Pareto-aware Neural Architecture Generation for Diverse Computational Budgets

Yong Guo, Yaofo Chen, Yin Zheng, Qi Chen, Peilin Zhao, Junzhou Huang, Jian Chen, Mingkui Tan*

South China University of Technology, Tencent AILab, University of Adelaide

{guo.yong, sechenyaofo}@mail.scut.edu.cn, yzheng3xg@gmail.com, qi.chen04@adelaide.edu.au,

{masonzhao, joehhuang}@tencent.com, {ellachen, mingkuitan}@scut.edu.cn

Abstract

Designing feasible and effective architectures under diverse computational budgets, incurred by different applications/devices, is essential for deploying deep models in real-world applications. To achieve this goal, existing methods often perform an independent architecture search process for each target budget, which is very inefficient yet unnecessary. More critically, these independent search processes cannot share their learned knowledge (i.e., the distribution of good architectures) with each other and thus often result in limited search results. To address these issues, we propose a Pareto-aware Neural Architecture Generator (PNAG) which only needs to be trained once and dynamically produces the Pareto optimal architecture for any given budget via inference. To train our PNAG, we learn the whole Pareto frontier by jointly finding multiple Pareto optimal architectures under diverse budgets. Such a joint search algorithm not only greatly reduces the overall search cost but also improves the search results. Extensive experiments on three hardware platforms (i.e., mobile device, CPU, and GPU) show the superiority of our method over existing methods.

1. Introduction

Deep neural networks (DNNs) [30] have been the workhorse of many challenging tasks, including image classification [18, 24, 37, 51], semantic segmentation [6, 50, 60, 63] and object detection [5, 47, 58, 71]. However, designing effective architectures often relies heavily on human expertise. To alleviate this issue, neural architecture search (NAS) methods have been proposed to automatically design effective architectures [73]. Existing studies show that these automatically searched architectures often outperform the manually designed ones in many computer vision tasks [8, 15, 31, 55, 61, 67, 74].

*Corresponding author.

However, the state-of-the-art deep networks often contain a large number of parameters and come with extremely high computational cost. As a result, it is hard to deploy these models to real-world scenarios with limited computation resources. Regarding this issue, we have to carefully design architectures to fulfill a specific computational budget (*e.g.*, a feasible model should have a latency lower than 100ms on a specified mobile device). More critically, we may have to consider different computational budgets in the real world. For example, a company may simultaneously develop/maintain multiple applications and each of them has a specific budget of latency.

In order to design feasible architectures, most methods [52, 53] only considers a single computational budget and incorporates architecture's computational cost into the objective function. When we consider diverse budgets, they have to conduct an independent search process for each budget [53], which is very inefficient yet unnecessary. Unlike these methods, one can also exploit the population-based methods to simultaneously find multiple architectures and then select an appropriate one from them to fulfill a specific budget [38, 40]. However, due to the limited population size, these searched architectures do not necessarily satisfy the required budget. More critically, all these searched architectures are fixed after search and cannot be easily adapted for a slightly changed budget. Thus, how to design effective architectures under diverse computational budgets in an efficient and flexible way still remains an open question.

In this paper, we propose a Pareto-aware Neural Architecture Generator (PNAG) which only needs to be trained once and then dynamically produces Pareto optimal architectures for diverse budgets via *inference* (as shown in Fig. 1a). Note that the Pareto optimal architectures under different budgets should lie on a distribution, *i.e.*, the Pareto frontier over model performance and computational cost [28]. We propose to jointly learn the whole Pareto frontier (*i.e.*, improving the blue curve to the red curve in Fig. 1b) instead of finding a single Pareto optimal architecture. During training, we randomly sample budgets from a
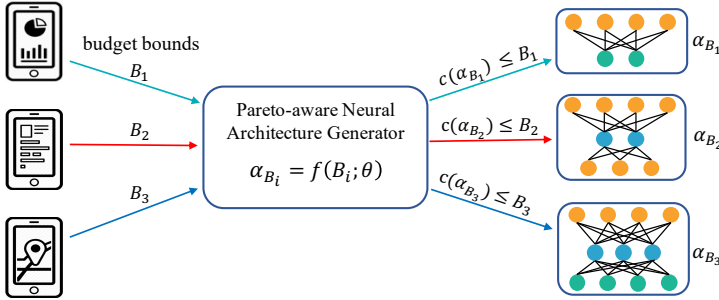
(a) Illustration of generating feasible architectures for diverse budgets using PNAG.
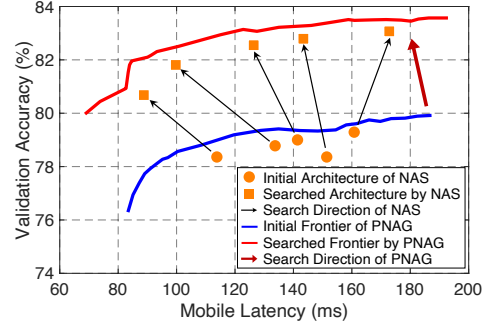
(b) Comparisons between PNAG and conventional NAS.

Figure 1. We show an illustration of how to apply our PNAG to generate feasible architectures for diverse computational budgets and the comparisons between our PNAG and conventional NAS methods. (a) PNAG takes an arbitrary computational budget as input and flexibly generates architectures. (b) Our PNAG learns the whole Pareto frontier rather than finding discrete architectures. Here, the accuracy is measured on the constructed validation set.

predefined distribution and maximize the expected reward of the searched architectures to approximate the ground-truth Pareto frontier. It is worth noting that learning the Pareto frontier is able to share the learned knowledge across different budgets and greatly improve the search results in practice (see results in Table 3). Furthermore, when evaluating architectures under diverse budgets, we design an architecture evaluator that learns a Pareto dominance rule to determine which architecture is a relatively better one in pairwise comparisons. Unlike the existing methods, we highlight that the proposed PNAG designs architectures through a generation process instead of search, which is very efficient (see results in Table 4) and practically useful in real-world model design and deployment.

