MOO-DNAS: Efficient Neural Network Design via Differentiable Architecture Search Based on Multi-Objective Optimization

HUI WEI1,2, FEIFEI LEE1,2, (Member, IEEE), CHUNYAN HU3, AND QIU CHEN4, (Member, IEEE)

1Shanghai Engineering Research Center of Assistive Devices, School of Medical Instrument and Food Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China
2Rehabilitation Engineering and Technology Institute, University of Shanghai for Science and Technology, Shanghai 200093, China
3School of Optical-Electrical and Computer Engineer, University of Shanghai for Science and Technology, Shanghai 200093, China
4Major of Electrical Engineering and Electronics, Graduate School of Engineering, Kogakuin University, Tokyo 163-8677, Japan

Corresponding authors: Feifei Lee (feifeilee@ieee.org) and Qiu Chen (q.chen@ieee.org)

This work was supported in part by the Program for Professor of Special Appointment (Eastern Scholar) at the Shanghai Institutions of Higher Learning.

ABSTRACT The progress devoted to improving the performance of neural networks has come at a high price in terms of cost and experience. Fortunately, the emergence of Neural Architecture Search improves the speed of network design, but most excellent works only optimize for high accuracy without penalizing the model complexity. In this paper, we propose an efficient CNN architecture search framework, MOO-DNAS, with multi-objective optimization based on differentiable neural architecture search. The main goal is to trade off two competing objectives, classification accuracy and network latency, so that the search algorithm is able to discover an efficient model while maintaining high accuracy. In order to achieve a better implementation, we construct a novel factorized hierarchical search space to support layer variety and hardware friendliness. Furthermore, a robust sampling strategy named “hard-sampling” is proposed to obtain final structures with higher average performance by keeping the highest scoring operator. Experimental results on the benchmark datasets MINST, CIFAR10 and CIFAR100 demonstrate the effectiveness of the proposed method. The searched architectures, MOO-DNAS-Nets, achieve advanced accuracy with fewer parameters and FLOPs, and the search cost is less than one GPU-day.

INDEX TERMS Differentiable neural architecture search, CNNs, multi-objective optimization, accuracy-efficiency trade-off.

1. INTRODUCTION

Convolutional neural networks (CNNs) have made great achievements in a variety of computer vision applications, including image recognition [1]–[5], object detection [6]–[8] and semantic segmentation [9], etc. An important reason for this successive progress is that various but effective CNN architectures have been elaborately designed. However, traditional CNN models have to be deeper or larger to meet the demands of higher performance, resulting in higher computational resources and making it more difficult to reproduce models on resource-constrained mobile or embedded devices. As a result, most recent research focuses on developing efficient CNN models with improved performance for intelligent devices.

Over the years, there are a number of approaches to address the issue of complexity and efficiency. Existing model compression techniques such as network pruning [10], [11], quantization [12], [13], and knowledge distillation [14] can be used to study compact deep neural networks by trading accuracy for efficiency. However, these techniques inevitably result in performance degradation due to information loss and the compressed models are usually the upper bound by pre-trained models. On the other hand, handcrafting novel and resource-constrained models is also a challenge work, which depends heavily on researchers’ experiences and requires a lot of trials-and-errors. In fact, it is hard to design light-weight neural networks [15]–[19], because researchers need to consider many influencing factors, including connection
methods, depth or width, and convolutional computation. Moreover, it is necessary to constantly fine-tune the architectures according to the performance on the target device. In a conclusion, the design of efficient neural networks not only requires a wealth of expertise knowledge and domain knowledge, but also relies on manual experience.

Limited by the cost of training time and computing resources, it is natural to push automatic techniques for designing efficient neural networks [20]–[26]. Neural Architecture Search (NAS) has been adopted in many tasks with outstanding achievements. It refers to discover high-performance neural network structures in a particular search space using a specific search strategy. Earlier works utilize reinforcement learning (RL) [27]–[30] to guide the search, which construct a controller to sample architectures from the search space for final evaluation. Shortly thereafter, evolutionary algorithms (EA) [31]–[33] are used to explore the search space. The population consists of initial network architectures, and the novel architectures are generated by mutating or reproducing. Later, Bayesian optimization [34], SMBO [35], [36], and MCTS [37] are also adopted in NAS. However, these approaches require hundreds of GPU hours because they employ a discrete search space and the search process is regarded as a black-box optimization. For final evaluation, the sampled architectures must be trained from scratch again and again, which takes time. Differentiable Neural Architecture Search (DNAS) [26], [38]–[40] models the search process as training a continuous mathematical distribution, and the distribution can be optimized by gradient information so as to escape from a large number of repeated evaluations. It is a bi-level optimization: architecture parameters and weight parameters for the combined optimization of the architectures and weights. The formulation is sophisticated and conceptually sound.

However, the above-mentioned works usually model NAS as a single-objective optimization problem, which only pursues high classification accuracy without penalizing the model complexity. Thus, the searched architectures contain a large number of redundant operators and connections, so that the architectures are limited to be implemented on mobile platforms and efficient scenarios. To this end, many works focus on multi-objective optimization problem in considering both accuracy and efficiency of models. Specifically, the search objective contains not only the metric of accuracy but also the metric of efficiency, such as latency, power, energy, parameters and FLOPs. These metrics depend on the complexity of architectures and the configuration of devices. For easier implementation, most works focus on hardware-independent metrics like FLOPs and parameters. It has been observed that FLOP is often an inaccurate proxy. The FLOPs of MobileNets [15] and NASNet [29] are similar, but their latencies are drastically different. Thus, the discrepancy between hardware-independent and hardware-related metrics makes it hard to design efficient neural networks. Furthermore, the design of search space is critical. Many excellent works, such as NASNet [29], DARTS [38] and SNAS [39], adopt cell-based structure to simplify the search process. However, they sacrifice layer diversity which is crucial for model efficiency and bring network fragmentation to reduce degree of parallelism. Other researches, such as MnasNet [20], FBNet, [26] adopt MobileNet-based search space to search for efficient network module. In our work, we primarily focus on inference latency to control the efficiency of architectures and explore the MobileNet-based search space.

In this research, we present a novel approach for automating the design of light-weight neural networks via differentiable neural architecture search, allowing the searched architectures to be executed efficiently on resource-constrained platforms. The flow of our algorithm is illustrated in Fig. 1. We first define neural architecture search as a multi-objective optimization problem which considers the classification accuracy and the inference latency, so the loss function includes both metric counterparts. Secondly, we construct a factorized hierarchical search space for each layer to choose a different block. And the search space is represented by a stochastic supernet, which is modeled with a set of architecture parameters $\theta$ w.t.t. candidate blocks, multiplied by a mask variable $m$. The candidate blocks are a set of inverted residual bottleneck structures with varying hyperparameters. Next, we relax architecture parameters into continuous by employing the Gumbel Softmax [41] technique, so that the architecture distribution can be trained via gradient-based optimization. Finally, a discrete architecture is derived by hard-sampling technique which retains the highest score operator, or soft-sampling that randomly samples from the architecture distribution yielding the optimal architecture. The loss function used to train the stochastic supernet includes the cross-entropy loss of accuracy and the latency loss. For the latency term, we develop a latency lookup table (LUT) according to the latency of each candidate operator in different layers. The total latency is then the linear summation of the latency of the chosen blocks. This model allows the search algorithm to quickly estimate the latency in a large search space, and it makes the latency term differentiable w.r.t. operation choices.

