Multiobjective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification

Zhichao Lu, *Student Member, IEEE*, Ian Whalen, Yashesh Dhebar, Kalyanmoy Deb, *Fellow, IEEE*, Erik D. Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti, *Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) are the backbones of deep learning paradigms for numerous vision tasks. Early advancements in CNN architectures are primarily driven by human expertise and by elaborate design processes. Recently, neural architecture search was proposed with the aim of automating the network design process and generating task-dependent architectures. While existing approaches have achieved competitive performance in image classification, they are not well suited to problems where the computational budget is limited for two reasons: 1) the obtained architectures are either solely optimized for classification performance, or only for one deployment scenario and 2) the search process requires vast computational resources in most approaches. To overcome these limitations, we propose an evolutionary algorithm for searching neural architectures under multiple objectives, such as classification performance and floating point operations (FLOPs). The proposed method addresses the first shortcoming by populating a set of architectures to approximate the entire Pareto frontier through genetic operations that recombine and modify architectural components progressively. Our approach improves computational efficiency by carefully down-scaling the architectures during the search as well as reinforcing the patterns commonly shared among past successful architectures through Bayesian model learning. The integration of these two main contributions allows an efficient design of architectures that are competitive and in most cases outperform both manually and automatically designed architectures on benchmark image classification datasets: CIFAR, ImageNet, and human chest X-ray. The flexibility provided from simultaneously obtaining multiple architecture choices for different compute requirements further differentiates our approach from other methods in the literature.**

***Index Terms*—Convolutional neural networks (CNNs), evolutionary deep learning, genetic algorithms (GAs), neural architecture search (NAS).**

I. Introduction

DEEP convolutional neural networks (CNNs) have been overwhelmingly successful in a variety of computer-vision-related tasks like object classification, detection, and segmentation. One of the main driving forces behind this success is the introduction of many CNN architectures, including GoogLeNet [1], ResNet [2], DenseNet [3], etc., in the context of object classification. Concurrently, architecture designs, such as ShuffleNet [4], MobileNet [5], LBCNN [6], etc., have been developed with the goal of enabling real-world deployment of high-performance models on resource-constrained devices. These developments are the fruits of years of painstaking efforts and human ingenuity.

Neural architecture search (NAS), on the other hand, presents a promising path to alleviate this painful process by posing the design of CNN architectures as an optimization problem. By altering the architectural components in an algorithmic fashion, novel CNNs can be discovered that exhibit improved performance metrics on representative datasets. The huge surge in research and applications of NAS indicates the tremendous academic and industrial interest NAS has attracted, as teams seek to stake out some of this territory. It is now well recognized that designing bespoke neural network architectures for various tasks is one of the most challenging and practically beneficial components of the entire deep neural network (DNN) development process, and is a fundamental step toward automated machine learning (ML).

Early methods for NAS relied on reinforcement learning (RL) to navigate and search for architectures with high performance. A major limitation of these approaches [7], [8] is the steep computational requirement for the search process itself, often requiring weeks of wall clock time on hundreds of graphics processing unit (GPU) cards. Recent *relaxation*-based methods [9]–[12] seek to improve the computational efficiency of NAS approaches by approximating the connectivity between different layers in the CNN architectures by real-valued variables that are learned (optimized) through gradient descent together with the weights. However, such relaxation-based NAS methods suffer from excessive GPU memory requirements during search, resulting in constraints on the size of the search space (e.g., reduced layer operation choices).

Manuscript received November 29, 2019; revised May 8, 2020 and July 21, 2020; accepted September 8, 2020. Date of publication September 21, 2020; date of current version March 31, 2021. This work was supported by the National Science Foundation under Cooperative Agreement under Grant DBI-0939454. *(Corresponding author: Zhichao Lu.)*

The authors are with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: lu.zhichao@outlook.com).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2020.3024708.

Digital Object Identifier 10.1109/TEVC.2020.3024708

In addition to being accurate in prediction, real-world applications demand that NAS methods find network architectures that are also efficient in computation—e.g., have low power consumption in mobile applications and low latency in autonomous driving applications. It has been a common observation that the predictive performance continuously improves as the complexity (i.e., # of layers, channels, etc.) of the network architectures increases [2], [3], [8], [13]. This alludes to the competing nature of trying to simultaneously maximize predictive performance and minimize network complexity, thereby necessitating multiobjective optimization. Despite recent advances in RL and relaxation-based NAS methods, they are still not readily applicable for multiobjective NAS.

Among the many different NAS methods being continually proposed, evolutionary algorithms (EAs) are getting a plethora of attention, due to their population-based nature and flexibility in encoding. They offer a viable alternative to conventional ML-oriented approaches, especially under the scope of multiobjective NAS. An EA, in general, is an iterative process in which individuals in a population are made gradually better by applying variations to selected individuals and/or recombining parts of multiple individuals. Despite the ease of extending them to handle multiple objectives, most existing EA-based NAS methods [14]–[19] are still single-objective driven.

In this article, we present NSGANetV1, a multiobjective EA for NAS, extending on an earlier proof-of-principle method [20], to address the aforementioned limitations of current approaches. The key contributions followed by the extensions made in this article are summarized below.

1) NSGANetV1 populates a set of architectures to approximate the entire Pareto front in one run through customized genetic operations that recombine and modify architectural components progressively. NSGANetV1 improves computational efficiency by carefully downscaling the architectures during the search as well as reinforcing the emerging patterns shared among past successful architectures through a Bayesian network (BN)-based distribution estimation operator. Empirically, the obtained architectures, in most cases, outperform both manually and other automatically designed architectures on various datasets.
2) By obtaining a set of architectures in one run, NSGANetV1 allows designers to choose a suitable network *a-posteriori* as opposed to a predefined preference weighting of objectives prior to the search. Further post-optimal analysis of the set of nondominated architectures often reveals valuable design principles, which is another benefit of posing NAS as a multiobjective optimization problem, as is done in NSGANetV1.
3) From an algorithmic perspective, we extend our previous work [20] in a number of ways: a) an expanded search space to include five more layer operations and one more option that controls the width of the network; b) improved encoding, mutation, and crossover operators accompanying the modified search space; and c) a more thorough lower-level optimization process for weight learning, resulting in better and more reliable performance.
4) From an evaluation perspective, we extend our previous work [20] in two different ways: a) adding three more tasks, including medical imaging, robustness to adversarial attacks, and car key-point estimation and b) evaluating the searched architectures on five new datasets, including, ImageNet, ImageNet-V2, CIFAR-10.1, corrupted CIFAR-10, and corrupted CIFAR-100.

The remainder of this article is organized as follows. Section II introduces and summarizes related literature. In Section III, we provide a detailed description of the main components of our approach. We describe the experimental setup to validate our approach along with a discussion of the results in Section IV, followed by further analysis and an application study in Sections V and VI, respectively. Finally, we conclude with a summary of our findings and comment on possible future directions in Section VII.

II. Related Work

Recent years have witnessed growing interest in NAS. The promise of being able to automatically and efficiently search for task-dependent network architectures is particularly appealing as DNNs are widely deployed in diverse applications and computational environments. Early methods [21], [22] made efforts to simultaneously evolve the topology of neural networks along with weights and hyperparameters. These methods perform competitively with hand-crafted networks on control tasks with shallow fully connected networks. In the following, we present studies related to deep CNNs for image classification. Readers are referred to the supplementary materials for a more detailed review of the topic.

Evolutionary NAS: Designing neural networks through evolution has been a topic of interest for a long time. Recent evolutionary approaches focus on evolving solely the topology while leaving the learning of weights to gradient descent algorithms, and using hyper-parameter settings that are manually tuned. Xie and Yuille's work of Genetic CNN [14] is one of the early studies that shows the promise of using EAs for NAS. Real *et al.* [15] introduced perhaps the first truly large scale application of a simple EA to NAS. The extension of this method presented in [17], called AmoebaNet, provides the first large scale comparison of EA and RL methods. Their EA, using an age-based selection similar to [23], has demonstrated faster convergence to an accurate network when compared to RL and random search. Concurrently, Liu *et al.* [16] evolved a hierarchical representation that allows nonmodular layer structures to emerge. Despite the impressive improvements achieved on various datasets, these EA methods are extremely computationally inefficient, e.g., one run of the regularized evolution method [17] takes one week on 450 GPU cards.

Concurrently, another streamlining of EA methods for use in budgeted NAS has emerged. Suganuma *et al.* [24] used Cartesian genetic programming to assemble an architecture from existing modular blocks (e.g., Residual blocks). Sun *et al.* [19] used a random forest as an offline surrogate model to predict the performance of architectures, partially

eliminating the lower-level optimization via gradient descent. The reported results yield 3× savings in wall clock time with similar classification performance when compared to their previous works [18], [25]. However, results reported from these budgeted EA methods are far from state-of-the-art and only demonstrated on small-scale datasets—i.e., CIFAR-10 and CIFAR-100.

