been empirically deemed as the rule of thumb to design networks with promising accuracy [4–6]. Despite its promise, this empirical rule requires substantial human expertise to manually explore the optimal network structure [7]. To overcome such limitations, recent network design practices have shifted from *manual* to *automated*, also known as neural architecture search (NAS), which is dedicated to automatically exploring novel network structures [8–12]. Among them, differentiable NAS [8] has dominated recent success in the field of NAS, thanks to its strong search efficiency [7].

The dominant success of differentiable NAS [8] has subsequently sparked a plethora of **H**ard**W**are-aware **D**ifferentiable **NAS** (HW-DNAS) methods to search for hardware-friendly networks [13–22]. Specifically, previous state-of-the-art (SOTA) HW-DNAS methods [13–22] feature sufficient network depth to explore deep networks in order to maintain decent accuracy [23]. However, the resulting deep networks often exhibit degraded hardware efficiency [24]. To remedy this issue, we may simply re-engineer previous SOTA HW-DNAS methods to explore shallow networks instead. However, the resulting shallow networks, despite their superior hardware efficiency, suffer from inferior accuracy since the attainable accuracy highly relies on sufficient network depth [4]. This dilemma reveals that previous SOTA HW-DNAS methods can only win either accuracy or hardware efficiency and thus cannot deliver an aggressive accuracy-efficiency win-win. To tackle this dilemma, we further turn back to the following ***counterintuitive*** question:

> *Can we explore deep networks to first win accuracy, which then can be equivalently transformed into their shallow counterparts to win hardware efficiency and finally enable an aggressive* ***win-win****?*

To investigate the above question, we establish the first-of-its-kind deep-to-shallow transformable HW-DNAS paradigm, namely Double-Win NAS (DW-NAS), pioneering to draw insights from deep-to-shallow transformable networks to marry the best of both worlds. As shown in Figure 1, DW-NAS strives to explore deep networks to first win strong accuracy, which then can be equivalently transformed into their shallow counterparts to win significant hardware efficiency, and more importantly, without accuracy loss. To summarize, this paper makes the following novel contributions:

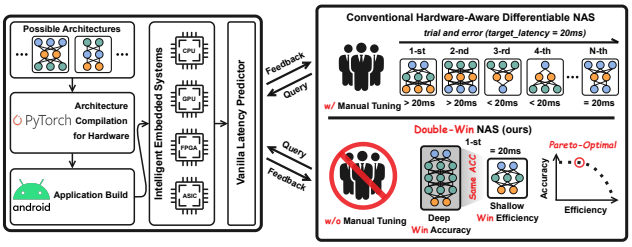

Figure 1: An intuitive overview of the proposed Double-Win NAS.

- **Fundamentals.** To the best of our knowledge, DW-NAS is the first deep-to-shallow transformable HW-DNAS paradigm, which opens up a fresh NAS perspective and also provides a *holistic* solution for *designing* and *training* transformable networks to aggressively win both accuracy and hardware efficiency.
- **Framework.** DW-NAS features a novel hybrid transformable search space to achieve deep-to-shallow transformable search, which is seamlessly integrated with an efficient vanilla latency predictor and an effective sandwich-inspired differentiable search algorithm to navigate the optimal transformable network around the specified latency constraint. Besides, we also introduce hybrid transformable training to unleash the attainable accuracy of the searched transformable network, which, once well trained, can be equivalently transformed into its shallow counterpart to largely boost the on-device efficiency without accuracy loss.
- **Evaluations.** Extensive experiments are conducted to evaluate DW-NAS on two popular NVIDIA Jetson intelligent embedded systems (i.e., Xavier and Nano), which clearly show the superiority of DW-NAS over previous SOTA NAS approaches in terms of both accuracy and hardware efficiency. For example, DW-Net-Xavier-20ms achieves +0.8% higher accuracy on ImageNet than FBNet-C [17], while also maintaining ×1.3 speedup on Xavier.

2 BACKGROUND

2.1 Preliminaries on Differentiable NAS [8]

Let $O = \{o_n\}_{n=1}^{N}$ denote the operator space with N operator candidates. As shown in DARTS [8], an over-parameterized network dubbed supernet is first initialized. The supernet has L searchable layers, each of which has N operator candidates $\{o_n\}_{n=1}^{N}$. To relax the discrete search space to be continuous, a set of architecture parameters $\alpha \in \mathbb{R}^{L\times N}$ are also assigned to the supernet (see Figure 5). Finally, we can formulate the l-th layer of the supernet as follows:

$$x_{l+1} = \sum_{n=1}^{N} p_l^n \cdot o_n(x_l), \text{ s.t., } p_l^n = \frac{\exp(\alpha_l^n)}{\sum_{n'=1}^{N}\exp(\alpha_l^{n'})} \text{ and } o_n \in O \quad (1)$$

where x_l is the input data of the l-th layer. Thanks to the above continuous relaxation, both architecture parameters α and network

* This research is partially supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG94/23), and partially supported by Nanyang Technological University, Singapore, under its NAP (M4082282/04INS000515C130).