**FIGURE 1. The overview of our proposed method named MOO-DNAS.**

We conduct efficient experiments on the benchmark MNIST, CIFAR10 and CIFAR100 classification datasets to verify the accuracy and efficiency of the searched architectures. MOO-DNAS (h), by hard-sampling, achieves an average error rate of 0.56% with 3.0M parameters/ 300M FLOPs on the MNIST and the competitive error rate of 3.31% with 3.5M parameters/ 360M FLOPs on CIFAR10.
The soft-sampling architecture, MOO-DNAS (s), achieves the error rate of 0.54% with 3.3M parameters/320M FLOPs on the MNIST and the state-of-the-art error rate of 3.24% with 4.1M parameters/420M FLOPs on CIFAR10. When transferred to CIFAR100 dataset, the searched architectures on CIFAR10 achieve an average error rate of 19.93%. MOO-DNAS-Nets surpasses the most advanced efficient neural networks designed manually and automatically. The accuracies of them are better and the model sizes are smaller. Moreover, we also demonstrate the remarkable efficiency improvement of our search algorithm: the search cost is several orders of magnitude faster than non-differentiable approaches, and the total search time is less than one GPU-day on the MNIST and CIFAR10 datasets with the limited computational resource device.

In summary, the contributions of our paper are as the following four aspects:

(I) We propose an efficient search framework to design light-weight neural networks, namely MOO-DNAS. This algorithm is based on a differential neural architecture search and the main goal is to trade off two conflicting objectives, classification accuracy and network inference latency.

(II) We construct a factorized hierarchical search space to enable layer diversity and hardware-friendliness, while striking an appropriate balance between flexibility and search space size.

(III) Inspired by the argmax function, we propose a robust “hard-sampling” technique to obtain the final architectures. The sampled architectures can get a higher average accuracy than ones from random choice.

(IV) The proposed search framework is able to discover light-weight neural networks in about one day. Experimental results show that the searched architectures, MOO-DNAS-Nets, can achieve competitive accuracy with fewer parameters or FLOPs on the MNIST and CIFAR10/100 datasets.

The remainder of this work is structured as follows. Section II provides a brief overview of several studies that are related to our research. The proposed MOO-DNAS for light-weight CNN automated designing is presented in Section III. Section IV displays the experimental results and related discussion. Finally, in Section V, we bring this research a conclusion.

II. RELATED WORKS

In this section, we briefly introduce two aspects of previous works related to our study: (1) Efficient neural networks by model compression and compact designing; (2) neural architecture search algorithms and related methods for efficient network designing. We analyze the prevalent methods and clarify the differences between them.

A. EFFICIENT NEURAL NETWORKS

Reducing the computational resources of models has been a popular research topic in recent years. There are two straightforward ways to solve the complexity and efficiency issues: one is model compression, and the other is compact model designing.

Current compression techniques such as pruning [10], [11], quantization [12], [13] and distillation [14] can reduce the complexity of models. Han et al. [10] propose to prune insignificant weight parameters with low scores in neural networks, channel pruning [11], [42], [43] further removes the redundant channels from feature maps, and L1-norm regularization is utilized in Li et al. [43] to prune filters according to device-aware metrics. Xnor-Net [12] quantizes the weight parameters and activate functions with 1-bit data to simplify the model by binary technique. Jacob et al. [13] propose a quantization theme which relies only on integer arithmetic to approximate the flop computations of a neural network. Tensor decomposition [44] exploits light-weight operators such as $1 \times 1$, $3 \times 3$ convolutions to reduce the parameters and computation. Knowledge distillation [14] transfers knowledge from a large “teacher” network to a small “student” one, so as to improve the performance of the small model. However, the determination of the sparsity ratio of each layer is a recurring challenge in these approaches. The effectiveness of these methods is typically based on pre-trained models and does not focus on the building of innovative structures. Thus, further researches on operators and architectures will make them better.

The compact model designing for mobile devices with fewer calculations has also attracted the interest of researchers. Xception [45] uses depth-wise convolution to unify the different branches of feature maps for fuller use of weights. Recently, a series of manual light-weight neural networks are proposed, which focus on FLOPs rather than parameter sizes. MobileNets [15] propose depth-wise separable convolutions to approximate the original convolutions with less computational complexity. MobileNetv2 [16] proposes a search space of light-weight operations with efficient inverted residuals and linear bottlenecks. ShuffleNet [17] employs group convolution to reduce FLOP count and shuffle operation to increase information flow between groups. ShuffleNetv2 [18] takes into account the actual speed of an efficient neural network on the target device. GhostNet [19] utilizes the lower cost operator of linear transformation to generate more feature maps.

Although these manual network designing can achieve advanced performance, handcrafting novel compact models is engineering expensive and heavily depends on the expertise and experience of experts. This no doubt brings a challenge of manually network designing.

Abbreviations: RL: Reinforcement Learning, EA: Evolutionary Algorithm, PSO: Particle Swarm Optimization, GD: Gradient Descend, BO: Bayesian Optimization, MCTS: Monte Carlo Tree Search, SMBO: Sequential Model-based Optimization.

B. NEURAL ARCHITECTURE SEARCH

Recently, there has been a surge in interest in employing NAS to automatically design network models. These methods
mainly include reinforcement learning (RL) [27]–[30], evolutionary algorithms (EA) [31]–[33], gradient-based [38]–[40], and other optimization algorithms [35], [37]. Furthermore, many researches focus on multi-objective issues [20]–[26], in which measurements of resource efficiency and predictive performance are added into objectives.

In order to generate architectures from the search space, RL-based approaches build an RNN controller, which is updated with an appropriate policy depending on the architecture evaluation. Zoph and Le [28] utilize REINFORCE policy gradient to train the RNN controller, and NASNet [29] uses Proximal Policy Optimization (PPO) instead. EA-based approaches use evolutionary algorithms to get the optimal architecture. The initial populations consist of the simplest structures, then the novel architectures through mutation or reproduction are evaluated for fitness. However, these approaches need to evaluate the searched architecture repeatedly, which is computationally expensive. Current works try to accelerate the process by weight sharing or one-shot training. ENAS [47] uses weight sharing among child models to get rid of redundant training. Single-path with uniform sampling [48] is a powerful NAS framework, which constructs a simplified Single Path One-shot model to be trained by uniform path sampling. Once the supernet has been trained, the evolutionary algorithm is applied to search the best-performance architecture without fine-tuning. In addition, differentiable architecture search (DNAS) aims to relax the architecture information into continuous for gradient descent to optimize. DARTS [38] proposes deterministic attention on operations to analytically calculate expectation at each edge of the direct acyclic graph. SNAS [39] and proxylessNAS [40] suggest optimizing a parameterized distribution across the possible operations rather than weights of possible operations.