We summarize the contributions of our paper as follows.

- Instead of designing architectures for a single budget, we propose a Pareto-aware Neural Architecture Generator (PNAG) which is only trained once and flexibly generates effective architectures for arbitrary budget via inference (see Fig. 1a). In this way, our architecture generation process becomes very efficient and practically useful in real-world applications.

- To train PNAG, we explicitly learn the Pareto frontier by maximizing the expected reward of the searched architectures over diverse budgets. Interestingly, learning the Pareto frontier shares the learned knowledge across the search processes under diverse budgets and greatly improves the search results (see Table 3).

- Since an architecture should have different rewards/scores under different budgets, we propose an architecture evaluator to adaptively evaluate architectures for any given budget. To train the evaluator, we propose to learn a Pareto dominance rule which determines whether an architecture is better than the other in pairwise comparisons.

- We measure the latencies on three hardware platforms and take them as the computational budgets to generate feasible architectures. Extensive experiments show that the architectures produced by PNAG consistently outperform the architectures searched by existing methods across different budgets and platforms.

2. Related Work

In this section, we provide a brief overview of existing work on neural architecture search, architecture design under resource constraints, as well as Pareto frontier learning.

2.1. Neural Architecture Search (NAS)

Unlike manually designing architectures with expert knowledge, NAS seeks to automatically design more effective architectures [23, 33, 68, 69, 72]. Existing NAS methods can be roughly divided into three categories, namely, reinforcement-learning-based methods, evolutionary approaches, and gradient-based methods. Specifically, reinforcement-learning-based methods [42, 44, 57, 59, 73] learn a controller to produce architectures. Evolutionary approaches [7, 34, 36, 39, 45, 46] search for promising architectures by gradually evolving a population. Gradient-based methods [10,11,13,35,66] relax the search space to be continuous and optimize architectures by gradient descent. Besides designing effective search algorithms, many efforts have also been made to improve the accuracy of architecture evaluation [14, 65, 70]. Unlike these methods that find a single architecture, one can design different architectures by training an architecture generator. Specifically, Rand-Wire [64] designs stochastic network generators to generate randomly wired architectures. NAGO [48] is the first work to learn an architecture generator and proposes a hierarchical and graph-based search space to reduce the optimization difficulty. However, these generated architectures tend to perform very similarly (*i.e.*, low diversity) in terms of
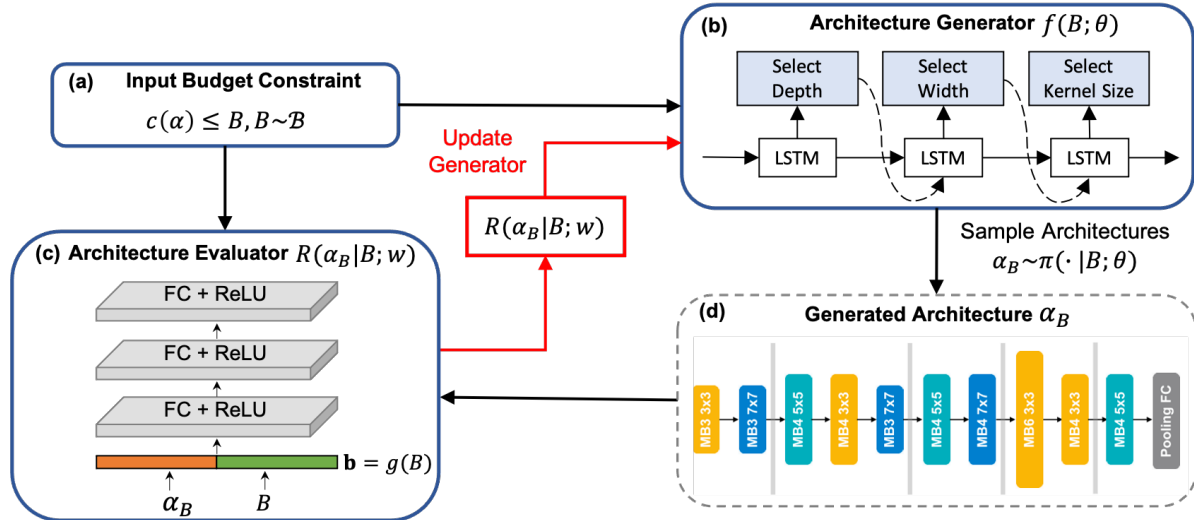
Figure 2. Overview of the proposed PNAG. Our PNAG mainly consists of two modules: an architecture generator $f(\cdot; \theta)$ and an architecture evaluator $R(\cdot|\cdot; w)$. Specifically, we build the generator model based on an LSTM network, which takes a budget constraint B as input and produces a promising architecture α_B that satisfies the budget constraint, i.e., $c(\alpha)$. To optimize the generator model, we design the evaluator using three fully connected (FC) layers to estimate the performance of the generated architectures α_B. The orange and green boxes in (c) denote the embeddings of architecture α_B and the budget w.r.t. B, respectively.

both model performance and computational cost [48, 64]. Thus, these architectures may not satisfy an arbitrary required budget. In other words, they still have to learn a generator for a required budget to produce architectures.