Multiobjective NAS: In this work, the term *multiobjective NAS* refers to methods that simultaneously approximate the entire set of efficient tradeoff architectures in one run [26], [27]. Kim *et al.* [28] presented NEMO, one of the earliest evolutionary multiobjective approaches to evolve CNN architectures. NEMO uses NSGA-II [29] to maximize classification performance and inference time of a network and searches over the space of the number of output channels from each layer within a restricted space of seven different architectures. DPP-Net [30], an extension from [31], progressively expands networks from simple structures and only trains the top-K (based on Pareto-optimality) networks that are predicted to be promising by a surrogate model. Elsken *et al.* [32] presented the LEMONADE method, which is formulated to develop networks with high predictive performance and lower resource constraints. LEMONADE reduces compute requirements through custom-designed approximate network morphisms [33], which allow newly generated networks to share parameters with their forerunners, obviating the need to train new networks from scratch. However, LEMONADE still requires nearly 100 GPU-days to search on the CIFAR datasets [34].

Search Efficiency: The main computation bottleneck of NAS resides in the lower-level optimization of learning the weights for evaluating the performance of architectures. One such evaluation typically requires hours to finish. To improve the practical utility of search under a constrained computational budget, NAS methods commonly advocate for substitute measurements without a full-blown lower-level optimization. A widely used approach proceeds as follows: it reduces the depth (number of layers) and the width (number of channels) of the intended architecture to create a small-scale network—i.e., a *proxy model*. Proxy models require an order of magnitude less computation time (typically, minutes) to perform lower-level optimization, and the performance of proxy models is then used as surrogate measurements to guide the search. However, most existing NAS work [8], [16], [17], [31], [35] follows simple heuristics to construct the proxy model, resulting in low correlation in prediction. For instance, NASNet [8] has an additional reranking stage that trains the top 250 architectures for 300 epochs each (takes more than a year on a single GPU card) before picking the best one, and the reported NASNet-A model was originally ranked 70th among the top 250 according to the performance measured at proxy model scale. Similarly, AmoebaNet [17] relies on the evaluation of duplicate architectures to gauge representative performance, leading to 27K models being evaluated during search.

In this work, we focus on both the efficiency and the reliability aspects of the proxy model; through a series of systematic studies in a controlled setting, we empirically establish the tradeoff between the correlation of proxy performance to true performance and the speed-up in estimation. We then implement a suitable setting that is specific to our search space and dataset.

Algorithm 1: General Framework of NSGANetV1

Input : Complexity objective $\tilde{f}$ (see Eq. (1)), Max. number of generations G, Population size K, Crossover probability p_c, Mutation probability p_m, The starting generation of exploitation τ.

1 $g \leftarrow 0$ // initialize a generation counter.
2 $\rho \leftarrow 1$ // initialize the control parameter for *exploration*.
3 $\mathcal{A} \leftarrow$ initialize an empty archive to keep track of evaluated archs.
4 $P \leftarrow$ initialize the parent population by uniform sampling.
5 // compute accuracy through lower-level optimization in Algo. 2.
6 $f \leftarrow Evaluate(P)$
7 // calculate domination rank and crowding distance.
8 $[F_1, F_2, \dots] \leftarrow NondominatedSort(f, \tilde{f}(P))$
9 $dist \leftarrow CrowdingDistance(F_1, F_2, \dots)$
10 **while** $g < G$ **do**
11 $k \leftarrow 0$ // initialize an individual counter.
12 $Q \leftarrow \emptyset$ // offspring population.
13 **while** $k < K$ **do**
14 // one offspring is created in each iteration k.
15 **if** $rand() < \rho$ **then**
16 // choose two parents for mating.
17 $p \leftarrow BinaryTournamentSelection(P, [F_1, F_2, \dots], dist)$
18 $q \leftarrow Crossover(p, p_c)$
19 $q \leftarrow Mutation(q, p_m)$
20 **else**
21 // estimate the distribution of the Pareto set.
22 $BN \leftarrow$ construct a Bayesian Network from $\mathcal{A}$.
23 $q \leftarrow$ sample an offspring from BN.
24 **end**
25 $Q \leftarrow Q \cup q$; $k \leftarrow k + 1$
26 **end**
27 $f' \leftarrow Evaluate(Q)$ // see line 5.
28 $[F_1, F_2, \dots] \leftarrow NondominatedSort(f \cup f', \tilde{f}(P) \cup \tilde{f}(Q))$
29 $dist \leftarrow CrowdingDistance(F_1, F_2, \dots)$
30 // survive the top-K archs to next generation following the environmental selection procedures outlined in [29].
31 $P \leftarrow Selection(P \cup Q, [F_1, F_2, \dots], dist, K)$
32 $g \leftarrow g + 1$; $\mathcal{A} \leftarrow \mathcal{A} \cup Q$
33 **if** $g = \tau$ **then**
34 $\rho \leftarrow 0.75$ // assign 25% of the offspring to be created by BN.
35 **else if** $g > \tau$ **then**
36 update ρ according to Eq. (2).
37 **else**
38 $\rho \leftarrow 1$ // remain unchanged from initial value.
39 **end**
40 **end**
41 **Return** parent population P.

III. Proposed Approach

Practical applications of NAS can rarely be considered from the point of view of a single objective of maximizing performance; rather, they must be evaluated from at least one additional, conflicting objective that is specific to the deployment scenario. In this work, we approach the problem of designing high-performance architectures with diverse complexities for different deployment scenarios as a multiobjective bilevel optimization problem[1] [36]. We mathematically formulate the problem as

$$\begin{aligned} &\text{minimize} \;\; \boldsymbol{F}(\boldsymbol{x}) = \big(f_1\big(\boldsymbol{x}; \boldsymbol{w}^*(\boldsymbol{x})\big), f_2(\boldsymbol{x})\big)^T \\ &\text{subject to} \;\; \boldsymbol{w}^*(\boldsymbol{x}) \in \operatorname{argmin} \;\; \mathcal{L}(\boldsymbol{w}; \boldsymbol{x}) \end{aligned}$$

[1]See the supplement, Section III, for more about bilevel optimization.

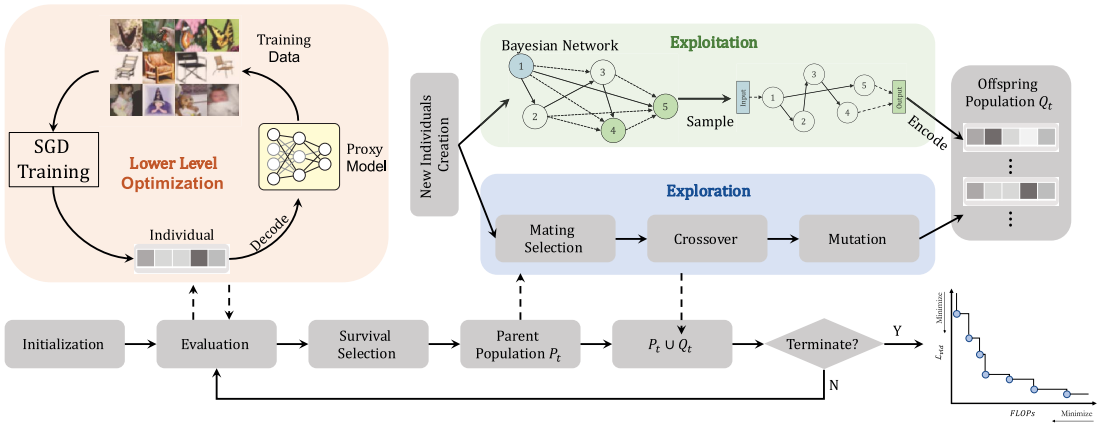


Fig. 1. *Overview:* Given a dataset and objectives, NSGANetV1 designs a set of custom architectures spanning the trading-off front. NSGANetV1 estimate the performance of an architecture through its *proxy model*, optimized by SGD in the lower-level. The search proceeds in *exploration* via genetic operations, followed by *exploitation* via distribution estimation. See Algorithm 1 for pseudocode and colors are in correspondence.

$$\boldsymbol{x} \in \Omega_x, \boldsymbol{w} \in \Omega_w \tag{1}$$

here $\boldsymbol{\Omega}_x = \Pi_{i=1}^{n}[a_i, b_i] \subseteq \mathbb{Z}^n$ is the architecture decision space, where a_i, b_i are the lower and upper bounds, $\boldsymbol{x} = (x_1, \ldots, x_n)^T \in \boldsymbol{\Omega}_x$ is a candidate architecture, and the lower-level variable $\boldsymbol{w} \in \boldsymbol{\Omega}_w$ denotes its associated weights. The upper-level objective vector $\boldsymbol{F}$ comprises of the classification error (f_1) on the validation data $\mathcal{D}_{\text{vld}}$, and the complexity (f_2) of the network architecture. The lower level objective $\mathcal{L}(\boldsymbol{w}; \boldsymbol{x})$ is the cross-entropy loss on the training data $\mathcal{D}_{\text{trn}}$.