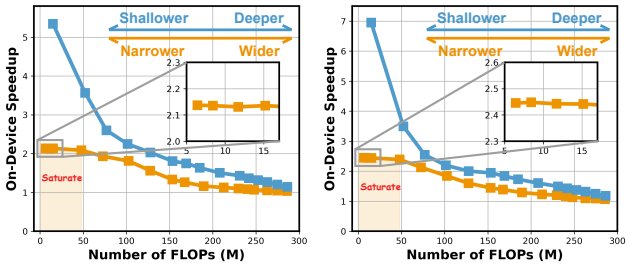

Figure 2: Shallow vs. narrow networks on Xavier (*left*) / Nano (*right*).

weights w can be optimized with gradient descent. In light of this, DARTS leverages the following bi-level differentiable optimization strategy to alternate the optimization process of α and w:

$$\underset{\alpha}{\text{minimize}} \ \mathcal{L}_{valid}(w^*(\alpha), \alpha), \ \text{s.t.}, \ w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \quad (2)$$

where $\mathcal{L}_{train}(\cdot)$ and $\mathcal{L}_{valid}(\cdot)$ are the training and validation loss functions. Once the above bi-level differentiable optimization converges, we can discretize the searched architecture according to the learned architecture parameters α. This discretization process reserves the strongest operator in each searchable layer and eliminates the remaining operators, where the strength of o_n in the l-th searchable layer is defined as $\exp(\alpha_l^n)/\sum_{n'=1}^{N}\exp(\alpha_l^{n'})$ [8, 17].

2.2 Observations and Motivations

Observation I. *Multiple consecutive linear layers can be equivalently transformed into one single linear layer.* Without loss of generality, we consider the following two consecutive linear layers:

$$Y = W_1 X + B_1 \ \text{ and } \ Z = W_2 Y + B_2 \quad (3)$$

where X is the input data. Furthermore, W_1, W_2, B_1, and B_2 are the weight and bias parameters of the above two consecutive linear layers. Taken together, we can re-formulate Eq (3) as follows:

$$Z = W_2(W_1 X + B_1) + B_2 = (W_1 W_2)X + (W_2 B_1 + B_2) \quad (4)$$

This indicates that the above two linear layers can be transformed into one single linear layer with $W^* = W_1 W_2$ and $B^* = W_2 B_1 + B_2$, which can also maintain the same output Z. Note that Eq (3)-(4) can be generalized to all linear layers, such as convolutional (Conv), batch normalization (BN), and fully-connected (FC) layers [25].

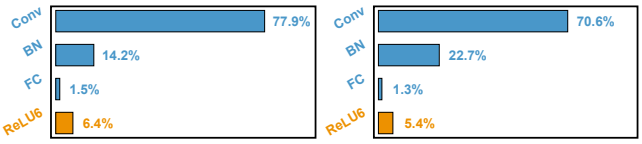

Figure 3: Layer-wise latency profiling results on Xavier and Nano.

Observation II. *The linear layers dominate the latency on target hardware.* To show this, we profile MobileNetV2 [5] on Xavier and Nano with an input batch size of 8, where the inter-layer communications and data movements are ignored for the sake of simplicity [25]. The profiling results are illustrated in Figure 3, which shows that the linear layers, including Conv, BN, and FC layers, account for over 93% of the total latency on both Xavier and Nano.

Observation III. *Hardware-friendly networks should be shallow rather than narrow.* To this end, we take MobileNetV2 as an example. Specifically, we leverage layer-wise and channel-wise pruning to gradually trim down the network depth and width to derive shallow and narrow networks. As shown in Figure 2, under similar FLOPs, shallow networks exhibit much better speedups than their narrow counterparts on both Xavier and Nano. Besides, we also find that, for narrow networks, their speedups may saturate at large pruning ratios, whereas shallow networks can consistently deliver enhanced speedups. These findings clearly demonstrate the merits of shallow networks in pursuit of superior hardware efficiency [24, 25].

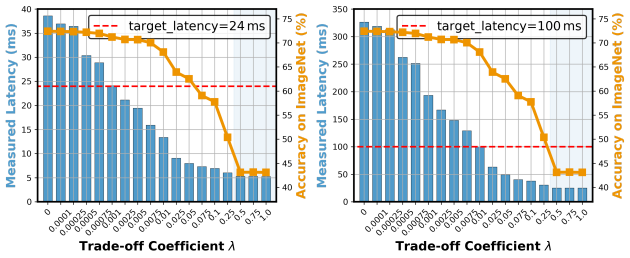

Figure 4: Architecture search results under $\lambda \in [0, 1]$ on Xavier (*left*) / Nano (*right*), where the accuracy is trained on ImageNet for 50 epochs.

Observation IV. *Previous HW-DNAS methods must repeat multiple search runs to navigate the required architecture.* As discussed in [23], previous HW-DNAS methods typically feature the following multi-objective optimization to explore hardware-efficient architectures:

$$\underset{\alpha}{\text{minimize}} \ \mathcal{L}_{valid}(w^*(\alpha), \alpha) + \lambda \cdot LAT(\alpha) \quad (5)$$

where $LAT(\cdot)$ is the on-device latency and $\lambda \geq 0$ is a constant to control the trade-off magnitude between accuracy and latency. In fact, the above multi-objective optimization can end up with hardware-efficient architecture with strong accuracy-efficiency trade-offs [15], but only when λ is properly selected. To show such limitations, we leverage FBNet [17] as the search engine to perform various search experiments under $\lambda \in [0, 1]$, where the specified latency constraint is set to 24 ms on Xavier and 100 ms on Nano. As shown in Figure 4, λ can trade off accuracy and latency, which, unfortunately, is quite sensitive. As a result, to derive the required architecture, we have to repeat multiple search runs to manually tune λ through trial and error [15], which is inflexible and also incurs huge search cost.