MnasNet [20] is the first study to search efficient neural network with RL-based NAS. It models the search process as a multi-objective problem with accuracy and speed and presents a hierarchical search space to allow different blocks for different phases. FBNet [26] further develops differentiable neural architecture search combining Gumbel Softmax [41] to explore the efficient inverted residual block search space. DPP-Net [21], LEMONADE [22], CARS [24] and MOOSO/D [25] consider device-related and device-agnostic metrics into objectives, such as model size and accuracy. NSGA-Net [23] considers FLOPs and accuracy. Instead of searching novel architectures,
MobileNetv3 [49] utilizes NetAdapt [50] technology to get better performance with fewer FLOPs, and AMC [51] improve the quality of model compression with reinforcement learning.

Multi-objective NAS is similar to network compression in that both seek highly accurate and efficient topologies. As a result, some compression methods can be also regarded as NAS methods and vice versa. We also model the design of efficient neural networks as multi-objective optimization, taking both the accuracy and latency into account.

The analysis of the related works in NAS is summarized in Tab. 1, and these works are concluded in three parts: Evaluation Metric, Search Space and Search strategy.

III. PROPOSED METHODS

Our method aims to search for light-weight architectures in an efficient way. We first build a multi-objective optimization based on accuracy and efficiency. Then, a novel hierarchical search space is constructed to enable layer diversity and hardware-friendliness. In order to evaluate the latency of networks, we build a lookup table (LUT) based on the latency of each operator and the total network latency is the sum of every chosen operators. Finally, the search space is represented as a supernet whose operators are executed stochastically. The problem of searching for the optimal architecture is relaxed to find an architecture distribution. The distribution is trained by gradient descent and the architectures are sampled from this distribution.

A. MULTI-OBJECTIVE FORMULATION

In this research, we formulate the design of neural networks as a multi-objective optimization to enhance both accuracy and efficiency. Instead of optimizing direct hardware-agnostic metrics of FLOPs or Params as in earlier works, we incorporate the inference latency of models into our agnostic metrics of FLOPs or Params as in earlier works, and uses of both the accuracy and latency into account.

The analysis of the related works in NAS is summarized in Tab. 1, and these works are concluded in three parts: Evaluation Metric, Search Space and Search strategy.

III. PROPOSED METHODS

Our method aims to search for light-weight architectures in an efficient way. We first build a multi-objective optimization based on accuracy and efficiency. Then, a novel hierarchical search space is constructed to enable layer diversity and hardware-friendliness. In order to evaluate the latency of networks, we build a lookup table (LUT) based on the latency of each operator and the total network latency is the sum of every chosen operators. Finally, the search space is represented as a supernet whose operators are executed stochastically. The problem of searching for the optimal architecture is relaxed to find an architecture distribution. The distribution is trained by gradient descent and the architectures are sampled from this distribution.

A. MULTI-OBJECTIVE FORMULATION

In this research, we formulate the design of neural networks as a multi-objective optimization to enhance both accuracy and efficiency. Instead of optimizing direct hardware-agnostic metrics of FLOPs or Params as in earlier works, we incorporate the inference latency of models into our objectives. Given an architecture \( a \), let \( ACC(a) \) be denoted as its accuracy, while \( LAT(a) \) as the inference latency on the target platform measured in micro second, and \( T \) is the target latency. Thus, the problem is to maximize the accuracy under hard latency constraints:

\[
\max \ a \ ACC(a) \quad s.t. \ LAT(a) \leq T \tag{1}
\]

The above formula only maximizes a single accuracy metric and uses the hard constraint of latency. The solution does not reflect multiple Pareto optimal, which is said that: if none of the objectives can be improved without affecting some of other objectives, this state reaches Pareto balance and the solution is to be in the Pareto front.

Given to design an architecture with high accuracy and low inference, we are more interested in finding multiple Pareto solutions. The neural architecture search in our paper is modeled as minimizing the loss function reflecting the relationship between parameters and architectures:

\[
\min \ \min_{a \in A} \ L(a, w_a; \theta) \tag{2}
\]

where the architecture \( a \in A \) is controlled by the architecture parameters and corresponding weight parameters. The loss function not only reflects the accuracy but also the latency on the target hardware:

\[
L(a, w_a) = CE(a, w_a) \cdot \alpha \log(LAT(a))^\beta \tag{3}
\]

where CE denotes the cross-entropy loss of the architecture with the parameters \( w_a \), and the coefficient \( \alpha, \beta \) respectively control the overall loss function and the magnitude of latency term.

B. FACTORIZED HIERARCHICAL SEARCH SPACE

It is observed that the previous methods [29], [38], [39] mainly focus on the cell-based search space, and the searched cell architecture is repeated for several times to build the final neural network. This simplifies the search process, but eliminates the layer diversity. Inspired by the previous works MnasNet [20] and FBNNet [26], we construct a novel factorized hierarchical search space. As shown in Fig. 2, it has a manually fixed macro architecture and automatically searches different micro architectures separately within different stages. The overall network is grouped into a predefined skeleton call stages with fixed input resolutions and filter numbers. Each stage contains \( N \) layers, where the first layer has the stride of 2 for resolution reduction while all others have the stride of 1. For each layer, we search for the different block types in the micro search space to form the final efficient model.

1) MACRO TOPOLOGY

The macro topology defines the structures of stem head, classification head, the number of stages, and the input/output dimensions of each stage.

An issue of FBNNet [26] lies in that the predefined macro architecture is fit to large ImageNet [52] dataset rather than CIFAR10, because the image resolution is \( 224 \times 224 \times 3 \) in ImageNet while \( 32 \times 32 \times 3 \) in CIFAR10. Since the network structure of VGG-16 is designed for ImageNet dataset, we inspire from its variant of the Github project [56] which is widely used in literatures to construct the macro topology described in Tab. 2. Specifically, the structure of stem head is one \( 3 \times 3 \) convolutional layer followed by a BN layer and a ReLU activate as shown in Fig. 3(a), and classification head is one \( 3 \times 3 \) convolutional layer followed by a global average pooling with a full-connection as shown in Fig. 3(b). For the rest of the model, network is divided into several stages with diverse blocks whose filter numbers are hand-picked empirically. The information flow between adjacent layers is Eq. (4), where \( b_l \) denotes a specific block type described in Sec. 3.2.2.

\[
x_{j+1} = b_l(x_j) \tag{4}
\]

Our goal is to search for the appropriate operators for better accuracy-latency trade-off. Given a widely used light-weight depthwise separable convolution denoted as \((k, k, C_{in}, C_{out})\),
it transforms the input of size \((H, W, C_{in})\) to the output of size \((H, W, C_{out})\). The total number of parameters and multiply-adds is:

\[
\text{params} = \left( k^2 + C_{out} \right) \times C_{in} \tag{5}
\]

\[
\text{FLOPs} = 2 \left( k^2 + C_{out} \right) \times H \times W \times C_{in} \tag{6}
\]

where the input/output resolutions are equal with size of \((H, W)\), \(C_{in}, C_{out}\) is the input/output channel size and \(k\) is the kernel size.