2.2. Architecture Design under Constraints

Many efforts have been made in designing architectures under a resource constraint [1,3,22,26,32,56]. Specifically, PONAS [26] builds an accuracy table to find architectures satisfying a single budget constraint. TuNAS [1] proposes a reward function to restrict the latency of the searched architecture, which omits additional hyper-parameter tuning. Related to our work, SGNAS [27] proposes an architecture generator which generates architectures for specific budget constraints. Nevertheless, SGNAS optimizes a regression loss *w.r.t.* budget constraint and the resultant architecture does not necessarily have lower cost than the target budget, *i.e.*, violating the budget. More critically, SGNAS considers a fixed hyper-parameter λ to balance the regression loss and a classification loss. Due to the large diversity among architectures, their accuracy and computational cost may vary significantly across different budgets, also leading to sub-optimal search results (See Table 1).

2.3. Pareto Frontier Learning

Given multiple objectives, Pareto frontier learning aims to find a set of Pareto optimal solutions over them. Most methods exploit evolutionary algorithms [16, 29] to solve this problem. Inspired by them, many efforts have been made to simultaneously find a set of Pareto optimal ar-

chitectures over accuracy and computational cost [12, 17]. Recently, NSGANetV1 [41] presents an evolutionary approach to find a set of trade-off architectures over multiple objectives in a single run. NSGANetV2 [38] further presents two surrogates (at the architecture and weights level) to produce task-specific models under multiple competing objectives. Given a target budget, these methods may manually select an appropriate architecture from a set of searched architectures. However, given limited population size, the selected architectures do not necessarily satisfy a required budget. More critically, all the searched architectures are fixed after search and cannot be easily adapted for a slightly changed budget. Thus, how to learn the Pareto frontier and use it to generate architectures for arbitrary budget in a flexible way still remains unexplored.

3. Pareto-aware Architecture Generation

In this paper, we focus on the architecture generation problem and intend to generate effective architectures for diverse computational budgets via *inference* instead of search/training. Note that the optimal architectures under different budgets lie on the Pareto frontier over model performance and computational cost [28]. Thus, we develop a Pareto-aware Neural Architecture Generator (PNAG) to explicitly learn the whole Pareto frontier. To locate the best architecture from the frontier for a given budget, we build our PNAG as a conditional model which takes the budget as input and directly produces a feasible architecture. In Section 3.1, we depict our architecture generator model and

present a novel learning algorithm to learn the Pareto frontier. In Section 3.2, we propose an architecture evaluator, as well as its training algorithm, to adaptively evaluate architectures under different budgets. Algorithm 1 shows the whole training process of PNAG.

3.1. Learning the Architecture Generator $f(B; \theta)$

We seek to build an architecture generator model to dynamically and flexibly produce effective architectures for any given computational budget. Let B be a budget (*e.g.*, latency or MAdds) which can be considered as a random variable drawn from some distribution $\mathcal{B}$, namely $B \sim \mathcal{B}$. Let Ω be an architecture search space. For any architecture $\alpha \in \Omega$, we use $c(\alpha)$ and $\text{Acc}(\alpha)$ to measure the cost and validation accuracy of α, respectively.

Since an architecture can be represented as a sequence of tokens (each token denotes a setting of a layer, *e.g.*, width or kernel size) [44, 73], we cast the architecture generation problem as a sequential decision problem and build the architecture generator $f(B; \theta)$ using an LSTM network. As shown in Fig. 2, the generator takes a budget B as input and generates architectures $\alpha_B = f(B; \theta)$ (satisfying the constraint $c(\alpha_B) \leq B$) by sequentially predicting the token sequences, *i.e.*, the depth, width, and kernel size of each layer. Here, θ denotes the learnable parameters. Note that the optimal architecture under a specific budget should lie on the Pareto frontier over model performance and computational cost. To make the generator generalize to arbitrary budget, we seek to learn the Pareto frontier rather than finding discrete architectures. In the following, we first illustrate our training method in Section 3.1.1 and then discuss how to represent a arbitrary budget in Section 3.1.2.

3.1.1 Training Method of $f(B; \theta)$

To illustrate the training objective of our method, we first revisit the NAS problem with a single budget and then generalize it to the problem with diverse budgets.

NAS under a single budget. Since it is non-trivial to directly find the optimal architecture [73], by contrast, one can first learn a policy $\pi(\cdot; \theta)$ and then conduct sampling from it to find promising architectures, *i.e.*, $\alpha \sim \pi(\cdot; \theta)$. Given a budget B, the optimization problem becomes

$$\max_{\theta} \ \mathbb{E}_{\alpha \sim \pi(\cdot; \theta)} \left[R\left(\alpha | B; w\right) \right], \ \text{s.t.} \ c(\alpha) \leq B. \quad (1)$$

Here, $\pi(\cdot; \theta)$ is the learned policy parameterized by θ, and $R(\alpha | B; w)$ is the reward function parameterized by w that measures the joint performance of both the accuracy and latency of the architecture α. $\mathbb{E}_{\alpha \sim \pi(\cdot; \theta)} [\cdot]$ is the expectation over the searched architectures.

NAS under diverse budgets. Problem (1) only focuses on one specific budget constraint. In fact, we seek to learn

---

Algorithm 1 Training method of PNAG.