Motivations. As discussed in Observation I-III, deep networks with consecutive linear layers can be equivalently transformed into their shallow counterparts to win substantial hardware efficiency without accuracy loss. Nonetheless, this is infeasible since linear and non-linear layers alternate in order to maintain strong accuracy as shown in [4–6]. This motivates us to investigate deep-to-shallow transformable networks that feature consecutive linear layers. With this in mind, we establish the first deep-to-shallow transformable HW-DNAS paradigm dubbed DW-NAS, pioneering to explore deep-to-shallow transformable networks to marry the best of both deep and shallow networks towards an aggressive accuracy-efficiency win-win. Furthermore, in contrast to previous SOTA HW-DNAS methods that have to repeat multiple search runs (see Observation IV), DW-NAS strives to navigate the optimal transformable network around the specified latency constraint in one single search, which can deliver considerable search efficiency and flexibility.

3 DOUBLE-WIN NAS

In this section, we first introduce Double-Win NAS (DW-NAS) and then discuss the relationships with previous SOTA NAS methods.

3.1 Hybrid Transformable Search Space

Following recent HW-DNAS practices [16–18], we build the hybrid transformable search space $\mathcal{A}$ based on MobileNetV2 [5]. As shown in Figure 5, the operator space O consists of various linear and non-linear MBConv operators with diverse kernel sizes of $K \in \{3, 5, 7\}$ and expansion ratios of $E \in \{3, 6\}$. Among them, each linear MBConv operator contains multiple consecutive linear layers, including three

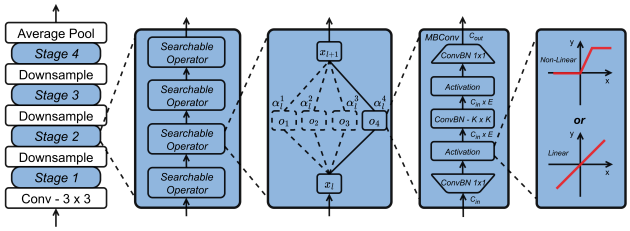

Figure 5: Illustration of the hybrid transformable search space, where both activation layers share the same linearity and non-linearity.

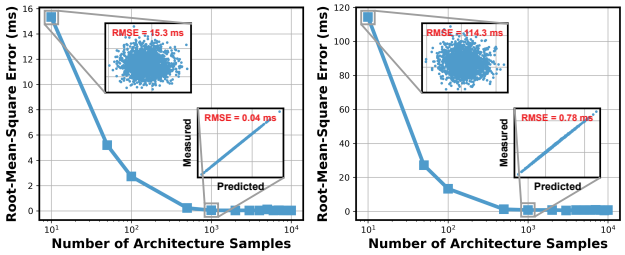

Figure 6: Latency prediction results on Xavier (*left*) and Nano (*right*).

Conv layers, three BN layers, and two linear activation layers, which can be equivalently transformed into one single Conv layer to enable deep-to-shallow transformation, and more importantly, without accuracy loss (see Observation I). Finally, we have $|O| = 2{\times}3{\times}2 = 12$ and given that the supernet has $L = 21$ searchable layers [16], we can interpret that the hybrid transformable search space consists of $|\mathcal{A}| = 12^{21} \approx 4.6 \times 10^{22}$ transformable architecture candidates.

3.2 Vanilla Latency Prediction

As described in Section 3.1, the hybrid transformable search space consists of $|\mathcal{A}| \approx 4.6 \times 10^{22}$ transformable architecture candidates, which necessitates non-trivial resources for exhaustive on-device latency measurements [16, 17]. To avoid this, we introduce a simple yet effective vanilla latency predictor to reliably estimate the on-device latency. Specifically, we first encode possible transformable architecture candidates with the following sparse matrix $\overline{\alpha} \in \mathbb{R}^{L \times N}$:

$$\overline{\alpha}_l^n = \text{one\_hot}(\alpha) = \begin{cases} 1, & \text{if } \alpha_l^n = \arg\max_n ||\alpha_l|| \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $\overline{\alpha}_l^n = 1$ indicates that the operator o_n is reserved in l-th layer while the remaining operators with $\overline{\alpha} = 0$ are eliminated.

Furthermore, we construct the vanilla latency predictor $LAT(\overline{\alpha})$ using a multi-layer perceptron (MLP) with three FC layers [15, 25]. To train the vanilla latency predictor, we first generate 1,000 random transformable architecture candidates $\{\overline{\alpha}_i\}_{i=1}^{1,000}$ from $\mathcal{A}$, after which we transform each of their linear MBConv operators with multiple consecutive linear layers into one single Conv layer to derive their shallow counterparts. The resulting shallow networks are then deployed on Xavier and Nano to collect their latency measurements. We note that the total design cost, including (1) collecting 1,000 architecture-latency pairs and (2) training the vanilla latency predictor, takes less than one hour on both Xavier and Nano.

Results. The latency prediction results on the validation set are illustrated in Figure 6, which clearly shows that the proposed vanilla latency predictor can achieve low root-mean-square errors (RMSE) of 0.04 ms on Xavier and 0.78 ms on Nano, respectively. Besides, we also find that 1,000 architecture-latency pairs are sufficient to yield reliable latency prediction performance on both Xavier and Nano.