Our proposed algorithm, NSGANetV1, is an iterative process in which initial architectures are made gradually better as a group, called a *population*. In every iteration, a group of *offspring* (i.e., new architectures) is created by applying variations through crossover and mutation to the more promising of the architectures already found, also known as *parents*, from the population. Every member in the population (including both parents and offspring) compete for survival and reproduction (becoming a parent) in each iteration. The initial population may be generated randomly or guided by prior-knowledge, i.e., seeding the past successful architectures directly into the initial population. Subsequent to initialization, NSGANetV1 conducts the search in two sequential stages: 1) *exploration*, with the goal of discovering diverse ways to construct architectures and 2) *exploitation* that reinforces the emerging patterns commonly shared among the architectures successful during exploration. A set of architectures representing efficient tradeoffs between network performance and complexity is obtained at the end of evolution, through genetic operators and a Bayesian-model-based learning procedure. A flowchart and a pseudocode outlining the overall approach are shown in Fig. 1 and Algorithm 1, respectively. In the remainder of this section, we provide a detailed description of the aforementioned components in Sections III-A–III-C.

A. Search Space and Encoding

The search for optimal network architectures can be performed over many different search spaces. The generality of the chosen search space has a major influence on the quality of results that are even possible. Most existing evolutionary NAS approaches [14], [19], [24], [32] search only one aspect of the architecture space—e.g., the connections and/or hyper-parameters. In contrast, NSGANetV1 searches over both operations and connections—the search space is thus more comprehensive, including most of the previous successful architectures designed both by human experts and algorithmically.

Modern CNN architectures are often composed of an outer structure (network-level) design where the width (i.e., number of channels), the depth (i.e., number of layers) and the spatial resolution changes (i.e., locations of pooling layers) are decided; and an inner structure (block-level) design where the layer-wise connections and computations are specified, e.g., Inception block [1], ResNet block [2], DenseNet block [3], etc. As seen in the CNN literature, the network-level decisions are mostly hand-tuned based on meta-heuristics from prior knowledge and the task at hand, as is the case in this work. For block-level design, we adopt the one used in [8], [9], [17], and [31] to be consistent with previous work.

A *block* is a small convolutional module, typically repeated multiple times to form the entire neural network. To construct scalable architectures for images of different resolutions, we use two types of blocks to process intermediate information: 1) the *Normal* block, a block type that returns information of the same spatial resolution and 2) the *Reduction* block, another block type that returns information with spatial resolution halved by using a stride of two. See Fig. 2(a) for a pictorial illustration.

We use directed acyclic graphs (DAGs) consisting of five nodes to construct both types of blocks (a Reduction block uses a stride of two). Each *node* is a two-branched structure, mapping two inputs to one output. For each node in block i, we need to pick two inputs from among the output of the previous block h_{i-1}, the output of the previous-previous block h_{i-2}, and the set of hidden states created in any previous nodes of block i. For pairs of inputs chosen, we choose a computation operation from among the following options, collected based on their prevalence in the CNN literature.

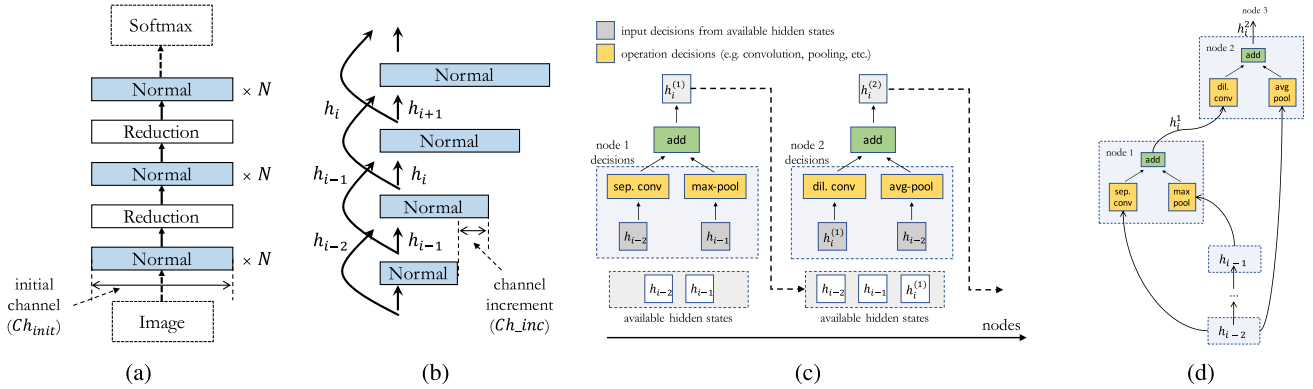


Fig. 2. Schematic of the NSGANetV1 search space motivated from [8]. (a) Architecture is composed of stacked blocks. (b) Number of channels in each block is gradually increased with the depth of the network. (c) Each block is composed of five nodes, where each node is a two-branched computation applied to outputs from either previous blocks or previous nodes within the same block. (d) Graphical visualization of (c).

1) identity;
2) 3×3 max pooling;
3) 3×3 average pooling;
4) squeeze-and-excitation [37];
5) 3×3 local binary conv [6];
6) 5×5 local binary conv [6];
7) 3×3 dilated convolution;
8) 5×5 dilated convolution;
9) 3×3 depthwise-separable conv;
10) 5×5 depthwise-separable conv;
11) 7×7 depthwise-separable conv;
12) 1×7 then 7×1 convolution.

The results computed from both branches are then added together to create a new hidden state, which is available for subsequent nodes in the same block. See Fig. 2(b)–(d) for pictorial illustrations. The search space we consider in this article is an expanded version of the *micro search space* used in our previous work [20]. Specifically, the current search space: 1) search for a factor that gradually increments the channel size of each block with depth [see Fig. 2(b)] as opposed to sharply doubling the channel size when down-sampling and 2) considers an expanded set of primitive operations to include both more recent advanced layer primitives, such as squeeze-and-excitation [37] and more parameter-efficient layer primitives like local binary convolution [6].

With the above-mentioned search space, there are in total 20 decisions to constitute a block structure—i.e., choose two pairs of input and operation for each node, and repeat for five nodes. The resulting number of combinations for a block structure is

$$\mathcal{B} = ((n+1)!)^2 \cdot (n\_ops)^{2n}$$

where n denotes the number of nodes, n_ops denotes the number of considered operations. Therefore, with one Normal block and one Reduction block with five nodes in each, the overall size of the encoded search space is approximately 10^{33}.

B. Performance Estimation Strategy

To guide NSGANetV1 toward finding more accurate and efficient architectures, we consider two metrics as objectives, namely, classification accuracy and architecture complexity.

Algorithm 2: Performance Evaluation of a CNN

Input : The architecture α, training data $\mathcal{D}_{trn}$, validation data $\mathcal{D}_{vld}$, number of epochs T, weight decay λ, initial learning rate η_{max}.

$\omega \leftarrow$ Randomly initialize the weights in α;
$t \leftarrow 0$;
while $t < T$ **do**
  $\eta \leftarrow \frac{1}{2}\eta_{max}\left(1 + \cos\left(\frac{t}{T}\pi\right)\right)$;
  for *each data-batch in* $\mathcal{D}_{trn}$ **do**
    $\mathcal{L} \leftarrow$ Cross-entropy loss on the *data-batch*;
    $\nabla\omega \leftarrow$ Compute the gradient by $\partial\mathcal{L}/\partial\omega$;
    $\omega \leftarrow (1-\lambda)\omega - \eta\nabla\omega$;
  end
  $t \leftarrow t + 1$;
end
$acc \leftarrow$ Compute accuracy of $\alpha(\omega)$ on $\mathcal{D}_{vld}$;
Return the classification accuracy acc.

Assessing the classification accuracy of an architecture during search requires another optimization to first identify the optimal values of the associated weights via stochastic gradient descent (SGD; Algorithm 2). Even though there exist well-established gradient descent algorithms to efficiently solve this optimization, repeatedly executing such an algorithm for every candidate architecture renders the overall process computationally very prohibitive. Therefore, to overcome this computational bottleneck, we carefully (using a series of ablation studies) down-scale the architectures to create their proxy models [8], [17], which can be optimized efficiently in the lower-level through SGD. Their performances become surrogate measurements to select architectures during search. Details are provided in Section V-C.