Require: Search space Ω, latency distribution $\mathcal{B}$, learning rate η, training data set $\mathcal{D}$, parameters M, N, and K.
1: Initialize model parameters θ for the generator and w for the architecture evaluator.
  // Collect the architectures with accuracy and latency
2: Train a supernet S on $\mathcal{D}$.
3: Randomly sample architectures $\{\beta_i\}_{i=1}^{M}$ from Ω.
4: Construct tuples $\{(\beta_i, c(\beta_i), \text{Acc}(\beta_i))\}_{i=1}^{M}$ using S.
  // Learn the architecture evaluator
5: **while** not convergent **do**
6:   Sample a set of latencies $\{B_k\}_{k=1}^{K}$ from $\mathcal{B}$.
7:   Update the architecture evaluator by:
8:     $w \leftarrow w - \eta \nabla_w L(w).$
9: **end while**
  // Learn the architecture generator
10: **while** not convergent **do**
11:   Sample a set of latencies $\{B_k\}_{k=1}^{K}$ from $\mathcal{B}$.
12:   Obtain $\{\alpha_{B_k}^{(i)}\}_{i=1}^{N}$ from $\pi(\cdot | B_k; \theta)$ for each B_k.
13:   Update the generator via policy gradient by:
14:     $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta).$
15: **end while**

---

the Pareto frontier over the whole range of budgets (*e.g.*, latency). However, this problem is hard to solve since there may exist infinite Pareto optimal architectures with different computational cost. To address this, one can learn an approximated Pareto frontier by finding a set of uniformly distributed Pareto optimal points [20]. Here, we evenly sample K budgets from the range of latency and maximize the expected reward over them. Thus, the problem becomes

$$\max_{\theta} \ \mathbb{E}_{B \sim \mathcal{B}} \left[\mathbb{E}_{\alpha_B \sim \pi(\cdot | B; \theta)} \left[R\left(\alpha_B | B; w\right) \right] \right],$$
$$\text{s.t.} \ c(\alpha_B) \leq B, \ B \sim \mathcal{B}, \quad (2)$$

where $\mathbb{E}_{B \sim \mathcal{B}} [\cdot]$ denotes the expectation over the distribution of budget. Unlike Eqn. (1), $\pi(\cdot | B; \theta)$ is the learned policy conditioned on the budget of B. In practice, we use policy gradient to learn the architecture generator. To encourage exploration, we follow [21, 44] to introduce an entropy regularization. Please refer to the supplementary materials for more details.

Advantages over existing NAS methods. Our PNAG exhibits two advantages over existing NAS methods. *First*, our PNAG is able to share the learned knowledge across the search processes under different budgets, which greatly improves the search results (see Table 3). The main reason is that, once we find a good architecture for one budget, we may easily obtain a competitive architecture for a larger/smaller budget by slightly modifying some components (model width or kernel size). *Second*, given a well-trained PNAG, we can directly use it to generate feasible architectures for any required budget via inference, which is very efficient and practically useful (see Table 4).

3.1.2 Vector Representation of Budget Bounds

To learn the architecture generator, we still have to consider how to represent the budget bound B as the inputs of PNAG. As mentioned before, our PNAG considers K discrete budgets during training. To represent different budgets, we use an embedding vector [44] to represent different budgets (See details in Section 3.1.2). Following [44], we build a learnable embedding vector $\mathbf{b} = g(B)$ for each sampled budget B. We incorporate these learnable embedding vectors into the parameters of the architecture generator and train them jointly. In this way, we are able to automatically learn the vectors of these budgets and encourage PNAG to produce feasible architectures.

As mentioned before, we only sample a set of discrete budgets to train PNAG. To accommodate all the budgets belonging to a continuous space, we propose an embedding interpolation method to represent a budget with any possible value. Specifically, we perform a linear interpolation between the embedding of two adjacent discrete budgets to represent the considered budgets. For a target budget B between two sampled budgets $B_1 < B < B_2$, the linear interpolation of the budget vector $\mathbf{b}$ can be computed by

$$\mathbf{b} = g(B) = \xi g(B_1) + (1-\xi)g(B_2), \text{ where } \xi = \frac{B_2 - B}{B_2 - B_1},$$

Here, $\xi \in [0, 1]$ denotes the weight of B_1 in interpolation.

3.2. Learning the Architecture Evaluator $R(\cdot|B; w)$

Given diverse budgets, an architecture should have different rewards/scores regarding whether it satisfies the corresponding budget constraint. However, it is non-trivial to manually design a reward function for each budget. Instead, we propose to learn an architecture evaluator to automatically predict the score. To this end, we build an evaluator with three fully connected layers. Given any architecture β and a budget B, we seek to predict the performance $R(\beta|B; w)$ of β under the budget B. Since we have no ground-truth labels for training, following [2, 9, 19], we learn the evaluator via pairwise architecture comparisons.

3.2.1 Training Method of $R(\cdot|B; w)$

To obtain a promising evaluator, we train the architecture evaluator using a pairwise ranking loss, which has been widely used in ranking problems [2, 9, 19]. Specifically, we collect M architectures with accuracy and latency, and record them as a set of triplets $\{(\beta_i, c(\beta_i), \mathrm{Acc}(\beta_i))\}_{i=1}^{M}$. Thus, given M architectures, we have $M(M-1)$ architecture pairs $\{(\beta_i, \beta_j)\}$ in total after omitting the pairs with themselves. Assuming that we have K budgets, the pair-

wise ranking loss becomes

$$L(w) = \frac{1}{KM(M-1)} \sum_{k=1}^{K} \sum_{i=1}^{M} \sum_{j=1, j \neq i}^{M} \phi\Big(d(\beta_i, \beta_j, B_k)$$
$$\cdot \big[R(\beta_i|B_k; w) - R(\beta_j|B_k; w)\big]\Big), \tag{3}$$

where $d(\beta_1, \beta_2, B_k)$ denotes a function to indicate whether β_i is better than β_j under the budget B_k, as will be discussed in Section 3.2.2. $\phi(z) = \max(0, 1 - z)$ is a hinge loss function and we use it to enforce the predicted ranking results $R(\beta_i|B_k; w) - R(\beta_j|B_k; w)$ to be consistent with the results of $d(\beta_i, \beta_j, B_k)$ obtained by a comparison rule based on Pareto dominance.