3.3 Sandwich-Inspired Transformable Search

As shown in Eq (1), DARTS [8] simultaneously optimizes all the operators in the supernet, which, despite its efficacy, involves excessive memory consumption and thus can only be applied to small proxy task (e.g., CIFAR-10). The resulting optimal network on small proxy task, however, often exhibits sub-optimal accuracy on target task [16]. To enable proxyless search, GDAS [26] further leverages Gumbel-Softmax reparameterization [27] to discretize single-path sub-networks from the supernet, which, thanks to its memory efficiency, has been widely followed in subsequent HW-DNAS works [13–15, 19, 21, 22]. Specifically, GDAS first relaxes the architecture parameters $\alpha \in \mathbb{R}^{L \times N}$ and then re-formulates Eq (1) as follows:

$$x_{l+1} = \sum_{n=1}^{N} \overline{u}_l^n \cdot o_n(x_l), \text{ s.t., } u_l^n = \frac{\exp((p_l^n + g_l^n)/\tau)}{\sum_{n'=1}^{N} \exp((p_l^{n'} + g_l^{n'})/\tau)} \tag{7}$$

where $g \sim \text{Gumbel}(0, 1)$ [27] and $\tau > 0$ is the softmax temperature. Similar to Eq (6), $\overline{u} = \text{one\_hot}(u)$ is the one-hot encoding of u. As such, the involved memory is reduced from *multi-path* to *single-path* since x_{l+1} only depends on the operator o_n with $\overline{u}_l^n = 1$.

☹ **The Rich-Get-Richer Collapse.** The above single-path discretization, despite being able to alleviate the memory bottleneck, suffers from severe search unfairness and thus often ends up with sub-optimal networks [28]. The rationale here is that the operators discretized in the early search process are also more likely to be discretized in the subsequent search process (i.e., *the rich get richer*). As a result, the search engine focuses on discretizing a limited number of operators throughout the search process, leaving the remaining operators less explored as shown in Figure 7 (*top*).

☺ **The Sandwich Rule.** To mitigate the *rich-get-richer* collapse, we introduce an effective sandwich rule to enforce strong search fairness while also incurring similar search cost. Specifically, we turn back to Gumbel-Softmax Top-k [29], which generalizes vanilla Gumbel-Softmax [27] to iteratively discretize k single-path sub-networks without replacement, spanning from the most to the least critical path. In contrast to GDAS [26] and its follow-up NAS works [13–15, 19, 21, 22] that only consider the most critical path (see Eq (7)), the sandwich rule discretizes three unique paths as follows:

$$w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}^{most}(w, \alpha) + \mathcal{L}_{train}^{random}(w, \alpha) + \mathcal{L}_{train}^{least}(w, \alpha) \tag{8}$$

where $\mathcal{L}_{train}^{most}(\cdot)$, $\mathcal{L}_{train}^{least}(\cdot)$, and $\mathcal{L}_{train}^{random}(\cdot)$ are the training loss functions of the most critical path, the least critical path, and the random path that excludes the above two paths, respectively.

Remarks. The sandwich rule involves three forward passes and one backward pass, which marginally increases the search cost from 0.4 to 0.6 GPU-days compared with the single-path discretization (see Eq (7)). Besides, compared with another option that discretizes $|O| = 12$ unique paths to enforce strict search fairness, the sandwich rule maintains considerable search efficiency (i.e., 0.6 vs. 2.1 GPU-days), while achieving similar search fairness (see Figure 7 (*bottom*)).

3.4 Hardware-Aware Transformable Search

Furthermore, to enable hardware-aware transformable search, we integrate the vanilla latency predictor into the sandwich-inspired differentiable search algorithm, which can be formulated as follows:

$$\underset{\alpha}{\text{minimize}} \ \mathcal{L}_{valid}(w^*(\alpha), \alpha) + \lambda \cdot \left| \frac{LAT(\alpha)}{T} - 1 \right| \tag{9}$$

where λ is the trade-off coefficient and T is the specified latency. In contrast to previous well-established HW-DNAS methods [16–18], λ is not a constant but a learnable hyper-parameter, which can be optimized during the search process to guarantee $LAT(\alpha) = T$. For simplicity, we use $\mathcal{L}(w, \alpha, \lambda)$ to represent the optimization objective

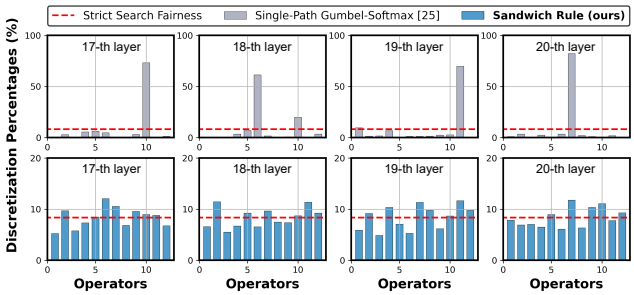

Figure 7: The sandwich rule vs. single-path Gumbel-Softmax [26].

in Eq (9). Finally, we leverage gradient descent to update w and α [8], whereas λ is optimized via gradient ascent [15] as follows:

$$\begin{cases} w^* = w - \eta_w \cdot \frac{\partial \mathcal{L}(w,\alpha,\lambda)}{\partial w}, \ \alpha^* = \alpha - \eta_\alpha \cdot \frac{\partial \mathcal{L}(w,\alpha,\lambda)}{\partial \alpha} \\ \lambda^* = \lambda + \eta_\lambda \cdot \frac{\partial \mathcal{L}(w,\alpha,\lambda)}{\partial \lambda} = \lambda + \eta_\lambda \cdot \left[\frac{LAT(\alpha)}{T} - 1\right] \end{cases} \tag{10}$$

where η_w, η_α, and η_λ are the learning rates of w, α, and λ. Below we further discuss the rationale behind Eq (9)-(10). Note that a smaller λ leads to the architecture with higher latency and vice versa for a larger λ (see Figure 4). In light of this, if $LAT(\alpha) < T$, gradient ascent decreases λ to diminish the trade-off magnitude. As a result, $LAT(\alpha)$ tends to increase towards T in the next search iteration. Likewise, if $LAT(\alpha) > T$, gradient ascent increases λ to enhance the trade-off magnitude. As a result, $LAT(\alpha)$ tends to decrease towards T in the next search iteration. Finally, the search process converges upon $LAT(\alpha) = T$ in one search run, which can satisfy the specified latency constraint while exhibiting optimized accuracy [15].