A number of metrics can serve as proxies for complexity, including: the number of active nodes, number of active connections between the nodes, number of parameters, inference time, and number of floating-point operations (FLOPs) needed to execute the forward pass of a given architecture. Our initial experiments considered each of these metrics in turn. We concluded from extensive experimentation that inference time cannot be estimated reliably due to differences and inconsistencies in the computing environment, GPU manufacturer, ambient temperature, etc. Similarly, the number of parameters, active connections, or active nodes only relate to one aspect of

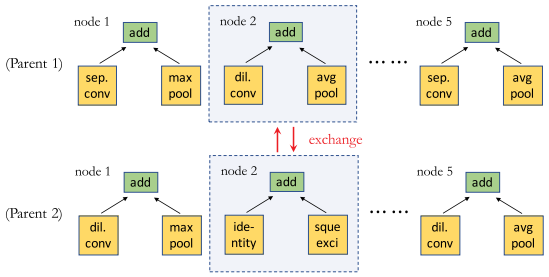


Fig. 3. Illustration of node-level crossover.

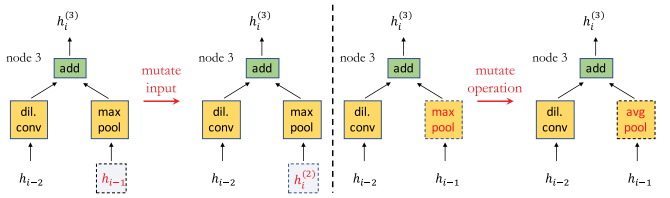


Fig. 4. Input and operation mutation: dashed line boxes with red color highlight the mutation. h_{i-2} and h_{i-1} are outputs from previous-previous and previous blocks, respectively. $h_i^{(3)}$ indicates output from node 3 of the current block.

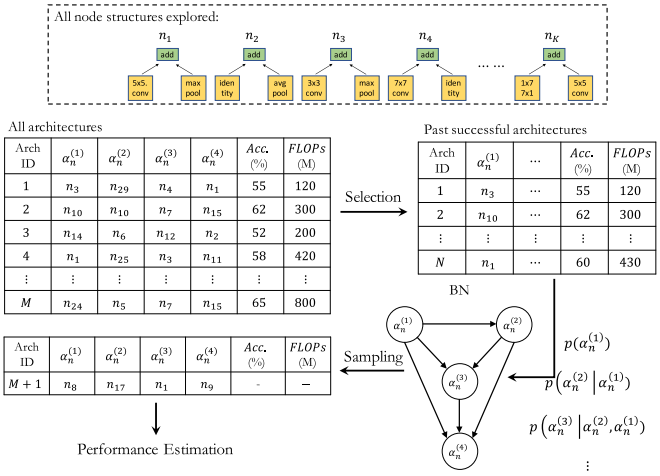


Fig. 5. Illustrative example of BN-based exploitation step in NSGANetV1: given past successful architectures, we construct a BN relating the dependencies between the four nodes inside the Normal block. A new architecture is then sampled from this BN and proceeds forward for performance estimation.

the complexity. In contrast, we found an estimate of FLOPs to be a more accurate and reliable proxy for network complexity. Therefore, classification accuracy and FLOPs serve as our choice of twin objectives to be traded off for selecting architectures. To simultaneously compare and select architectures based on these two objectives, we use the nondominated ranking and the "crowded-ness" concepts proposed in [29].

C. Creation of New Generation

Exploration: Given a population of architectures, parents are selected from the population with a fitness bias. This choice is dictated by two observations: 1) offspring created around better parents are expected to have higher fitness on average than those created around worse parents, with the assumption of some level of gradualism in the solution space and 2) occasionally (although not usually), offspring perform better than their parents, through inheriting useful traits from both parents. Because of this, one might demand that the best architecture in the population should always be chosen as one of the parents. However, the deterministic and greedy nature of that approach would likely lead to premature convergence due to the loss of diversity in the population [38]. To address this problem, we use binary tournament selection [39] to promote parent architectures in a stochastic fashion. At each iteration, binary tournament selection randomly picks two architectures from the population, then the one favored by the multiobjective selection criterion described in Section III-B becomes one of the parents. This process is repeated to select a second parent architecture; the two parent architectures then undergo a crossover operation.

In NSGANetV1, we use two types of crossover (with equal probability of being chosen) to efficiently exchange substructures between two parent architectures. The first type is at the block level, in which the offspring architectures are created by recombining the Normal block from the first parent with the Reduction block from the other parent and vice versa. The second type is at the node level, where a node from one parent is randomly chosen and exchanged with another node at the same position from the other parent. We apply the node-level crossover to both Normal and Reduction blocks. Fig. 3 illustrates an example of node-level crossover. Note that two offspring architectures are generated after each crossover operation, and only one of them (randomly chosen) is added to the offspring population. In each generation, an offspring population of the same size as the parent population is generated.

To enhance the diversity of the population and the ability to escape from local attractors, we use a discretized version of the polynomial mutation (PM) operator [40] subsequent to crossover. We allow mutation to be applied on both the input hidden states and the choice of operations. Fig. 4 shows an example of each type of mutation using the parent-centric PM operator, in which the offspring are intentionally created around the parents in the decision space. In association with PM, we sort our discrete encoding of input hidden states chronologically and choice of operations in ascending order of computational complexity. In the context of neural architecture, this step results in the mutated input hidden states in offspring architectures to more likely be close to the input hidden states in parent architectures in a chronological manner. For example, $h_i^{(2)}$ is more likely to be mutated to $h_i^{(1)}$ than to h_{i-2} by PM. A similar logic is applied in the case of mutation on layer operations.

Exploitation: After a sufficient number of architectures has been explored (consuming 2/3 of the total computational budget; i.e., τ in Algorithm 1), we start to enhance the exploitation aspect of the search. The key idea is to reinforce and reuse the patterns commonly shared among past successful architectures. We use the BN [41] as the probabilistic model to estimate the distribution of the Pareto set (of architectures).

In the context of our search space and encoding, this translates to learning the correlations among the operations and

connections of nodes within a block. Our exploitation step uses a subset (top-100 architectures selected based on domination rank and crowding distance [29]) of the past evaluated architectures to guide the final part of the search. More specifically, say we are designing a Normal block with three nodes, namely $\alpha_n^{(1)}$, $\alpha_n^{(2)}$, and $\alpha_n^{(3)}$. We would like to know the relationship among these three nodes. For this purpose, we construct a BN relating these variables, modeling the probability of Normal blocks beginning with a particular node $\alpha_n^{(1)}$, the probability that $\alpha_n^{(2)}$ follows $\alpha_n^{(1)}$, and the probability that $\alpha_n^{(3)}$ follows $\alpha_n^{(2)}$ and $\alpha_n^{(1)}$. In other words, we estimate the conditional distributions $p(\alpha_n^{(1)})$, $p(\alpha_n^{(2)}|\alpha_n^{(1)})$, and $p(\alpha_n^{(3)}|\alpha_n^{(2)}, \alpha_n^{(1)})$ by using the population history, and update these estimates during the exploitation process. New offspring architectures are created by sampling from this BN. A pictorial illustration of this process is provided in Fig. 5. This BN-based exploitation strategy is used in addition to the genetic operators, where we initially (i.e., at the beginning of exploitation) assign 25% of the offspring (line 34 in Algorithm 1) to be created by BN and we update this probability adaptively (line 36 in Algorithm 1). To be more specific, we calculate the probabilities of using genetic operators and sampling from the BN model at generation t based on the survival rates of offspring created using them in the previous generation, following the softmax function:

$$\rho_t^{(i)} = \frac{\exp\left(s_{t-1}^{(i)}\right)}{\sum_{i=1}^{2} \exp\left(s_{t-1}^{(i)}\right)} \tag{2}$$

where $\rho_t^{(i)}$ are the probabilities of using genetic operators ($i=1$) and sampling from the learned BN model ($i=2$); and $s_{t-1}^{(i)}$ are the survival rates of the offspring created by genetic operators ($i=1$) and the learned BN model ($i=2$) at the previous generation $t-1$. Note that $\rho_t^{(i=1)}$ corresponds to ρ in Algorithm 1.

IV. Experimental Setup and Results

In this section, we will evaluate the efficacy of NSGANetV1 on multiple benchmark image classification datasets.

A. Baselines

To demonstrate the effectiveness of the proposed algorithm, we compare the nondominated architectures achieved at the conclusion of NSGANetV1's evolution with architectures reported by various peer methods published in top-tier venues. The chosen peer methods can be broadly categorized into three groups: 1) architectures manually designed by human experts; 2) non-EA- (mainly RL or relaxation)-based; and 3) EA-based. Human engineered architectures include ResNet [2], ResNeXt [42], DenseNet [3], etc. The second and third groups range from earlier methods [7], [14], [15] that are oriented toward "proof-of-concept" for NAS, to more recent methods [8], [9], [17], [43], many of which improve state-of-the-art results on various computer vision benchmarks at the time they were published. The effectiveness of the different architectures is judged on both classification accuracy and computational complexity. For comparison on classification accuracy, three widely used natural object classification benchmark datasets are considered, namely, CIFAR-10, CIFAR-100, and ImageNet. More details and a gallery of examples from these three datasets are provided in Fig. 2 in supplementary materials under Section V.