For efficient neural network designing, we must carefully balance the kernel size \(k\) and the channel size \(C_{out}\). For example, increasing the receptive field by increasing the kernel size must be balanced by decreasing the channel size. For early stages, CNN models usually process larger spatial resolution, and the selection of channel size \(C_{in}\) is relatively small, because the spatial resolution is larger than later, and the computational cost is quadratic with the input resolution from Eq. (6). For later stages, we gradually reduce input resolutions and increase kernel sizes.

2) MICRO TOPOLOGY

Mobile neural networks have been building more efficient blocks. MobileNets [15] proposes depth-wise separable convolution to replace vanilla convolution with two separate layers. The first is a light-weight depth-wise convolution for filtering each channel, and the second is a heavier \(1 \times 1\) point-wise convolution for computing linear combinations of each channel. MobileNetv2 [16] introduces the inverted residual module with linear bottleneck as shown in Fig. 4 to build more efficient networks by leveraging the low rank nature of features.

We use a combination of these efficient modules as building blocks. Each block has a list of operations, whose operators and connection methods are defined in Fig. 5. It consists of a \(1 \times 1\) pointwise convolution for expansion, a \(k \times k\) depthwise convolution for feature generation, and another \(1 \times 1\) convolution for inverted projection. “ReLU” activation does not follow the last \(1 \times 1\) convolution, because it is necessary to remove non-linearities of the thin layers to maintain...
representational power. If the output and input dimension are the same, the short-cut is inserted to add the input to the output. This connection can form inverted residual connection described in MobileNetv2 [16], which contributes to alleviate gradient vanishing.

![FIGURE 5. The structure of micro topology. Each candidate blocks can select a different expansion rate, kernel size, groups number and whether or not to short-cut.](image)

To manage the micro topologies for light-weight network searching, a series of hyper-parameters such as the kernel size $k$, expansion ratio $e$, and group number $g$ are specified. It influences how much the input channel size is expanded with the first $1 \times 1$ convolution and the size of the filter kernel for the depthwise convolution. Furthermore, we employ group convolution for the first and last pointwise convolutions to reduce computational complexity, and the channel shuffle technique is used for information communicating. As a result, the following options comprise the micro topologies for a block $i$:

- Operation types: inverted bottleneck module, skip-connection.
- Kernel size $k$: 3 x 3, 5 x 5.
- Expansion ratio $e$: 1, 3, 6.
- Group number: 1, 2.

In conclusion, our factorized hierarchical search space offers the obvious benefit of balancing layer diversity with search space capacity. We can adjust the hyper-parameters of architecture to change the size of neural network, which is convenient for us to design efficient model. However, supposing that we partition the network into $N$ layers and there are $C$ choices of layer types, then the total search space size would be $C^N$. In our work, the layer $N = 15$, $C = 9$, so that the search space contains $9^{15} \approx 10^{14}$ possible architectures. It is a non-trivial work to explore the huge search space.

**C. LATENCY PREDICTOR**

Besides accuracy, latency is another important metric when designing efficient neural network architectures for hardware platforms. Unfortunately, unlike accuracy with cross-entropy that can be optimized easily by gradient information, latency is harder to estimate. Because we need to measure the actual runtime of the model on the target device and there are about $10^{14}$ architectures in the huge search space.

To simplify this problem, we build a latency lookup table model to estimate the overall latency of a model. The latency lookup table lists the runtime of each operator in different layers, and the runtime is measured with a specific input resolution 50 times to obtain an average for the final accurate estimation. Each operator’s runtime is independent of the other operators. The latency of overall architecture $a$ is the linear summation of the latency of every block:

$$\text{LAT}(a) = \sum_l \text{LAT}(\tilde{b}_l^{(a)})$$

where $\tilde{b}_l^{(a)}$ denotes the operation block of layer $l$ in architecture $a$.

By adding up the runtime of operators, we can simply estimate the latency inference of the $10^{14}$ architectures. More crucially, the latency lookup table renders the latency term in Eq. (3) differentiable w.r.t. block selection, allowing us to optimize the multi-objective problem using gradient descent.

**D. DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH**

It is difficult to solve the multi-objective problem Eq. (3) through the huge search space. The search process consists of two-level optimizations: the inner optimization of $w_a$ involves training an ordinary neural network and the outer optimization of $\theta_a$ controls the architecture selections in the huge search space.

We use a gradient-based technique to guide the exploration rather than reinforcement learning [20]. The search space is initially represented as a stochastic supernet, the macro topology of which is given in Sec. (III. B. 1) and the micro topology in Sec. (III. B. 2). During the inference of the supernet, just one candidate block is sampled with a sampling probability of:

$$P_{\theta}(b_l = b_{l,i}) = \text{softmax} \left( \theta_{l,i}; \theta_l \right) = \frac{\exp(\theta_{l,i})}{\sum_i \exp(\theta_{l,i})}$$

where $\theta_l$ denotes architecture parameters which control the sampling probability of each block at layer-$l$. To determine the specific block type at layer-$l$, we set a random variable $m_{l,i}$ in $\{0, 1\}$ which $m_{l,i} = 1$ if block $b_{l,i}$ is sampled and $m_{l,i} = 0$ if block $b_{l,i}$ is masked. The output of layer-$l$ can be converted from Eq. (4) to:

$$x_{l+1} = \tilde{b}_l(x_l) = \sum_i m_{l,i} \cdot b_{l,i}(x_l)$$

where $b_{l,i}$ denoted the candidate block of block-$i$ at layer-$l$.