3.2.2 Pareto Dominance Rule

To compare the performance between two architectures, we need to define a reasonable function $d(\beta_1, \beta_2, B)$ in Eqn. (3). To this end, we define a Pareto dominance to guide the design of this function. Specifically, Pareto dominance requires that the quality of an architecture should depend on both the satisfaction of budget and accuracy. That means, given a specific budget B, a good architecture should be the one with the cost lower than or equal to B and with high accuracy. In this sense, we use Pareto dominance to compare two architectures and judge which one is dominative.

Given any two architectures β_1, β_2, if both of them satisfy the budget constraints (namely $c(\beta_1) \leq B$ and $c(\beta_2) \leq B$), then β_1 dominates β_2 if $\mathrm{Acc}(\beta_1) \geq \mathrm{Acc}(\beta_2)$. Moreover, when at least one of β_1, β_2 violates the budget constraint, clearly we have that β_1 dominates β_2 if $c(\beta_1) \leq c(\beta_2)$. Formally, we define the Pareto dominance function $d(\beta_1, \beta_2, B)$ to reflect the above rules:

$$d(\beta_1, \beta_2, B) = \begin{cases} 1, & \text{if } (c(\beta_1) \leq B \wedge c(\beta_2) \leq B) \\ & \quad \wedge (\mathrm{Acc}(\beta_1) \geq \mathrm{Acc}(\beta_2)); \\ -1, & \text{else if } (c(\beta_1) \leq B \wedge c(\beta_2) \leq B) \\ & \quad \wedge (\mathrm{Acc}(\beta_1) < \mathrm{Acc}(\beta_2)); \\ 1, & \text{else if } c(\beta_1) \leq c(\beta_2); \\ -1, & \text{otherwise.} \end{cases} \tag{4}$$

Based on Eqn. (4), we have $d(\beta_1, \beta_2, B) = -d(\beta_2, \beta_1, B)$ if $\beta_1 \neq \beta_2$, making it a symmetric metric *w.r.t.* β_1 and β_2.

Remark 1 *The accuracy constraint* $\mathrm{Acc}(\beta_1) \geq \mathrm{Acc}(\beta_2)$ *plays an important role in the proposed Pareto dominance function* $d(\beta_1, \beta_2, B)$. *Without the accuracy constraint, we may easily find the architectures with very low computation cost and poor performance (See results in Table 2).*
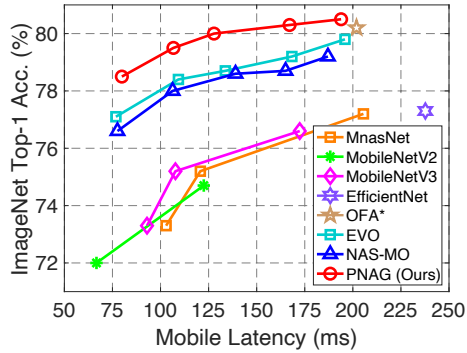
Figure 3. Comparisons of the architectures obtained by different methods on a mobile device (Qualcomm Snapdragon 821).
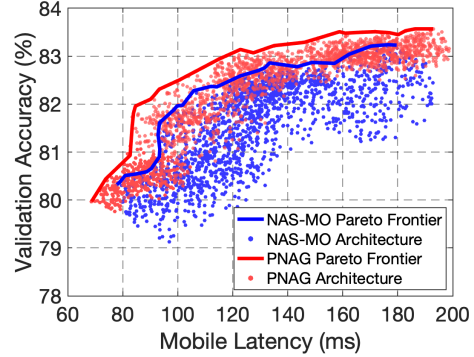

Figure 4. Comparisons of the Pareto frontiers of the generated architectures between NAS-MO and PNAG. Here, we report the accuracy evaluated on the constructed validation set.
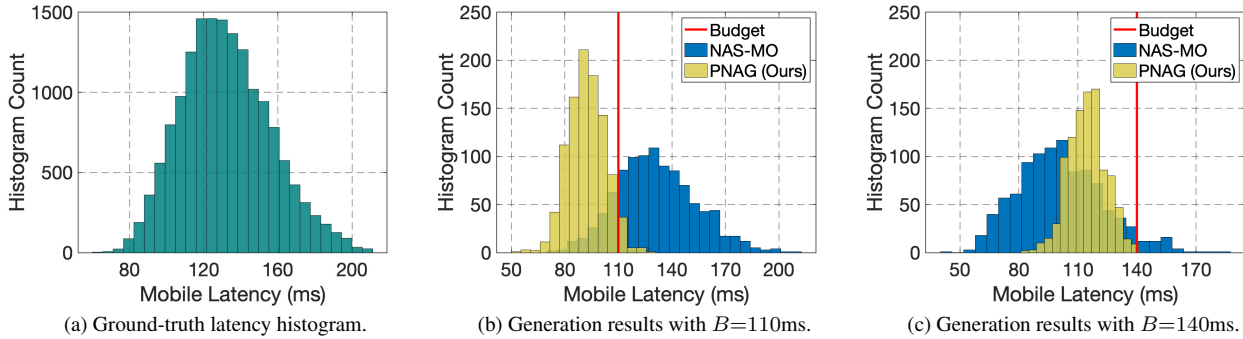

(a) Ground-truth latency histogram.

(b) Generation results with B=110ms.

(c) Generation results with B=140ms.

Figure 5. Latency histograms of sampled architectures on mobile devices. (a) Ground-truth latency histogram of $16,000$ architectures that are uniformly sampled from the search space. (b) The latency histogram of $1,000$ architectures sampled by different methods given B=110ms. (c) The latency histogram of $1,000$ architectures sampled by different methods given B=140ms.