B. Implementation Details

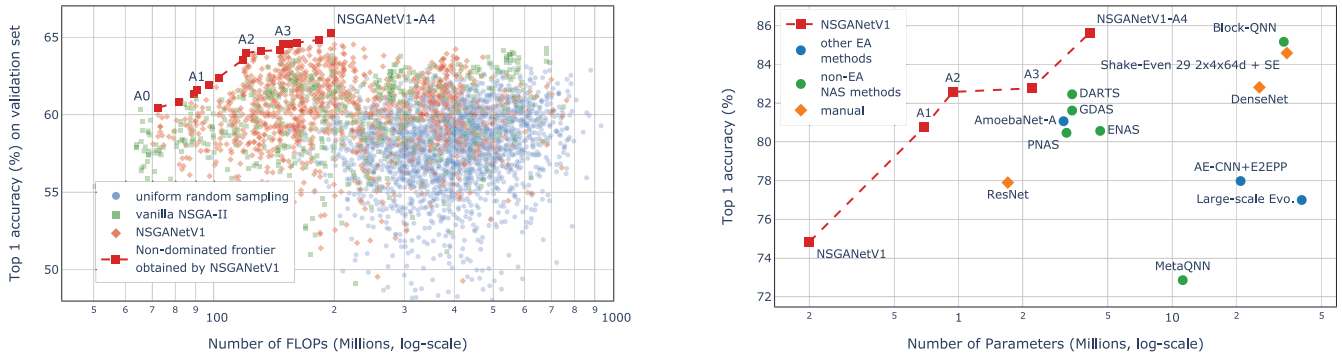
Motivated by efficiency and practicality considerations most existing NAS methods, including [8], [17], [18], [44], carry out the search process on the CIFAR-10 dataset. However, as we demonstrate through ablation studies in Section V-C the CIFAR-100 provides a more reliable measure of an architecture's efficacy in comparison to CIFAR-10. Based on this observation, in contrast to existing approaches, we use the more challenging CIFAR-100 dataset for the search process. Furthermore, we split the original CIFAR-100 training set (80%–20%) to create a training and validation set to prevent over-fitting to the training set and improve the generalizability. We emphasize that the original testing set is *never* used to guide the selection of architectures in any form during the search.

The search itself is repeated five times with different initial random seeds. We select and report the performance of the median run as measured by hypervolume (HV). Such a procedure ensures the reproducibility of our NAS experiments and mitigates the concerns that have arisen in recent NAS studies [45], [46]. We use the standard SGD algorithm for learning the associated weights for each architecture. Other hyper-parameter settings related to the search space, the gradient descent training, and the search strategy are summarized in Table I. We provide analysis aimed at justifying some of the hyper-parameter choices in Section V-C. All experiments are performed on 8 Nvidia 2080Ti GPU cards.

Our post-search training settings largely follow [9]. We extend the number of epochs to 600 with a batch size of 96 to thoroughly retrain the selected models from scratch. We also incorporate a data preprocessing technique *cutout* [47], and a regularization technique *scheduled path dropout* introduced in [8]. In addition, to further improve the training process, an auxiliary head classifier [1] is appended to the architecture at approximately 2/3 depth (right after the second resolution-reduction operation). The loss from this auxiliary head classifier, scaled by a constant factor 0.4, is aggregated with the loss from the original architecture before back-propagation during training.

C. Effectiveness of NSGANetV1

We first present the objective space distribution of all architectures generated by NSGANetV1 during the course of evolution on CIFAR-100, in Fig. 6(a). We include architectures generated by the original NSGA-II algorithm and uniform random sampling as references for comparison. Details of these two methods are provided in Section IV-D. From the set of nondominated solutions [outlined by red box markers in Fig. 6(a)], we select five architectures based on the ratio of the gain on accuracy over the sacrifice on FLOPs. For reference



(a) (b)

Fig. 6. (a) Accuracy versus FLOPs of all architectures generated by NSGANetV1 during the course of evolution on CIFAR-100. A subset of nondominated architectures (see text), named NSGANetV1-A0 to A4, are retrained thoroughly and compared with other peer methods in (b).

TABLE I
SUMMARY OF HYPER-PARAMETER SETTINGS

Categories	Parameters	Settings
search space	# of initial channels (Ch_{init})	32
	# of channel increments (Ch_{inc})	6
	# of repetitions of Normal blocks ($\mathcal{N}$)	4/5/6
gradient descent	batch size	128
	weight decay (L_2 regularization)	5.00E-04
	epochs	36/600
	learning rate schedule	Cosine Annealing [48]
search strategy	population size	40
	# of generations	30
	crossover probability	0.9
	mutation probability	0.1

purposes, we name these five architectures as NSGANetV1-A0 to -A4 in ascending FLOPs order. See Fig. 1 in the supplementary materials for a visualization of the searched architectures.

For comparison with other peer methods, we follow the training procedure in [9] and retrain the weights of NSGANetV1-A0 to -A4 on CIFAR-100, following the steps outlined in Section IV-B. We would like to mention that since most existing approaches do not report the number of FLOPs for the architectures used on the CIFAR-100 dataset, we instead compare their computational complexity through a number of parameters to prevent potential discrepancies from reimplementation. Fig. 6(b) shows the post-search architecture comparisons, NSGANetV1-A0 to -A4, i.e., the algorithms derived in this article, jointly dominate all other considered peer methods with a clear margin. More specifically, NSGANetV1-A1 is more accurate than peer EA method, AE-CNN-E2EPP [19], while being 30× *more efficient* in network parameters; NSGANetV1-A2 achieves better performance than AmoebaNet [17] and NSGA-Net [20] with 3× *fewer parameters*. Furthermore, NSGANetV1-A4 exceeds the classification accuracy of *Shake-Even 29* $2 \times 4 \times 64d + SE$ [37] using 8× *fewer parameters*. More comparisons can be found in Table II(b).

Following the practice adopted in most previous approaches [8], [9], [17], [31], [44], we measure the transferability of the obtained architectures by allowing the architectures evolved on one dataset (CIFAR-100 in this case) to be inherited and used on other datasets, by retraining the weights from scratch on the new dataset–in our case, on CIFAR-10 and ImageNet.

The effectiveness of NSGANetV1 is further validated by the transferred performance on the CIFAR-10 dataset. As we show in Fig. 7(a), the tradeoff frontier established by NSGANetV1-A0 to -A4 completely dominates the frontiers obtained by the peer EMO methods, both DPP-Net [30] and LEMONADE [32], as well as those obtained with other single-objective peer methods. More specifically, NSGANetV1-A0 uses 27× *fewer parameters* and achieves higher classification accuracy than large-scale Evo. [15]. NSGANetV1-A1 outperforms Hierarchical NAS [16] and DenseNet [3] in classification, while *saving* 122× and 51× in *parameters*. NSGANetV1-A2 uses 4× *less parameters* to achieve similar performance as compared to NSGA-Net [20]. Furthermore, NSGANetV1-A4 exceeds previous state-of-the-art results reported by Proxyless NAS [43] while being 1.4× *more compact*. Refer to Table II(a) for more comparisons.

For transfer performance comparison on the ImageNet dataset, we follow previous work [5], [8], [9], [44] and use the ImageNet-mobile setting, i.e., the setting where the number of FLOPs is less than 600M. The NSGANetV1-A0 is too simple for the ImageNet dataset and NSGANetV1-A4 exceeds the 600M FLOPs threshold for the mobile setting, so we provide results only for NSGANetV1-A1, -A2, and -A3. Fig. 7(b) compares the objective space with the other peer methods. Clearly, NSGANetV1 can achieve a better tradeoff between the objectives. NSGANetV1-A2 dominates a wide range of peer methods, including ShuffleNet [4] by human experts, NASNet-A [8] by RL, DARTS [9] by relaxation-based methods, and AmoebaNet-A [17] by EA. Moreover, NSGANetV1-A3 surpasses previous state-of-the-art performance reported by MobileNet-V2 [5] and AmoebaNet-C [17] on mobile-setting with a marginal overhead in FLOPs (1%–3%).