Assuming that each layer samples independently, the likelihood of sampling an architecture-$a$ is as follow:

$$P_{\theta}(a) = \prod_l P_{\theta}(b_l = b_{l,i}^{(a)})$$

where $\theta$ controls the block selection and $b_{l,i}^{(a)}$ denotes the sampled block-$i$ at layer-$l$. Instead of exploring though a discrete search space directly, we relax the problem to train a
probability $P_o$ of the stochastic super net. Thus, the discrete optimization problem is relaxed as:

$$\min_{\theta} \min_{w_a} E_{a \sim P_o}(L(a, w_a))$$

(11)

An intuitive interpretation of this objective is to optimize the expected performance of the sampling probability with $P_o(a)$. This is different from attention-based NAS like DARTS [38], which avoids the sampling process by taking analytical expectation for each operator in every edge. Note that the loss function Eq. (11) is differentiable w.r.t. the weight parameters $w_a$ while not to architecture parameters $\theta$, because the gradient information cannot pass the discrete random variable $m_{i,j}$ to $\theta_{i,j}$. To solve this problem, we use concrete distribution [41] to relax the discrete architecture distribution into continuous with reparameterization trick:

$$m_{i,j} = \text{GumbelSoftmax}(\theta_{i,j} | \theta_l) = \frac{\exp[(\theta_{i,j} + g_{l,i})/\tau]}{\sum_i \exp[(\theta_{i,j} + g_{l,i})/\tau]}$$

(12)

where $g_{l,i} \sim \text{Gumbel}(0, 1)$ is a random noise following the Gumbel distribution, $g_{l,i} = -\log(-\log(u_{l,i}))$, $u_{l,i}$ is a uniform random variable. $m_{i,j}$ is the soften one-hot variable. $\tau$ is the temperature of the softmax, which is steadily annealed to be close to zero. Thus, the mask variable is differentiable w.r.t. architecture parameters $\theta_{i,j}$.

For latency, Eq. (7) can be converted into:

$$LAT(a) = \sum_i \sum_j m_{i,j} \cdot LAT(b_{l,i})$$

(13)

where the latency $LAT(b_{l,i})$ is a constant coefficient, so the overall latency is differentiable w.r.t. mask $m_{i,j}$, therefore $\theta_{i,j}$. Thus, the loss function Eq. (3) is differentiable w.r.t. both weight parameters $w_a$ and architecture parameters $\theta$.

The search process is equal to train a stochastic supernet. During the training, we alternatively train the weight parameters and architecture parameters. It is the first to compute $\partial L / \partial w$ to learn each operation’s weights in the supernet, which is the same as training an ordinary neural network. After operators get trained, different candidate blocks have different contributions to the accuracy and the latency. Therefore, we compute $\partial L / \partial \theta$ to update the sampling probability $P_o$, and select more appropriate operators with higher accuracy and lower latency. After the supernet has been trained, the optimal architecture is derived by sampling from the architecture distribution $P_o$. The sampling technologies contain two different methods: hard-sampling and soft-sampling. Hard-sampling is to directly get operator with the largest weight by $architecture = \arg\max_{\theta} \exp(\theta)$, while soft-sampling is to randomly choose from the distribution $P_o$. We conduct different sample policies in our experiments to compare two approaches and make a conclusion about different techniques for their advantages and disadvantages.

IV. EXPERIMENTS AND DISCUSSIONS

To verify the effectiveness of our method, we conduct neural architecture search for efficient neural networks on MNIST, CIFAR10 and CIFAR100 datasets. Following the current NAS baselines [28], [29], the search process is partitioned into two stages: the architecture search phase and the evaluation phase. In the search phase, we train the stochastic supernet to find an optimal architecture distribution $P_o$. In the evaluation phase, we sample architectures from the trained distribution. Then, the architectures are trained from scratch to determine the final weight parameters and we evaluate their accuracy and efficiency on the test set. In addition, our two key contributions are: the improved search space and a new “hard-sampling” technique. Thus, we compare search spaces of different sizes with our search space in the ablation study and conduct an ablation study for the comparison between hard-sampling and soft-sampling.

A. DATASETS AND METRICS

1) DATASETS

MNIST dataset contains 70k handwritten digit images with a spatial resolution of $28 \times 28$, in which 60k images are the training images and 10k are the test images. CIFAR10 and CIFAR100 consist of 60k images with a spatial resolution of $32 \times 32$, in which 50k images are the training images and 10k images are the test images. CIFAR10 categorizes images into 10 classes, while CIFAR100 categorized into 100.

The stem head in Sec. III(B) is set for the images of $32 \times 32$, while the input size of MNIST is $28 \times 28$. Thus, we first resize the input of $28 \times 28$ into $32 \times 32$, so as to fit the macro search space.

2) METRICS

Our goal is to efficiently discover models with high accuracy and minimal latency on target devices. The loss function includes cross-entropy loss between architecture and corresponding weights, as well as latency estimation. We employ latency metric to guide efficient neural network search in our search process. The primary metrics we concern about are the classification error rate on the test set and the inference latency on the target device, but the latter is not consistent due to varied devices. For better comparison, we employ Params and FLOPs as secondary efficiency measurements.

In addition, the efficiency of search algorithm is also critical for NAS. We compare the search cost with automatically designed approaches.

B. EXPERIMENTAL DETAILS

First, we build a latency lookup table based on the target device. The latency of specific operator is calculated to be under batch size 1 by 50 times to get average on a server with 2.40GHz Intel(R) Xeon(R) CPU E5-2630 v4.

Next, we train the stochastic supernet for 90 epochs with a batch size of 96. During the training, the training set is divided into two subsets: 80% is for weight parameters $w_a$ and 20% for architecture parameters $\theta$. The splitting of training weights and architectures ensure the architecture generalization to validation dataset. In each epoch, we first use momentum SGD to optimize weight parameters $w_a$ with an initial
learning rate \( \eta_w = 0.1 \) (decay following a cosine decaying schedule), a momentum of 0.9, and a weight decay of \( 10^{-4} \). Then, we train the architecture parameters \( \theta \) with Adam [53] with a learning rate of 0.01 and weight decay of \( 5 \times 10^{-4} \). For the loss function, \( \alpha = 0.2 \), \( \beta = 0.6 \).

After the stochastic supernet has been trained, we derive from the architecture distribution \( P_\theta \) by hard-sampling and soft-sampling. And the sampled architectures are trained by momentum SGD from scratch with a batch size of 96, the initial learning rate is 0.025 (decay following a cosine schedule), a momentum of 0.9, and a weight decay of \( 3 \times 10^{-4} \). We then use additional enhancement cutout with a length of 16. All of the settings of experiments are consistent with the MnasNet [20] and FBNet [26]. In addition, some methods of standard augmentations are used on datasets before training, such as random crop, random affine, random horizontal flip and vertical flip.

C. COMPARISON WITH STATE-OF-THE-ART METHODS

1) MINST

We first train the supernet on the MNIST dataset to obtain the architecture distribution, and sample the final architectures from the distribution by hard-sampling and soft-sampling. The sampled architectures are shown as Fig. 6, and we train them from scratch on the MNIST dataset for evaluation. Next, the CNNs found are compared with current state-of-the-art manual and automated models as shown in Tab. 3, including ResNet, MnasNet, IPPSO, psoCNN and MOOSO/D-Net. The Error Rate, Params, FLOPs and Search Cost are cited from their original references.