4. Experiments

We apply the proposed PNAG to produce architectures under diverse latency budgets evaluated on three different hardware platforms, including a mobile device (equipped with a Qualcomm Snapdragon 821 processor), a CPU processor (Intel Core i5-7400), and a GPU card (NVIDIA TITAN X). For convenience, we use "Architecture-B" to represent the generated architecture that satisfies the latency budget B, e.g., PNAG-80. The results on CPU and GPU can be found in the supplementary. Our code and all the pretrained models are available at https://github.com/guoyongcs/PNAG.

4.1. Implementation Details

Following [3], we use MobileNetV3 [25] as the backbone to build the search space [3, 26]. We train the architecture evaluator for 250 epochs. The learning rate is initialized to 0.1 and decreased to 1×10^{-3} with a cosine annealing. We emphasize that training the architecture evaluator is very efficient and only takes less than *0.2 GPU hours*. We

train the architecture generator for 120k iterations using an Adam optimizer with a learning rate of 3×10^{-4}. To investigate the effectiveness of the proposed method, we compare our PNAG with two variants: 1) **EVO** uses the evolutionary search method [45] to perform architecture search. 2) **NAS-MO** conducts architecture search based by exploiting the multi-objective reward [53]. More implementation deatils can be found in the supplementary.

4.2. Architecture Search for Mobile Devices

In this experiment, we train our PNAG to produce feasible architectures for the latency budgets based on a mobile device (Qualcomm Snapdragon 821 processor). Based on the proposed budget interpolation method in Section 3.1.2, our PNAG is flexible to generate feasible architectures for any arbitrary budget. To evaluate our method, for simplicity, we manually choose 5 latency budgets {80ms, 110ms, 140ms, 170ms, 200ms} and reports the results under each of them. The other budgets are also possible.

We compare our PNAG with state-of-the-art methods given different latency budgets evaluated on he considered

Table 1. Comparisons with state-of-the-art architectures on mobile devices. * denotes the best architecture reported in the original paper. "-" denotes the results that are not reported. All the models are evaluated on 224×224 images of ImageNet.

Architecture	Latency (ms)	Test Accuracy (%)		#Params (M)	#MAdds (M)	Search Cost (GPU Days)
		Top-1	Top-5			
MobileNetV3-Large (0.75×) [25]	93.0	73.3	-	4.0	155	-
MobileNetV2 (1.0×) [49]	90.3	72.0	-	3.4	300	-
EVO-80	76.8	77.1	93.3	6.1	350	0.7
NAS-MO-80	77.6	76.6	93.2	7.9	340	0.7
PNAG-80	79.9	**78.3**	**94.0**	7.3	349	0.7
FBNet-A [62]	91.7	73.0	-	4.3	249	9.0
ProxylessNAS-Mobile [4]	97.3	74.6	-	4.1	319	8.3
MobileNetV3-Large (1.0×) [25]	107.7	75.2	-	5.4	219	-
EVO-110	109.3	78.4	94.0	10.2	482	0.7
NAS-MO-110	106.3	78.0	93.8	8.4	478	0.7
PNAG-110	106.8	**79.4**	**94.5**	9.9	451	0.7
ProxylessNAS-GPU [4]	123.3	75.1	-	7.1	463	8.3
MnasNet-A1 (1.0×) [53]	120.7	75.2	92.5	3.4	300	∼3792
FBNet-C [62]	135.2	74.9	-	5.5	375	9.0
EVO-140	133.7	78.7	94.1	9.1	488	0.7
NAS-MO-140	139.0	78.6	94.0	9.5	486	0.7
PNAG-140	127.8	**79.8**	**94.7**	9.2	492	0.7
NSGANetV1 [41]	-	76.2	93.0	5.0	585	27
PONAS-C [26]	145.1	75.2	-	5.6	376	8.8
P-DARTS [11]	168.7	75.6	92.6	4.9	577	3.8
EVO-170	168.3	79.2	94.4	10.7	661	0.7
NAS-MO-170	165.0	78.7	94.4	8.5	584	0.7
PNAG-170	167.1	**80.3**	**95.0**	10.0	606	0.7
PC-DARTS [66]	194.1	75.8	92.7	5.3	597	0.1
EfficientNet B0 [54]	237.7	77.3	93.5	5.3	390	-
Cream-L [43]	-	80.0	94.7	9.7	604	12
OFA* [3]	201.9	80.2	95.1	9.1	743	51.7
EVO-200	195.9	79.8	94.5	11.0	783	0.7
NAS-MO-200	187.4	79.2	94.4	9.1	630	0.7
PNAG-200	193.9	**80.5**	**95.2**	10.4	724	0.7

mobile device. In Fig. 3, we compare the architectures searched by different methods in terms of both accuracy and latency. We draw the following conclusions. *First*, our PNAG (red line) consistently generates better architectures than the considered variants EVO and NAS-MO under diverse budgets. *Second*, our best architecture (the rightmost point of the red line) yields a better trade-off between accuracy and latency than a strong baseline OFA*, *i.e.*, the best architecture reported in [3]. For convenience, we put more detailed comparison results in Table 1. Given diverse latency budgets, our PNAG greatly outperforms the compared NAS methods in terms of the accuracy of the generated/searched architectures. Specifically, our PNAG-200 yields the best accuracy of 80.5, which is better than the best reported results in OFA [3], namely OFA*. We also highlight that, besides the superior performance, the total training cost of our PNAG is about 0.7 GPU days, which is much more efficient than most SOTA NAS methods.