D. Efficiency of NSGANetV1

Comparing the search phase contribution to the success of different NAS algorithms can be difficult and ambiguous due to substantial differences in search spaces and training procedures used during the search. Therefore, we use *vanilla NSGA-II* and *uniform random sampling* as comparisons to

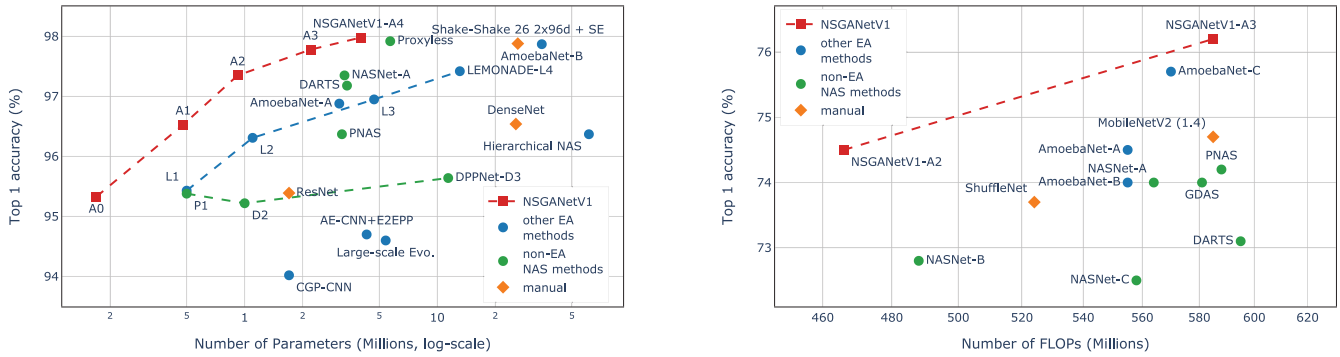


Fig. 7. Transferability of the NSGANetV1 architectures to (a) CIFAR-10 and (b) ImageNet. We compare Top-1 accuracy versus computational complexity. Architectures joined by dashed lines are from multiobjective algorithms.

TABLE II
COMPARISON BETWEEN NSGANETV1 AND OTHER BASELINE METHODS. NSGANETV1 ARCHITECTURES ARE OBTAINED BY SEARCHING ON CIFAR-100. NSGANETV1 RESULTS ON CIFAR-10 AND IMAGENET ARE OBTAINED BY RETRAINING THE WEIGHTS WITH IMAGES FROM THEIR RESPECTIVE DATASETS. RATIO-TO-NSGANETV1 INDICATES THE RESULTING SAVINGS ON #PARAMS AND #FLOPS. THE SEARCH COST IS COMPARED IN GPU-DAYS, CALCULATED BY MULTIPLYING THE NUMBER OF GPU CARDS DEPLOYED WITH THE EXECUTION TIME IN DAYS.
(a) CIFAR-10. (b) CIFAR-100. (c) IMAGENET

(a)

Architecture	Search Method	GPU-Days	Top-1 Acc.	#Params	Ratio-to-NSGANetV1
NSGANetV1-A0	**EA**	**27**	**95.33%**	**0.2M**	**1x**
CGP-CNN [24]	EA	27	94.02%	1.7M	8.5x
Large-scale Evo. [15]	EA	2,750	94.60%	5.4M	27x
AE-CNN+E2EPP [19]	EA	7	94.70%	4.3M	21x
ResNet [2]	manual	-	95.39%	1.7M	8.5x
NSGANetV1-A1	**EA**	**27**	**96.51%**	**0.5M**	**1x**
Hierarchical NAS [16]	EA	300	96.37%	61.3M	122x
PNAS [31]	SMBO	150	96.37%	3.2M	6.4x
DenseNet [3]	manual	-	96.54%	25.6M	51x
NSGANetV1-A2	**EA**	**27**	**97.35%**	**0.9M**	**1x**
CNN-GA [25]	EA	35	96.78%	2.9M	3.2x
AmoebaNet-A [17]	EA	3,150	96.88%	3.1M	3.4x
DARTS [9]	relaxation	1	97.18%	3.4M	3.8x
NSGA-Net [20]	EA	4	97.25%	3.3M	3.7x
NSGANetV1-A3	**EA**	**27**	**97.78%**	**2.2M**	**1x**
NASNet-A [8]	RL	1,575	97.35%	3.3M	1.5x
LEMONADE [32]	EA	90	97.42%	13.1M	6.0x
NSGANetV1-A4	**EA**	**27**	**97.98%**	**4.0M**	**1x**
AmoebaNet-B [17]	EA	3,150	97.87%	34.9M	8.7x
Proxyless NAS [43]	RL	1,500	97.92%	5.7 M	1.4x

(b)

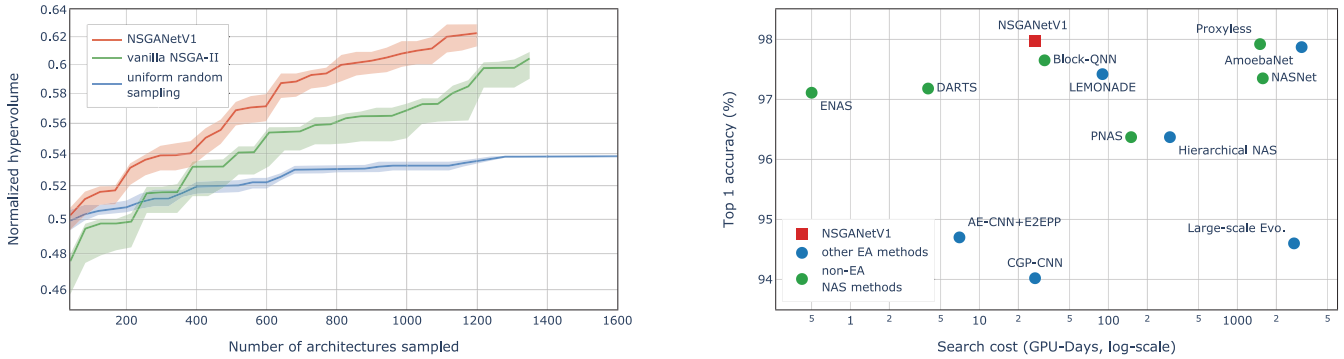
Architecture	Search Method	GPU-Days	Top-1 Acc.	#Params	Ratio-to-NSGANetV1
NSGANetV1-A0	**EA**	**27**	**74.83%**	**0.2M**	**1x**
Genetic CNN [14]	EA	17	70.95%	-	-
MetaQNN [49]	RL	90	72.86%	11.2M	56x
NSGANetV1-A1	**EA**	**27**	**80.77%**	**0.7M**	**1x**
Large-scale Evo. [15]	EA	2,750	77.00%	40.4M	58x
ResNet [2]	manual	-	77.90%	1.7M	2.4x
AE-CNN+E2EPP [19]	EA	10	77.98%	20.9M	30x
NSGA-Net [20]	EA	8	79.26%	3.3M	4.7x
CNN-GA [25]	EA	40	79.47%	4.1M	5.9x
PNAS [31]	SMBO	150	80.47%	3.2M	4.6x
ENAS [44]	RL	0.5	80.57%	4.6M	6.6x
NSGANetV1-A2	**EA**	**27**	**82.58%**	**0.9M**	**1x**
AmoebaNet-A [17]	EA	3,150	81.07%	3.1M	3.4x
GDAS [11]	relaxation	0.2	81.62%	3.4M	3.8x
DARTS [9]	relaxation	1	82.46%	3.4M	3.8x
NSGANetV1-A3	**EA**	**27**	**82.77%**	**2.2M**	**1x**
NSGANetV1-A4	**EA**	**27**	**85.62%**	**4.1M**	**1x**
DenseNet [3]	manual	-	82.82%	25.6M	6.2x
SENet [37]	manual	-	84.59%	34.4M	8.4x
Block-QNN [35]	RL	32	85.17%	33.3M	8.1x

(c)

Architecture	Search Method	GPU-Days	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-NSGANetV1	#FLOPs	Ratio-to-NSGANetV1
NSGANetV1-A1	**EA**	**27**	70.9%	90.0%	**3.0M**	**1x**	**270M**	**1x**
MobileNet-V2 [5]	manual	-	**72.0%**	**91.0%**	3.4M	1.1x	300M	1.1x
NSGANetV1-A2	**EA**	**27**	**74.5%**	**92.0%**	**4.1M**	**1x**	**466M**	**1x**
ShuffleNet [4]	manual	-	73.7%	-	5.4M	1.3x	524M	1.1x
NASNet-A [8]	RL	1,575	74.0%	91.3%	5.3M	1.3x	564M	1.2x
PNAS [31]	SMBO	150	74.2%	91.9%	5.1M	1.2x	588M	1.3x
AmoebaNet-A [17]	EA	3,150	74.5%	92.0%	5.1M	1.2x	555M	1.2x
DARTS [9]	relaxation	1	73.1%	91.0%	4.9M	1.2x	595M	1.3x
NSGANetV1-A3	**EA**	**27**	**76.2%**	**93.0%**	**5.0M**	**1x**	585M	1x
MobileNetV2 (1.4) [5]	manual	-	74.7%	92.5%	6.06M	1.2x	582M	1x
AmoebaNet-C [17]	EA	3,150	75.7%	92.4%	6.4M	1.3x	**570M**	**1x**

† SMBO stands for sequential model-based optimization. SENet is the abbreviation for Shake-Even 29 2x4x64d + SE.
‡ The CIFAR-100 accuracy and #params for ENAS [44] and DARTS [9] are from [11]. #Params for AE-CNN+E2EPP are from [18].

demonstrate the efficiency of the search phase in NSGANetV1. All three methods use the same search space and performance estimation strategy as described in Section III. The vanilla NSGA-II is implemented by discretizing the crossover and mutation operators in the original NSGA-II [29] algorithm with all hyper-parameters set to default values; and it does not utilize any additional enhancements—e.g., the Bayesian-network-model-based exploitation. The uniform random sampling method is implemented by replacing the crossover and mutation operators in the original NSGA-II algorithm with an initialization method that uniformly samples the search space. We run each of the three methods five times and record the



(a) (b)

Fig. 8. Search efficiency comparison between NSGANetV1 and other baselines in terms of (a) HV and (b) required compute time in GPU-Days. The search cost is measured on CIFAR-10 for most methods, except NSGANetV1 and Block-QNN [35], where the CIFAR-100 dataset is used for.