As observed from the Tab. 3, MOO-DNAS(h) achieves an error rate of 0.56\% with 3.0M parameters/ 300M FLOPs and MOO-DNAS(s) an error rate of 0.54\% with 3.3M/ 320M FLOPs. They achieve the average classification accuracy compared to the state-of-the-art works. The testing accuracy of MOO-DNAS-Nets is first 0.15\% lower than the baseline ResNet. As for the automatically designed models, MOO-DNAS-Nets outperform two models from MnasNet and IPPSO, but perform worse than two models from psoCNN and MOOSO/D-Net. Regarding the complexity of models, the proposed method is able to discover compact and efficient CNNs. MOO-DNAS-Nets have about 3.1M Params and 310M FLOPs, which are similar to psoCNN and fewer than MnasNet.

Note that MOOSO/D-Net outperforms ours in each of evaluation metrics. The reason for this phenomenon is that: the search space in ours is MobileNet-based, while the search space in MOOSO/D is global-based. MOOSO/D can adjust the size of search space based on decision variable, so as to find the most suitable search space.

2) CIFAR10 AND CIFAR100

Consistent with the MNIST experiment, we first train the supernet on CIFAR10 to get the architecture distribution, and sample architectures from distribution by hard-sampling and soft-sampling. The searched architectures are shown as Fig. 7(a)(b), and we train them from scratch on the CIFAR10 and CIFAR100 dataset for evaluation. Next, the searched networks are compared with various state-of-the-art models designed manually or automatically as shown in Tab. 4 and Tab. 5.

It can be seen from Tab. 4 that MOO-DNAS (h) achieves an error rate of 3.31\% with 3.5M parameters/360M FLOPs and MOO-DNAS (s) an error rate of 3.24\% with 4.1M parameters/420M FLOPs, which is comparable with the state-of-the-art accuracy for efficient model design. It can be considered that the proposed method outperforms all manually designed efficient models in terms of error rate. The manual models used for comparison include original VGG-16 [54] and ResNet-56 [1], as well as corresponding variants such as model compression, \( l_1 \) pruning [43], SBP [42] channel splitting (CP) [42] and AMC [51]. For compact network designing like GhostNet [19] or MobileNets [15],[16], our architectures also surpass them. Although we do not optimize parameter sizes or FLOPs directly, the size of our models stays consistent with the light-weight MobileNet-series, because the basic block of our search space is inverted residual bottleneck structure.

There have been many NAS approaches recently, but a lot of them aim at searching for networks with high accuracy rather than accuracy-efficiency trade-offs. Our goal is to automatically design a light-weight neural network based on a specific device. For the efficient-aware NAS methods, our approach also achieves competitive performance and the search cost is significantly lower than earlier approaches. It reduces the search cost to less than one GPU-day, when typical NAS approaches require multiple GPU servers for more than one day.

Then, we directly transfer our searched architectures on CIFAR10 to CIFAR100. It can be seen from Tab. 5 that our error rate achieves a competitive performance compared with automated design methods, and is higher than ResNet [1] which has a larger model size.

D. ABLATION STUDY

1) IMPROVED SEARCH SPACE

In this subsection, we explore the influence of different search space sizes and get an appropriate balance between the error rate and complexity of neural networks.
TABLE 3. Results compared with current advanced neural networks on the MNIST dataset. The search cost can be shortened into 0.3 GPU-days on Tesla P100 and 0.2 GPU-days on Tesla V100.

| Architecture    | Search Method | Error Rate (%) | Params (M) | FLOPs (M) | Search Cost (GPU-days) |
|-----------------|---------------|----------------|------------|-----------|------------------------|
| ResNet [1]      | Manual        | 0.41           | -          | -         | -                      |
| MnasNet [20]    | RL            | 0.82           | 3.5        | -         | 1200                   |
| IPSO [58]       | EA            | 1.21           | 1.47       | -         | 0.2                    |
| psoCNN [57]     | EA            | 0.44           | 3.26       | -         | 0.3                    |
| MOO/D-Net [25]  | EA            | 0.40           | 0.16       | -         | 0.25                   |
| MOO-DNAS (h)    | GD            | 0.56           | 3.0        | 300       | 1                      |
| MOO-DNAS (s)    | GD            | 0.54           | 3.3        | 320       | 1                      |

TABLE 4. Results compared with current advanced neural networks on the CIFAR-10 dataset. The search cost can be shortened into 0.3 GPU-days on Tesla P100 and 0.2 GPU-days on Tesla V100.

| Architecture     | Search Method | Efficient-aware | Error Rate (%) | Params (M) | FLOPs (M) | Search Cost (GPU-days) |
|------------------|---------------|-----------------|----------------|------------|-----------|------------------------|
| VGG-16 [54]      | Manual        | X               | 6.4            | 15         | 313       | -                      |
| SBP-VGG-16 [42]  | MC            | ✓               | 7.5            | -          | 136       | -                      |
| l2-VGG-16 [43]   | MC            | ✓               | 6.6            | 5.4        | 206       | -                      |
| Ghost-VGG-16 [19]| Manual        | ✓               | 6.3            | 7.7        | 158       | -                      |
| ResNet-56 [1]    | Manual        | X               | 7.0            | 0.85       | 125       | -                      |
| AMC-ResNet-56 [51]| MC           | ✓              | 8.1            | -          | 63        | -                      |
| CP-ResNet-56 [42]| MC            | ✓               | 8.0            | -          | 63        | -                      |
| l1-ResNet-56 [43]| MC            | ✓               | 7.5            | 0.73       | 91        | -                      |
| Ghost-ResNet-56 [19]| Manual  | ✓            | 7.3            | 0.43       | 63        | -                      |
| MobileNets [15]  | Manual        | ✓               | 4.5            | 3.2        | -         | -                      |
| MobileNetv2 [16] | Manual        | ✓               | 3.8            | 3.2        | 300       | -                      |
| NASNet [29]      | RL            | X               | 4.47           | 7.1        | -         | 2000                   |
| NAO-WS [35]      | RL+EA         | X               | 3.53           | 3.1        | -         | 0.4                    |
| PNAS [36]        | SMBO          | X               | 3.41           | 3.2        | -         | 225                    |
| AmoebaNet [31]   | EA            | X               | 3.34           | 3.2        | -         | 3150                   |
| DARTS [58]       | GD            | X               | 3.00           | 3.3        | -         | 1.5                    |
| MOO/D-Net [25]   | EA            | ✓               | 5.88           | 8.1        | -         | 0.33                   |
| DPP-Net [21]     | EA            | ✓               | 4.78           | 1.0        | 137       | -                      |
| NSGAnet [23]     | EA            | ✓               | 3.85           | 3.3        | -         | 8                      |
| MnasNet [20]     | RL            | ✓               | 3.85           | 3.5        | -         | 1200                   |
| LEMONADE [22]    | EA            | ✓               | 3.05           | 4.7        | -         | 80                     |
| CARS [24]        | EA            | ✓               | 2.81           | 3.0        | -         | 0.4                    |
| MOO-DNAS (h)     | GD            | ✓               | 3.31           | 3.5        | 360       | 0.93                   |
| MOO-DNAS (s)     | GD            | ✓               | 3.24           | 4.1        | 420       | 0.93                   |

TABLE 5. Results compared with the SOTA neural networks on the CIFAR-100 dataset.