Differentiable Analysis. As shown in [8, 17, 26] and Eq (10), $\mathcal{L}(w, \alpha, \lambda)$ is differentiable with respect to both w and λ. Furthermore, we also investigate the differentiability of α as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}(w,\alpha,\lambda)}{\partial \alpha} &= \frac{\partial \mathcal{L}_{valid}(w^*(\alpha),\alpha)}{\partial \alpha} + \frac{\lambda}{T} \cdot \frac{\partial LAT(\alpha)}{\alpha} \\ &= \frac{\mathcal{L}_{valid}}{\partial \overline{u}} \cdot \frac{\partial \overline{u}}{\partial u} \cdot \frac{\partial u}{\partial p} \cdot \frac{\partial p}{\partial \alpha} + \frac{\lambda}{T} \cdot \frac{\partial LAT(\alpha)}{\partial \overline{u}} \cdot \frac{\partial \overline{u}}{\partial u} \cdot \frac{\partial u}{\partial p} \cdot \frac{\partial p}{\partial \alpha} \end{aligned} \tag{11}$$

where $\frac{\partial \overline{u}}{\partial u} \approx 1$ as discussed in [15]. Besides, $\frac{\partial LAT(\alpha)}{\partial \overline{u}}$ is determined by the pre-trained vanilla latency predictor $LAT(\cdot)$, which can be easily interpreted through a one-time backward propagation [15]. Apart from these, other terms in Eq (11) are apparently differentiable since only continuous transformations are involved [8, 17, 26].

3.5 Hybrid Transformable Training

We next elaborate on the tailored hybrid transformable training. In practice, we do not directly train the searched transformable network with both linear and non-linear MBConv operators. As shown in Figure 8, we (1) start from the same network where all the MBConv operators are non-linear and (2) then gradually eliminate its non-linearity to convert it back to the original network for subsequent deep-to-shallow transformation. Without loss of generality, we use $Act(\cdot)$ to represent the non-linear activation (e.g., ReLU6). For the searched transformable network, we first graft the linear activation in its linear MBConv operators with the following hybrid activation:

$$Act^*(x) = Act(x) + \epsilon \cdot [x - Act(x)] \tag{12}$$

where x is the input and $\epsilon \in [0, 1]$ is the hyper-parameter to balance the linearity and non-linearity. As shown in Eq (12), $Act^*(\cdot)$ is linear when $\epsilon = 1$ and non-linear when $\epsilon \neq 1$. In view of this, we set $\epsilon = E_{curr}/E_{total}$, where E_{curr} is the current training epoch and E_{total} is the total number of training epochs to eliminate the grafted non-linearity. Note that $\epsilon = 1$ when $E_{curr} \geq E_{total}$. Therefore, at the beginning of training (i.e., $E_{curr} = 0$), we have $Act^*(x) = Act(x)$,

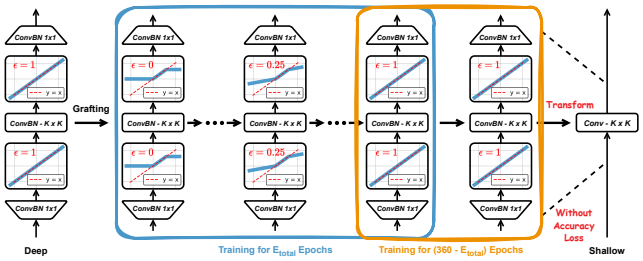

Figure 8: An intuitive overview of hybrid transformable training.

which exhibits strong non-linearity. Then, as the training proceeds, the grafted non-linearity is gradually eliminated. Finally, when $E_{curr} \geq E_{total}$, we have $Act^*(x) = x$, which converts the grafted non-linear activation back to its original linear counterpart. As such, the network structure remains the same at the end of training (see Figure 8) and thus maintains the same hardware efficiency.

3.6 Linear Transformation

Finally, for the well-trained deep-to-shallow transformable network, we further transform each of its linear MBConv operators with multiple consecutive linear layers into one single Conv layer to derive its shallow counterpart, which can boost the on-device efficiency without accuracy loss. Note that transforming one Conv layer and its subsequent BN or linear activation layer is equivalent to linearly manipulating the Conv layer itself. In view of this, below we mainly discuss how to transform multiple consecutive Conv layers.

Without loss of generality, we consider two consecutive Conv layers, which have an input feature $X \in \mathbb{R}^{C_1 \times H_1 \times W_1}$, an intermediate feature $Y \in \mathbb{R}^{C_2 \times H_2 \times W_2}$, and an output feature $Z \in \mathbb{R}^{C_3 \times H_3 \times W_3}$. In addition, $W^1 \in \mathbb{R}^{C_1 \times C_2 \times K_1 \times K_1}$ and $W^2 \in \mathbb{R}^{C_2 \times C_3 \times K_2 \times K_2}$ are the weights of the above two Conv layers. Following [25], we can transform the above two consecutive Conv layers as follows:

$$Z_{c,h,w} = \sum_{c_i=0}^{C_1-1} \sum_{k_h^*=0}^{K^*-1} \sum_{k_w^*=0}^{K^*-1} W^*_{c_i,c,k_h^*,k_w^*} X_{c_i,h+\Delta k_h^*,w+\Delta k_w^*} \tag{13}$$