Moreover, we compare the searched frontiers of differ-

ent methods and show the comparisons of Pareto frontiers in Fig. 4. We plot all the architectures produced by different methods to form the Pareto frontier. Specifically, we use the architectures searched by multiple independent runs under different budgets for NAS-MO. For PNAG, we use linear interpolation to generate architectures that satisfy different budgets. From Fig. 4, our PNAG finds a better frontier than NAS-MO due to the shared knowledge across the search process under different budgets. We also visualize the latency histograms of the architectures evaluated on mobile devices in Fig. 5b and Fig. 5c. Given latency budgets of 110ms and 140ms, NAS-MO is prone to produce architectures that cannot satisfy the target budgets. These results show that it is hard to design the multi-objective reward to obtain the preferred architectures. Instead, PNAG uses the Pareto dominance reward to encourage the architectures to satisfy the desired budget constraints. In this sense, most architectures generated by our PNAG fulfill the target budgets. We put more visual results in the supplementary.

Table 2. Comparisons of different reward functions based on PNAG. We report the latency on mobile devices.

Reward	B_1=80ms		B_2=110ms		B_3=140ms		B_4=170ms		B_5=200ms	
	Acc. (%)	Lat. (ms)	Acc.	Lat. (ms)	Acc.	Lat. (ms)	Acc.	Lat. (ms)	Acc.	Lat. (ms)
Multi-objective Reward [53]	77.0	77.6	78.5	106.3	78.9	139.0	79.3	165.1	79.5	187.3
Multi-objective Absolute Reward [1]	78.1	76.8	78.9	109.2	79.2	130.1	79.5	163.6	79.9	197.5
Pareto Dominance Reward (w/o acc. constraint)	73.8	74.4	73.6	64.9	74.3	66.5	73.9	70.0	74.0	70.8
Pareto Dominance Reward (Ours)	**78.4**	79.9	**79.5**	106.8	**79.8**	127.8	**80.3**	167.1	**80.5**	193.9

Table 3. Effect of different search strategies on the performance of PNAG. We report the accuracy on ImageNet.

Search Strategy	B_1=80ms	B_2=110ms	B_3=140ms	B_4=170ms	B_5=200ms
Repeated Independent Search	76.7	78.6	79.1	79.4	79.7
Pareto Frontier Search	**78.4**	**79.5**	**79.8**	**80.3**	**80.5**

Table 4. Comparisons of the time cost for architecture generation/design among different methods.

Method	PNAG	PC-DARTS	ENAS	DARTS
Time Cost	**≤5 s**	2 hours	12 hours	4 days

Table 5. Effect of K on the generation performance of PNAG with the target latency B=140ms on ImageNet.

K	1	2	5	10	30
Top-1 Acc. (%)	78.5	79.1	79.4	**79.8**	**79.8**

4.3. Further Experiments

Effect of the Pareto Dominance Reward We investigate the effectiveness of the Pareto frontier learning strategy and the Pareto dominance reward. From Table 2 and Table 3, the Pareto frontier learning strategy tends to find better architectures than the independent search process due to the shared knowledge across the search processes under different budgets. Compared with two existing multi-objective rewards [1, 53], the Pareto dominance reward encourages the generator to produce architectures that satisfy the considered budget constraints. Moreover, if we do not consider accuracy constraint in the Pareto dominance reward, the architectures have low latency and poor accuracy. With both the Pareto frontier learning strategy and the Pareto dominance reward, our method yields the best results.

Comparisons of Architecture Generation Cost In this part, we compare the architecture generation cost of different methods for 5 different budgets and show the comparison results in Table 4. Given an arbitrary target budget, existing NAS methods need to perform an independent search to find feasible architectures. By contrast, since PNAG directly learns the whole Pareto frontier, we are able to generate promising architectures based on a learned generator model via *inference*. Thus, the architecture generation cost of PNAG is much less than other methods (See Table 4). In this sense, we greatly accelerate the architecture design process in real-world scenarios. These results demonstrate the efficiency of our PNAG in generating architectures.

Effect of K on the Generation Performance We investigate the effect of K on the generation performance of PNAG. Note that we evenly select K budgets from the range of latency. To this end, we consider several candidate values

of $K \in \{2, 5, 10, 30\}$. We show the Top-1 accuracies of the architectures generated by PNAG with different K on ImageNet in Table 5. Since a small number of selected budgets K cannot accurately approximate the ground-truth Pareto frontier or provide enough shared knowledge between different search processes, our method yields poor results with $K = 2$. When we increase K larger than 5, we are able to greatly improve the performance of the generated architectures. From Table 5, our method yields the best result when $K \geq 10$ and we use this setting in the experiments.

5. Conclusion

In this paper, we focus on designing effective and feasible architectures via an architecture generation process. To this end, we have proposed a novel Pareto-aware Neural Architecture Generator (PNAG) which only needs to be trained once and dynamically generates promising architectures satisfying any given budget via inference. Based on the learned Pareto frontier, our PNAG consistently outperforms existing NAS methods across diverse budgets. Extensive experiments on three hardware platforms demonstrate the effectiveness of the proposed method.