Starting from FBNet [26], we first employ the same layer-wise search space containing 7 stages with 22 layers which is designed for ImageNet [52], and the searched architectures are shown in Fig. 7(c), (d). For further verifying the effectiveness of our search space, we decrease the layer number to 10, and the searched architectures are shown in Fig. 7(e), (f).
The comparison between different models with different layer number is shown in Tab. 4. These results show that our proposed architectures generate a better light-weight model by trading accuracy with model size. When N = 22, the error rate is competitive with our architectures, while the model size is much larger, and there are a lot of skip-connections overall the model as shown in Fig. 7(c), (d). The primary reason of excessive skip connections is the inherent unfair competition and the search space is not fit to small CIFAR10 dataset. The skip connection forms a residual module, which is convenient for the supernet training. When N = 10, the model size is much smaller while the accuracy is about 2.5% lower, so the accuracy-efficiency trade-off achieves not well.

### TABLE 6. Comparison between different model size on CIFAR10.

| Layers       | Error Rate (%) | Params (M) | FLOPs (M) |
|--------------|----------------|------------|-----------|
| MOO-DNAS (h) | 3.31           | 3.5        | 360       |
| MOO-DNAS (s) | 3.24           | 4.1        | 420       |
| N=22 (hard-sampling) | 3.29   | 4.7        | 537       |
| N=22 (soft-sampling)   | 3.39   | 4.3        | 496       |
| N=10 (hard-sampling)  | 5.94   | 1.1        | 137       |
| N=10 (soft-sampling)  | 5.84   | 1.4        | 174       |

2) HARD-SAMPLING V.S. SOFT-SAMPLING

In this subsection, we demonstrate the effectiveness of two sampling techniques, hard-sampling and soft-sampling, and discuss the advantages and disadvantages of them. To do this, we execute ten trials with different random seeds on the convincing CIFAR10 dataset and average the results.

From the table 4, it can be observed that the best accuracy of architecture after hard-sampling is lower than soft-sampling, but the average accuracy after 10 times independent hard-sampling is higher. The process of our search algorithm is to train a supernet distribution \( P_\theta \), so the stronger the distribution, the better the sampled architecture. Due to the randomness of soft-sampling, we can obtain different network architectures during different soft-sampling. And the performance of soft-sampling architectures depends on the well-trained distribution, but the stability of these architectures cannot be ensured. However, the nature of hard-sampling is to remain the candidate block with the highest score. It fixes the network architecture and decreases the flexibility of architectures, but can ensure the consistence of different sampling. In this way, the stability of hard-sampling is better than soft-sampling while the flexibility is lower.

### E. DISCUSSIONS

In this research, we aim to design efficient neural networks automatically, therefore the major goal is to trade off two conflicting objectives: classification accuracy and network latency on the target device. To explore light-weight neural networks, we present an efficient CNN architecture search framework based on differentiable neural architecture search.

First, we discuss the performance of the searched model. For the objective of accuracy, MOO-DNAS-Nets are able to achieve comparable accuracy with the state-of-the-art network models. As shown in Tab. 4, the error rates are firstly 3.3% lower when compared with the manual-designed VGG-16 [54] as well as its variants via model compression. For the ResNet [1] which is already significantly smaller than VGG-16 [54], our models attain a relative accuracy improvement of 4.0%. Compared with manual efficient networks, the error rates are slightly 0.5% lower than MobileNets [15] and 1.2% lower than MobileNetV2 [16]. Apart from handcrafted models, MOO-DNAS-Nets surpass the NAS baselines from RL-based and EA-based approaches, such as NASNet [29], NAO [35], PNAS [36] and AmoebaNet [31], and outperform most works of multi-objective NAS. The error rates are 0.5% lower than NSGANet [23] and MnasNet [20], and about 1.2% lower than NASNet [29] and DPP-Net [21]. In terms of model efficiency, the parameters and FLOPs of MOO-DNAS-Nets are slightly greater than those of manually designed efficient
networks, but comparable to those of the efficient-aware NAS models. Actually, the micro topology of our hierarchical search space is based on MobileNetV2 building blocks, the parameters and FLOPs are greatly decreased as a result of various deformed convolution. Furthermore, the macro topology allows the stochastic supernet to modify the operation blocks of each layer for varied model sizes.

However, MOO-DNAS-Nets perform worse than many NAS baselines. DARTS [38] outperforms ours by 0.3% in accuracy, and we suspect that searching for cell-based topology could develop a better CNN than layer-wise search space. LEMONADE [22] and CARS [24] outperform ours by 0.2-0.5%. Instead of optimizing the device-related metric of model’s inference latency, they aim to optimize the device-agnostic one of model size. This demonstrates that it is argued that optimizing the real speed of models might sacrifice a little accuracy.

The second issue is the search cost of NAS approaches. MOO-DNAS requires less than one GPU-day to get the final networks, which is 1000x faster than traditional RL-based approaches such as NASNet [29] and MnasNet [20], 1.5x faster than DARTS [38], and 0.5x slower than CARS [24] and MOOSO/D [25]. We utilize architecture parameters to model the process of architecture search and relax the discrete search space into continuous. The gradient descent technique can be used to optimize architectures and corresponding weights. As a result, the search process is greatly accelerated.

These evidences suggest that the proposed method is effective in discovering high accuracy and light-weight CNN architectures compared with both manually-designed networks and automated NAS approaches. On the other hand, multi-objective NAS may sacrifice accuracy for greater efficiency. The prediction of model efficiency, such as latency, parameters or FLOPs is critical for the NAS approaches to select objectives, and hence influences the performance of multi-objective NAS.

However, there are some limitations of our method: (1) As shown in Tab. 3, the performance of searched models on the MNIST does not achieve the average accuracy. This is because that the search space is MobileNetV2-based, and the macro topology is pre-defined for the image size of $32 \times 32$. As a result, the generalization of the search space is not enough for different image resolutions, and we combine manually pre-processing to solve this issue. (2) Compared with directly calculating the model size, the approximate model of computational resources.

In the future, we hope to establish a more accurate model for measuring inference latency. Because of the inconsistency between the training and the quantization version of models, an accurate approximation of actual runtime can improve the quality of search algorithm. Furthermore, we want to implement our search framework on different search spaces, such as NASNet or DARTS-based, for better generalization, so that the method can be adopted on other datasets.