where $\Delta k_h^* = k_h^* - [\frac{K^*-1}{2}]$ and $\Delta k_w^* = k_w^* - [\frac{K^*-1}{2}]$. Besides, $W^* \in \mathbb{R}^{C_1 \times C_3 \times K^* \times K^*}$ is the weight of the transformed Conv layer with the kernel size of $K^* = K_1 + K_2 - 1$ [25], which can be calculated as:

$$W^*_{c_i,c_o,h,w} = \sum_{c_j=0}^{C_2-1} \sum_{k_h=0}^{K_2-1} \sum_{k_w=0}^{K_2-1} W^1_{c_i,c_j,k_h,k_w} W^2_{c_j,c_o,h-\Delta k_h,w-\Delta k_w} \tag{14}$$

where $\Delta k_h = k_h - [\frac{K_2-1}{2}]$ and $\Delta k_w = k_w - [\frac{K_2-1}{2}]$. As shown in Eq (13)-(14), the above two consecutive Conv layers with W^1 and W^2 can be equivalently transformed into one single Conv layer with W^*, both of which maintain the same output Z.

Results. As shown in Figure 9, linear transformation can achieve up to ×1.9 speedup on Xavier and up to ×1.7 speedup on Nano without accuracy loss on ImageNet. We note that the on-device latency may fluctuate when linear transformation is not included. The rationale here is that different transformable networks favor different linear and non-linear MBConv operators in order to maximize the attainable accuracy under the given computational budgets.

3.7 Relationships with Previous Methods

First and foremost, previous SOTA NAS methods [9–22] typically feature sufficient network depth in order to achieve strong accuracy [23]. However, the resulting deep networks exhibit poor hardware efficiency [24]. To remedy this issue, we may re-engineer previous SOTA NAS methods to explore shallow networks, which, unfortunately, suffer from low accuracy since the attainable accuracy relies

Table 1: Comparisons with previous well-established NAS methods.

	[8]	[26]	[9]	[12]	[17]	[16]	[15]	ours
Differentiable Search	✓	✓	✗	✗	✓	✓	✓	✓
Proxyless Search	✗	✗/✓	✓	✓	✓	✓	✓	✓
Latency Optimization	✗	✗	✓	✓	✓	✓	✓	✓
Specified Latency	✗	✗	✓	✓	✗	✗	✓	✓
Transformable Search	✗	✗	✗	✗	✗	✗	✗	✓
Transformable Training	✗	✗	✗	✗	✗	✗	✗	✓
Search Cost (GPU-days)	1	0.2/0.4	1,667	53	9	8	0.4	0.6

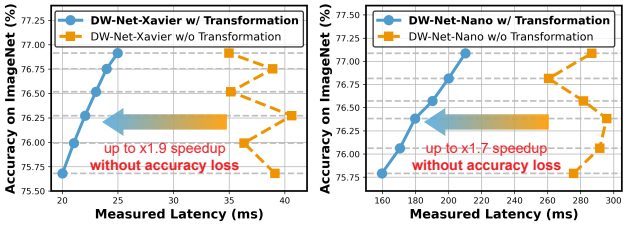

Figure 9: Linear transformation on Xavier (*left*) and Nano (*right*).

on sufficient network depth [4]. This dilemma reveals that previous SOTA NAS methods can only win either accuracy or efficiency and thus cannot enable an aggressive win-win in terms of both.

In contrast, DW-NAS draws insights from deep-to-shallow transformable networks and establishes the first deep-to-shallow transformable HW-DNAS paradigm to marry the best of both deep and shallow networks. More importantly, DW-NAS provides a holistic solution for designing and training deep-to-shallow transformable networks towards an aggressive win-win in terms of both accuracy and hardware efficiency, which distinguishes itself from previous well-established NAS practices as shown in Table 1.

4 EXPERIMENTS

In this section, we conduct extensive experiments, including thorough comparisons with previous SOTA efficient networks and insightful ablation studies, to show the superiority of DW-NAS.

4.1 Experimental Setup

Datasets. We consider two large-scale datasets, including ImageNet and ImageNet-100 [30]. Specifically, ImageNet consists of 1,281,167 training images and 50,000 validation images, which are distributed across 1,000 classes. In addition, ImageNet-100 is a subset of ImageNet and consists of 100 random classes from ImageNet.

Embedded Platforms. We consider two popular intelligent embedded systems, including NVIDIA Jetson AGX Xavier and NVIDIA Jetson Nano. Specifically, Xavier features 512-core Volta GPU and 8-core Carmel ARM CPU, whereas Nano comes with 128-core Maxwell GPU and quad-core ARM CPU. For fair comparisons, we report all the latency measurements under the same input batch size of 8.

Architecture Search Settings. The search experiments closely follow FBNet [17], where both network weights w and architecture parameters α are optimized on ImageNet-100. Specifically, the supernet is trained from scratch for 90 epochs with an input batch size of 192, where α is frozen in the first 10 epochs [17]. To optimize w, we employ SGD with a learning rate of 0.1, a momentum of 0.9, and a weight decay of 3×10^{-5}. In parallel, to optimize α, we utilize Adam with a learning rate of 0.001 and a weight decay of 1×10^{-3}. Furthermore, we initialize the trade-off coefficient λ (see Eq (9)) to zero and optimize λ using gradient ascent, where the learning rate is set to 0.0005. Note that all the search experiments are performed on one single GeForce RTX 3090 GPU, each of which takes 0.6 days. Finally, we denote the searched transformable networks on Xavier and Nano as DW-Net-Xavier and DW-Net-Nano, respectively.