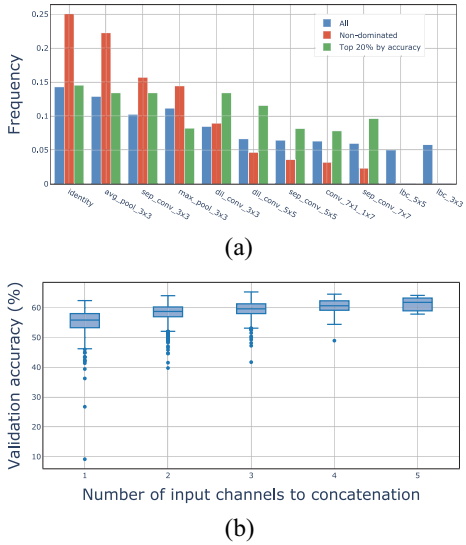
25-percentile, median, and 75-percentile of the normalized HV (NHV) that we obtain. The NHV measurements shown in Fig. 8(a) suggest that NSGANetV1 is capable of finding more accurate and simpler architectures than vanilla NSGA-II or uniform random sampling (even with an extended search budget), in a more efficient manner.

Apart from the HV metric, another important aspect of demonstrating the efficiency of NAS is the computational complexity of the methods. Since theoretical analysis of the computational complexity of different NAS methods is challenging, we compare the computation time spent on GPUs, *GPU-Days*, by each approach to arrive at the reported architectures. The number of GPU-Days is calculated by multiplying the number of employed GPU cards by the execution time (in units of days).

One run of NSGANetV1 on the CIFAR-100 dataset takes approximately 27 GPU-Days to finish, averaged over five runs. The search costs of most of the peer methods are measured on the CIFAR-10 dataset, except for Block-QNN [35] which is measured on CIFAR-100. From the search cost comparison in Fig. 8(b), we observe that our proposed algorithm is more efficient at identifying a set of architectures than a number of other approaches, and the set of architectures obtained has higher performance. More specifically, NSGANetV1 simultaneously finds multiple architectures while using 10× to 100× *less GPU-days* in searching than most of the considered peer methods, including Hierarchical NAS [16], AmoebaNet [17], NASNet [8], and Proxyless NAS [43], all of which find a single architecture at a time. When compared to the peer multiobjective NAS method, LEMONADE [32], NSGANetV1 manages to obtain a better (in the Pareto dominance sense) set of architectures than LEMONADE with 3× *fewer GPU-Days*. Further experiments and comparisons are provided in the supplementary materials under Section VII-A.

E. Observations on Evolved Architectures

Population-based approaches with multiple conflicting objectives often lead to a set of diverse solution candidates, which can be "mined" for commonly shared design principles [50]. In order to discover any patterns for more efficient design, we analyzed the entire history of architectures



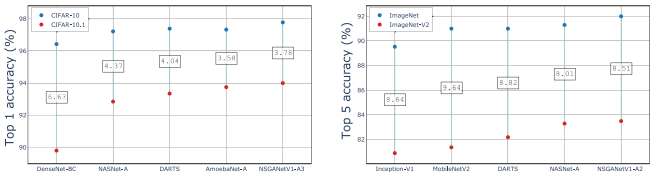
(a) (b)

Fig. 9. *Post-Search Analysis:* (a) Frequency of each operation selected during the search. (b) Effect of the number of input channels that are concatenation on the validation accuracy. More channels improve the predictive performance of the architectures, but adversely affect the computational efficiency.

generated by NSGANetV1. We make the following observations.

1) Nonparametric operations—e.g., skip connections (*identity*) and average pooling (*avg_pool_3x3*)—are effective in trading off performance for complexity. Empirically, we notice that three out of the four most frequently used operations in nondominated architectures are nonparametric, as shown in Fig. 9(a) (see also supplementary materials under Section VII-B for our follow-up study).
2) Larger kernel size and parallel operations improve classification accuracy, as shown in Fig. 9(a) and (b), respectively. In particular, the frequencies of convolutions with large kernel sizes (e.g., *dil_conv_5x5* and *conv_7x1_1x7*) are significantly higher in the top-20% most accurate architectures than in nondominated architectures in general, which must also balance FLOPs. Similar findings are also reported in previous work [17], [42].

The above common properties of multiple final nondominated solutions stay as important knowledge for future applications. It is noteworthy that such a post-optimal knowledge



(a) (b)

Fig. 10. *Generalization:* We evaluate the models on new and extended test sets for (a) CIFAR-10.1 and (b) ImageNet-V2. Numbers in the boxes indicate absolute drop in accuracy (%).

extraction process is possible only from a multiobjective optimization study, another benefit that we enjoy for posing NAS as a multiobjective optimization problem.

V. Further Analysis

The overarching goal of NAS is to find architecture models that generalize to new instances of what the models were trained on. We usually quantify generalization by measuring the performance of a model on a held-out testing set. Since many computer vision benchmark datasets, including the three datasets used in this article—i.e., CIFAR-10, CIFAR-100, and ImageNet, have been the focus of intense research for almost a decade, does the steady stream of promising empirical results from NAS simply arise from overfitting of these excessively reused testing sets? Does advancement on these testing sets imply better robustness vis-a-vis commonly observable corruptions in images and adversarial images by which the human vision system is more robust? To answer these questions in a quantitative manner, in this section, we provide systematic studies on newly proposed testing sets from the CNN literature, followed by hyper-parameter analysis.

A. Generalization

By mimicking the documented curation process of the original CIFAR-10 and ImageNet datasets, Recht *et al.* [51] proposed two new testing sets, CIFAR-10.1 and ImageNet-V2. Refer to supplementary materials under Section V-A for details and examples of the new testing sets. Representative architectures are selected from each of the main categories (i.e., RL, EA, relaxation-based, and manual). The selected architectures are similar in the number of parameters or FLOPs, except DenseNet-BC [3], and Inception-V1 [1]. All architectures are trained on the original CIFAR-10 and ImageNet training sets as in Section IV-C, then evaluated on CIFAR-10.1 and ImageNet-V2, respectively.

It is evident from the results summarized in Fig. 10(a) and (b) that there is a significant drop in accuracy of 3%–7% on CIFAR-10.1 and 8% to 10% on ImageNet-V2 across architectures. However, the relative rank-order of accuracy on the original testing sets translates well to the new testing sets, i.e., the architecture with the highest accuracy on the original testing set (NSGANetV1 in this case) is also the architecture with the highest accuracy on new testing sets. Additionally, we observe that the accuracy gains on the original testing sets translate to larger gains on the new testing sets, especially in the case of CIFAR-10 (curvatures of red versus blue markers in Fig. 10). These results provide evidence that extensive benchmarking on the original testing sets is an effective way to measure the progress of architectures.

B. Robustness

The vulnerability to small changes in query images may very likely prevent the deployment of deep learning vision systems in safety-critical applications at scale. Understanding the architectural advancements under the scope of robustness against various forms of corruption is still in its infancy. Hendrycks and Dietterich [52] recently introduced two new testing datasets, CIFAR-10-C and CIFAR-100-C, by applying commonly observable corruptions (e.g., noise, weather, compression, etc.) to the clean images from the original datasets. Each dataset contains images perturbed by 19 different types of corruption at five different levels of severity. More details and visualizations are provided in supplementary materials under Section V-B. In addition, we include adversarial images as examples of worst-case corruption. We use the fast gradient signed method (FGSM) [53] to construct adversarial examples for both the CIFAR-10 and -100 datasets. The severity of the attack is controlled via a hyper-parameter ϵ as shown below

$$\boldsymbol{x}' = \boldsymbol{x} + \epsilon \ \text{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y_{\text{true}}))$$

where $\boldsymbol{x}$ is the original image, $\boldsymbol{x}'$ is the adversarial image, y_{true} is the true class label, and $\mathcal{L}$ is the cross-entropy loss. Following the previous section, we pick representative architectures of similar complexities from each of the main categories. Using the weights learned on the clean images from the original CIFAR-10/100 training sets, we evaluate each architecture's classification performance on the corrupted datasets as our measure of robustness.