**ACKNOWLEDGMENT**

(Hui Wei and Feifei Lee contributed equally to this work.)

**REFERENCES**

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.

[2] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 7132–7141.

[3] L. Xie, F. Lee, L. Liu, K. Kotani, and Q. Chen, “Scene recognition: A comprehensive survey,” Pattern Recognit., vol. 102, Jun. 2020, Art. no. 107205.

[4] L. Xie, F. Lee, L. Liu, Z. Yin, and Q. Chen, “Hierarchical coding of convolutional features for scene recognition,” IEEE Trans. Multimedia, vol. 22, no. 5, pp. 1182–1192, May 2020.

[5] S. Huang, F. Lee, R. Miao, Q. Li, C. Lu, and Q. Chen, “A deep convolutional neural network architecture for interstitial lung disease pattern classification,” Med. Biol. Eng. Comput., vol. 58, no. 4, pp. 725–737, Jan. 2020.

[6] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as points,” 2019, arXiv:1904.07830.

[7] Z. Tian, C. Shen, H. Chen, and T. He, “FCOS: Fully convolutional one-stage object detection,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 9627–9636.

[8] J. Cai, F. Lee, S. Yang, C. Lin, H. Chen, K. Kotani, and Q. Chen, “Pedestrian as points: An improved anchor-free method for center-based pedestrian detection,” IEEE Access, vol. 8, pp. 179666–179677, 2020.

[9] L.-C. Chen and Y. Zhu, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 801–818.

[10] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in Proc. Int. Conf. Learn. Represent. (ICLR), May 2016, pp. 1–14.

[11] J.-H. Luo, J. Wu, and W. Lin, “ThiNet: A filter level pruning method for deep neural network compression,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 5058–5066.
B. Baker, O. Gupta, and N. Naik, "Designing neural network architectures," in Proc. ACM Conf. Comput. Commun. Secur. (CCS), Nov. 2014, pp. 525–542.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2704–2713.

G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, arXiv:1503.02531.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, arXiv:1704.04861.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 4510–4520.

X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 6848–6856.

N. Ma and X. Zhang, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 122–138.

K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 1577–1586.

M. Tan, B. Chen, and R. Pang, "MnasNet: Platform-aware neural architecture search for mobile," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 2820–2828.

J.-D. Dong and A.-C. Cheng, "DPP-Net: Device-aware progressive search for pareto-optimial neural architectures," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 517–531.

T. Elsken, F. Hutter, and J. H. Metzen, "Efficient multi-objective neural architecture search via Lamarckian evolution," in Proc. Int. Conf. Learn. Represent. (ICLR), 2019, pp. 1–23.

Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-net: Neural architecture search using multi-objective genetic algorithm," 2018, arXiv:1810.03522.

Z. Yang, Y. Wang, X. Chen, B. Shi, C. Xu, Q. Tian, and C. Xu, "CARS: Continuous evolution for efficient neural architecture search," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 1829–1839.

J. Jiang, F. Han, Q. Ling, J. Wang, T. Li, and H. Han, "Efficient network architecture search via multiobjective particle swarm optimization based on decomposition," Neural Netw., vol. 123, pp. 305–316, Mar. 2020.

B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 10734–10742.

B. Baker, O. Gupta, and N. Ngaik, "Designing neural network architectures using reinforcement learning," in Proc. Int. Conf. Learn. Represent. (ICLR), Apr. 2017, pp. 1–18.

B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in Proc. Int. Conf. Learn. Represent. (ICLR), Apr. 2017, pp. 1–16.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 8697–8710.

Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2423–2432.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in Proc. Conf. Artif. Intell. (AAAI), 2019, pp. 4780–4789.

E. Real, S. Moore, A. Selle, and S. Saxena, "Large-scale evolution of image classifiers," in Proc. Int. Conf. Mach. Learn. (ICML), 2017, pp. 4429–4446.

H. Liu, K. Simonyan, and O. Vinyals, "Hierarchical representations for efficient neural architecture search," in Proc. Int. Conf. Learn. Represent. (ICLR), 2018, pp. 1–13.

K. Kandasamy, W. Neiswanger, and J. Schneider, "Neural architecture search with Bayesian optimisation and optimal transport," in Proc. Conf. Neural Inf. Process. Syst. (NeurIPS), Dec. 2018, pp. 2016–2025.

R. Luo, F. Tian, and T. Qin, "Neural architecture optimization," in Proc. Conf. Neural Inf. Process. Syst. (NeurIPS), Dec. 2018, pp. 7816–7827.

C. Liu, B. Zoph, and M. Neumann, "Progressive neural architecture search," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 19–35.

R. Negrinho and G. Gordon, "DeepArchitect: Automatically designing and training deep architectures," 2017, arXiv:1704.08792.

H. Liu, K. Simonyan, and Y. Yang, "DArts: Differentiable architecture search," in Proc. Int. Conf. Learn. Represent. (ICLR), May 2019, pp. 1–13.

S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in Proc. Int. Conf. Learn. Represent. (ICLR), May 2019, pp. 1–17.

H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in Proc. Int. Conf. Learn. Represent. (ICLR), May 2019, pp. 1–13.

C. J. Maddision, A. Mnih, and Y. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in Proc. Int. Conf. Learn. Represent. (ICLR), Apr. 2017, pp. 1–20.

Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 1389–1397.

H. Li, H. Samet, and A. Kadav, "Pruning filters for efficient convnets," in Proc. Int. Conf. Learn. Represent. (ICLR), Apr. 2017, pp. 1–13.

E. L. Denton and W. Zaremba, "Exploiting linear structure within convolutional networks for efficient evaluation," in Proc. Conf. Neural Inf. Process. Syst. (NeurIPS), Jan. 2014, pp. 1269–1277.

F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 1251–1258.

X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 1761–1770.

H. Pham, M. Y. Guan, and B. Zoph, "Efficient neural architecture search via parameters sharing," in Proc. Int. Conf. Mach. Learn. (ICML), Jul. 2018, pp. 6522–6531.

Z. Guo, X. Zhang, and H. Mu, "Single path one-shot neural architecture search with uniform sampling," in Proc. Eur. Conf. Comput. Vis. (ECCV), Aug. 2020, pp. 544–560.

A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 1314–1324.

T. J. Yang, A. Howard, and B. Chen, "NetAdapt: Platform-aware neural network adaptation for mobile applications," in Proc. Eur. Conf. Comput. Vis. (ECCV), Sep. 2018, pp. 289–304.

Y. He, J. Lin, and Z. Liu, "AMC: AutoML for model compression and acceleration on mobile devices," in Proc. Eur. Conf. Comput. Vis. (ECCV), Sep. 2018, pp. 815–832.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2009, pp. 1449–1456.