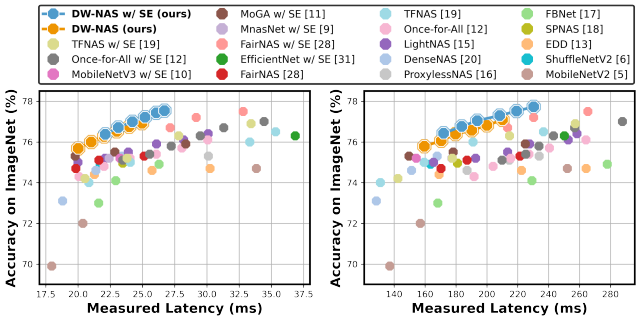

Figure 10: DW-NAS vs. SOTA efficient networks on Xavier (*left*) and Nano (*right*), where SE is the Squeeze-and-Excitation module [9, 10].

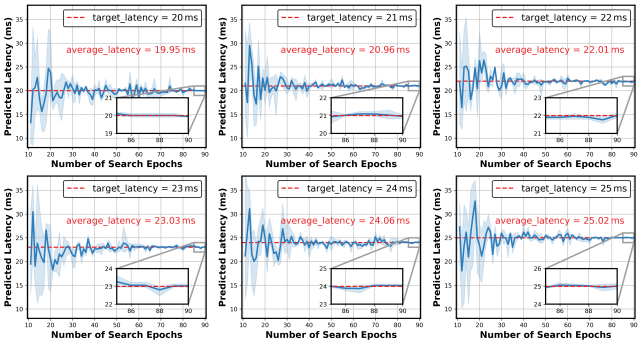

Figure 11: Visualization of the search process of DW-Net-Xavier.

Architecture Evaluation Settings. We also closely follow FBNet [17] to evaluate the searched DW-Nets on ImageNet and ImageNet-100. Specifically, all the searched DW-Nets are trained from scratch for 360 epochs with an input batch size of 256 on 4 GeForce RTX 3090 GPUs. The default optimizer is SGD with an initial learning rate of 0.1 (annealed to zero following the cosine schedule), a momentum of 0.9, and a weight decay of 4×10^{-5}. Finally, when hybrid transformable training is enabled, the grafted non-linearity is gradually eliminated in the first 120 epochs. Note that the total training epochs remain 360 regardless of hybrid transformable training.

4.2 Results and Ablation Studies

Architecture Evaluation Results. To demonstrate the efficacy of DW-NAS, we first compare the searched DW-Nets with a plethora of previous SOTA efficient networks on ImageNet, spanning from *manual* [5, 6] to *automated* [9–13, 15–20, 28, 31]. The experimental results on ImageNet are illustrated in Figure 10, which clearly shows that DW-NAS can deliver significantly better accuracy-efficiency trade-offs than the aforementioned SOTA efficient networks on both Xavier and Nano. For example, compared with FBNet-C [17], DW-Net-Xavier-20ms can achieve +0.8% higher accuracy on ImageNet, while at the same time maintaining ×1.3 speedup on Xavier.

Architecture Search Stability. To show the search stability, we further conduct a series of search experiments under various latency constraints on Xavier. For each latency constraint, we repeat three search experiments under different random initialization seeds. As illustrated in Figure 11, DW-NAS exhibits strong search stability, which can consistently navigate the required transformable network that satisfies the specified latency constraint.

Train-First vs. Transform-First. As shown in Figure 8, we first train the searched transformable network, which is then equivalently transformed into its shallow counterpart. In parallel, another natural alternative is to first transform the searched transformable

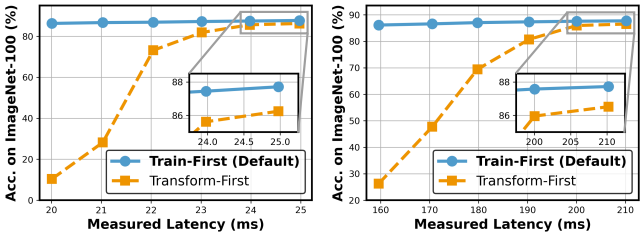

Figure 12: Train-First vs. Transform-First on Xavier (*left*)/Nano (*right*).

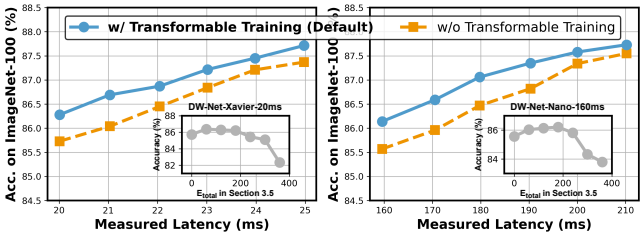

Figure 13: Ablation of hybrid transformable training on Xavier (*left*) / Nano (*right*), where zoom-in figures compare different E_{total} settings.

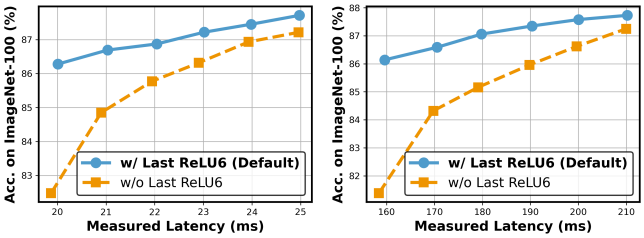

Figure 14: Ablation of last ReLU6 on Xavier (*left*) and Nano (*right*).

network into its shallow counterpart to eliminate its linear network redundancy and then train the resulting shallow network instead. Both options end up with the same shallow network structure. The latter, however, suffers from severe accuracy loss on ImageNet-100 (see Figure 12), which reveals that the linear network redundancy can enhance the training process towards better accuracy.