References

[1] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V. Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *IEEE International Conference on Computer Vision*, pages 14311–14320, 2020. 3, 8

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, pages 89–96, 2005. 5

[3] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 3, 6, 7

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 7

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *European Conference on Computer Vision*, volume 12346, pages 213–229, 2020. 1

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *European Conference on Computer Vision*, volume 11211, pages 833–851, 2018. 1

[7] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *IEEE International Conference on Computer Vision*, pages 12250–12260, 2021. 2

[8] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*, 2021. 1

[9] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323, 2009. 5

[10] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021. 2

[11] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *IEEE International Conference on Computer Vision*, pages 1294–1303, 2019. 2, 7

[12] An-Chieh Cheng, Jin-Dong Dong, Chi-Hung Hsu, Shu-Huan Chang, Min Sun, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. Searching toward pareto-optimal device-aware neural architectures. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–7, 2018. 3

[13] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021. 2

[14] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *IEEE International Conference on Computer Vision*, pages 12219–12228. IEEE, 2021. 2

[15] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 16276–16285, June 2021. 1

[16] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. 3

[17] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *European Conference on Computer Vision*, pages 517–531, 2018. 3

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. 1

[19] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(Nov):933–969, 2003. 5

[20] Crina Grosan and Ajith Abraham. Generating uniformly distributed pareto optimal points for constrained and unconstrained multicriteria optimization. *International Conference on Informatics and Systems*, pages 27–29, 2008. 4

[21] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Zhipeng Li, Jian Chen, Peilin Zhao, and Junzhou Huang. Towards accurate and compact architectures via neural architecture transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 4

[22] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560, 2020. 3

[23] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11990–11999, 2020. 2

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu,

Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *IEEE International Conference on Computer Vision*, pages 1314–1324, 2019. 6, 7

[26] Sian-Yao Huang and Wei-Ta Chu. Ponas: Progressive one-shot neural architecture search for very efficient deployment. *arXiv preprint arXiv:2003.05112*, 2020. 3, 6, 7

[27] Sian-Yao Huang and Wei-Ta Chu. Searching by generating: Flexible and efficient one-shot nas with architecture generator. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–992, 2021. 3

[28] Il Yong Kim and Oliver L De Weck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and Multidisciplinary Optimization*, 29(2):149–158, 2005. 1, 3

[29] Mifa Kim, Tomoyuki Hiroyasu, Mitsunori Miki, and Shinya Watanabe. SPEA2+: improving the performance of the strength pareto evolutionary algorithm 2. In *Parallel Problem Solving from Nature*, volume 3242, pages 742–751. Springer, 2004. 3

[30] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. 1

[31] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1989–1998, 2020. 1

[32] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584*, 2021. 3

[33] Guohao Li, Guocheng Qian, Itzel C. Delgadillo, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. SGAS: sequential greedy architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1617–1627, 2020. 2

[34] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance deep image recognition. In *IEEE International Conference on Computer Vision*, 2021. 2

[35] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 2

[36] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, 2021. 2

[37] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *IEEE International Conference on Computer Vision*, pages 9992–10002, 2021. 1

[38] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture

search. In *European Conference on Computer Vision*, pages 35–51. Springer, 2020. 1, 3

[39] Zhichao Lu, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Neural architecture transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2971–2989, 2021. 2

[40] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019. 1

[41] Zhichao Lu, Ian Whalen, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Multi-objective evolutionary design of deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 2020. 3, 7

[42] Ramakanth Pasunuru and Mohit Bansal. Continual and multi-task architecture search. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of Conference of the Association for Computational Linguistics*, pages 1911–1922, 2019. 2

[43] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. In *Advances in Neural Information Processing Systems*, 2020. 7

[44] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pages 4095–4104, 2018. 2, 4, 5

[45] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019. 2, 6

[46] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911, 2017. 2

[47] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1

[48] Binxin Ru, Pedro Esperanca, and Fabio Carlucci. Neural architecture generator optimization. In *Advances in Neural Information Processing Systems*, 2020. 2, 3

[49] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 7

[50] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017. 1

[51] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385, 2015. 1

[52] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Mar-

culescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497, 2019. 1

[53] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1, 6, 7, 8

[54] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019. 7

[55] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 10778–10787, 2020. 1

[56] Jan Hendrik Metzen Thomas Elsken and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019. 3

[57] Yuan Tian, Qin Wang, Zhiwu Huang, Wen Li, Dengxin Dai, Minghao Yang, Jun Wang, and Olga Fink. Off-policy reinforcement learning for efficient and effective GAN architecture search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *European Conference on Computer Vision*, volume 12352, pages 175–192, 2020. 2

[58] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *IEEE International Conference on Computer Vision*, pages 9627–9636, 2019. 1

[59] Arash Vahdat, Arun Mallya, Ming-Yu Liu, and Jan Kautz. UNAS: differentiable architecture search meets reinforcement learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11263–11272, 2020. 2

[60] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10):3349–3364, 2021. 1

[61] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems*, 34, 2021. 1

[62] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 7

[63] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34, 2021. 1

[64] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *IEEE International Conference on Computer Vision*, pages 1284–1293, 2019. 2, 3

[65] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. Renas: Relativistic evaluation of neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4411–4420, 2021. 2

[66] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. In *International Conference on Learning Representations*, 2020. 2, 7

[67] Zhicheng Yan, Xiaoliang Dai, Peizhao Zhang, Yuandong Tian, Bichen Wu, and Matt Feiszli. Fp-nas: Fast probabilistic neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15139–15148, 2021. 1

[68] Tien-Ju Yang, Yi-Lun Liao, and Vivienne Sze. Netadaptv2: Efficient neural architecture search with fast super-network training and architecture optimization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2402–2411, 2021. 2

[69] Xinbang Zhang, Zehao Huang, Naiyan Wang, Shiming Xiang, and Chunhong Pan. You only search once: Single shot neural architecture search via direct sparse optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2891–2904, 2021. 2

[70] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR, 2021. 2

[71] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019. 1

[72] Xiawu Zheng, Rongrong Ji, Yuhang Chen, Qiang Wang, Baochang Zhang, Jie Chen, Qixiang Ye, Feiyue Huang, and Yonghong Tian. MIGO-NAS: towards fast and generalizable neural architecture search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2936–2952, 2021. 2

[73] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, 2, 4

[74] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. 1