Ablation of Hybrid Transformable Training. Furthermore, we investigate the efficacy of hybrid transformable training. As shown in Figure 13, hybrid transformable training can deliver consistent accuracy gains on ImageNet-100 without degrading the on-device efficiency on both Xavier and Nano. Besides, as shown in zoom-in figures, $E_{total} \in [60, 180]$ allows us to safely eliminate the grafted network non-linearity and convert the grafted transformable network back to its original transformable network for subsequent transformation (see Figure 8). As such, we empirically set $E_{total} = 120$.

Ablation of Last ReLU6. As shown in Figure 3, non-linear ReLU6 activation is computationally cheap. Therefore, we also include an additional ReLU6 at the end of each linear `MBConv` operator, in which multiple consecutive linear layers can still be equivalently transformed into one single `Conv` layer. As shown in Figure 14, this can largely boost the attainable accuracy on ImageNet-100 with only negligible computational cost on both Xavier and Nano.

Ablation of Sandwich Rule We also compare the proposed sandwich rule with three other search counterparts [26, 27, 29]. Among them, vanilla Gumbel-Softmax [27] simultaneously optimizes all the operators in the supernet and inevitably suffers from out-of-memory (OOM) crashes due to the excessive memory consumption. As shown in Table 2, the sandwich rule can achieve reliable search performance, while also maintaining strong search efficiency.

Table 2: Ablation of the proposed sandwich rule on ImageNet-100.

Differentiable Search Algorithm	Search Cost (GPU-Days) ↓	Accuracy (%) ↑	Latency (ms) ↓
Vanilla Gumbel-Softmax [27]	OOM	OOM	OOM
Single-Path Gumbel-Softmax [26]	0.4	20.0	85.76
Gumbel-Softmax Top-k w/o Sandwich [29]	2.1	20.0	86.42
Gumbel-Softmax Top-k w/ Sandwich (ours)	0.6	20.0	86.28

5 CONCLUSION

In this paper, we introduce the first-of-its-kind deep-to-shallow transformable NAS paradigm, namely Double-Win NAS (DW-NAS), which provides a holistic solution for designing and training deep-to-shallow transformable networks, marrying the best of both deep and shallow networks towards an aggressive win-win in terms of both accuracy and hardware efficiency. Extensive experiments on two NVIDIA Jetson intelligent embedded systems clearly demonstrate the merits of DW-NAS over previous SOTA NAS approaches.

REFERENCES

[1] Wu et al. Machine Learning at Facebook: Understanding Inference at the Edge. In *HPCA*, 2019.
[2] Neseem et al. AdaCon: Adaptive Context-Aware Object Detection for Resource-Constrained Embedded Devices. In *ICCAD*, 2021.
[3] Li et al. RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering. In *ICCAD*, 2022.
[4] He et al. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.
[5] Sandler et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018.
[6] Ma et al. ShuffleNetV2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*, 2018.
[7] Cai et al. Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications. *TODAES*, 2022.
[8] Liu et al. DARTS: Differentiable Architecture Search. *arXiv*, 2019.
[9] Tan et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*, 2019.
[10] Howard et al. Searching for MobileNetV3. In *ICCV*, 2019.
[11] Chu et al. MoGA: Searching Beyond MobileNetV3. In *ICASSP*, 2020.
[12] Cai et al. Once-for-All: Train One Network and Specialize It for Efficient Deployment. *arXiv*, 2020.
[13] Li et al. EDD: Efficient Differentiable DNN Architecture and Implementation Co-search for Embedded AI Solutions. In *DAC*, 2020.
[14] Zhang et al. DIAN: Differentiable Accelerator-Network Co-Search Towards Maximal DNN Efficiency. In *ISLPED*, 2021.
[15] Luo et al. LightNAS: On Lightweight and Scalable Neural Architecture Search for Embedded Platforms. *TCAD*, 2022.
[16] Cai et al. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *arXiv*, 2019.
[17] Wu et al. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*, 2019.
[18] Stamoulis et al. Single-Path NAS: Designing Hardware-Efficient ConvNets in Less Than 4 Hours. In *ECML-PKDD*, 2019.
[19] Hu et al. TF-NAS: Rethinking Three Search Freedoms of Latency-Constrained Differentiable Neural Architecture Search. In *ECCV*, 2020.
[20] Fang et al. Densely Connected Search Space for More Flexible Neural Architecture Search. In *CVPR*, 2020.
[21] Luo et al. You Only Search Once: On Lightweight Differentiable Architecture Search for Resource-Constrained Embedded Platforms. In *DAC*, 2022.
[22] Li et al. Physics-Aware Differentiable Discrete Codesign for Diffractive Optical Neural Networks. In *ICCAD*, 2022.
[23] Benmeziane et al. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. *arXiv*, 2021.
[24] He et al. Convolutional Neural Network at Constrained Time Cost. In *CVPR*, 2015.
[25] Luo et al. Pearls Hide Behind Linearity: Simplifying Deep Convolutional Networks for Embedded Hardware Systems via Linearity Grafting. *ASP-DAC*, 2024.
[26] Dong et al. Searching for A Robust Neural Architecture in Four GPU Hours. In *CVPR*, 2019.
[27] Jang et al. Categorical Reparameterization with Gumbel-Softmax. *arXiv*, 2017.
[28] Chu et al. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. In *ICCV*, 2021.
[29] Kool et al. Stochastic Beams and Where to Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. In *ICML*, 2019.
[30] Deng et al. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.
[31] Tan et al. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, 2019.