Our empirical findings summarized in Fig. 11(a) and (b) appear to suggest that a positive correlation exists between the generalization performance on clean data and data under commonly observable corruptions—i.e., we observe that NSGANetV1 architectures perform noticeably better than other peer methods' architectures on corrupted datasets even though the robustness measurement was never a part of the architecture selection process in NSGANetV1. However, we emphasize that no architectures are considered robust to corruption, especially under adversarial attacks. We observe that the architectural advancements have translated to minuscule improvements in robustness against adversarial examples. The classification accuracy of all selected architectures deteriorates drastically with minor increments in adversarial attack severity ϵ, leading to the question of whether architecture is the "right" ingredient to investigate in pursuit of adversarial robustness. A further step toward answering this question is provided in supplementary materials under Section V-C.

C. Ablation Studies

Dataset for Search: As previously mentioned in Section III-B, our proposed method differs from most of the existing peer methods in the choice of datasets on which the search is carried out. Instead of directly following the current practice of using the CIFAR-10 dataset, we investigated the utility of search on multiple benchmark

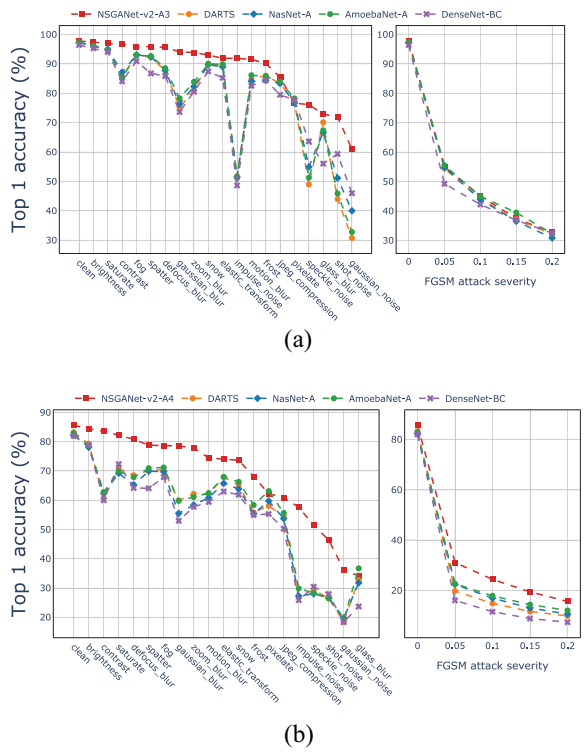


Fig. 11. *Robustness:* Effect of commonly observable corruptions and adversarial attacks on (a) CIFAR-10 and (b) CIFAR-100. Higher values of ϵ indicate more severe adversarial attacks. We use prediction accuracy on the corrupted test images (from each dataset) as a measurement of robustness.

datasets in terms of their ability to provide reliable estimates of classification accuracy and generalization. We carefully selected four datasets, SVHN [54], fashionMNIST [55], CIFAR-10, and CIFAR-100 for comparison. The choice was based on a number of factors including the number of classes, numbers of training examples, resolutions, and required training times. We uniformly sampled 40 architectures from the search space (described in Section III) along with five architectures generated by other peer NAS methods. We trained every architecture three times with different initial random seeds and report the averaged classification accuracy on each of the four datasets in Fig. 12(a). Empirically, we observe that the CIFAR-100 dataset is challenging enough for architectural differences to affect predicted performance. This can be observed in Fig. 12(a) where the variation (blue boxes) in classification accuracy across architectures is noticeably larger on CIFAR-100 than on the other three datasets. In addition, we observe that mean differences in classification accuracy on CIFAR-100 between randomly generated architectures and architectures from principle-based methods have higher deviations, suggesting that it is less likely to find a good architecture on CIFAR-100 by chance.

Proxy Models: The main computational bottleneck of NAS approaches resides in evaluating the classification accuracy of the architectures by invoking the lower-level weight optimization. One such evaluation typically takes hours to finish, which limits the practical utility of the search under a constrained search budget. In our proposed algorithm, we adopt the concept of a proxy model [8], [17]. Proxy models

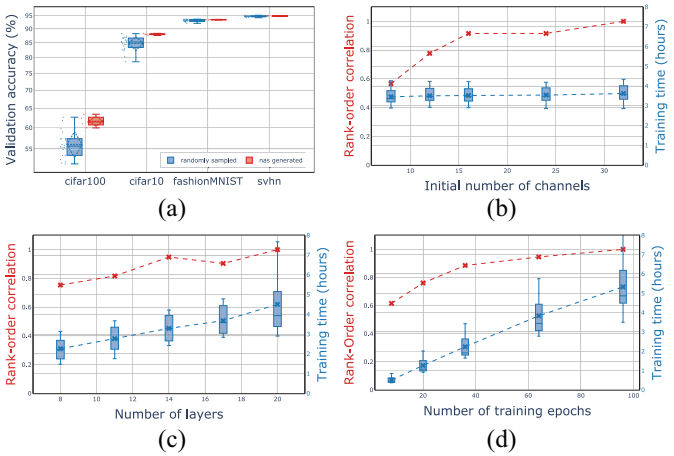


Fig. 12. (a) Mean classification accuracy distribution of randomly generated architectures and architectures from peer NAS methods on four datasets. Spearman correlation in performance (red lines) versus Savings in gradient descent wall time (blue boxes) by reducing, (b) number of channels in layers, (c) number of layers, and (d) number of epochs to train. Note that (b)–(d) have two y-axis labels corresponding to the color of the lines.

are small-scale versions of the intended architectures, where the number of layers [N in Fig. 2(a)] and the number of channels [Ch_{init} in Fig. 2(a)] in each layer are reduced. Due to the downscaling, proxy models typically require much less compute time to train.[2] However, there exists a tradeoff between gains in computation efficiency and loss of prediction accuracy. Therefore, it is not necessary that the performance of an architecture measured at proxy-model scale can serve as a reliable indicator of the architecture's performance measured at the desired scale.

To determine the smallest proxy model that can provide a reliable estimate of performance at a larger scale, we conducted parametric studies that gradually reduced the sizes of the proxy models of 100 randomly sampled architectures from our search space. Then, we measured the rank-order correlation and the savings in lower-level optimization compute time between the proxy models and the same architectures at the full scale. Fig. 12(b)–(d) show the effect of numbers of channels, layers, and epochs, respectively, on the training time, and the Spearman rank-order correlation between the proxy and full scale models. We make the following observations: 1) increasing the number of channels does not significantly affect the wall clock time and 2) reducing the number of layers or training epochs significantly reduces the wall clock time but also reduces the rank-order correlation. Based on these observations and the exact tradeoffs from the plots, for our proxy model, we set the number of channels to 36 (maximum desired), number of epochs to 36, and the number of layers to 14. Empirically, we found that this choice of parameters offers a good tradeoff between the practicality of search and reliability of proxy models.

VI. Application to Chest X-Ray Classification

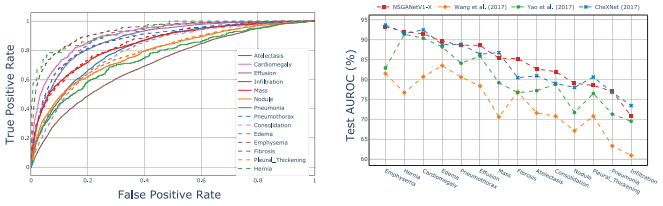
The ChestX-Ray14 benchmark was recently introduced in [56]. The dataset contains 112 120 high-resolution

[2]Small architecture size allows larger batch size to be used and a lower number of epochs to converge under Algorithm 2.

TABLE III
AUROC ON CHESTX-RAY14 TESTING SET

Method	Type	#Params	Test AUROC (%)
Wang et al. (2017) [56]	manual	-	73.8
Yao et al. (2017) [57]	manual	-	79.8
CheXNet (2017) [58]	manual	7.0M	84.4
Google AutoML (2018) [59]	RL	-	79.7
LEAF (2019) [60]	EA	-	84.3
NSGANetV1-A3	**EA**	**5.0M**	**84.7**
NSGANetV1-X	**EA**	**2.2M**	**84.6**

† Google AutoML result is from [60].
‡ NSGANetV1-A3 represents results under the standard transfer learning setup.



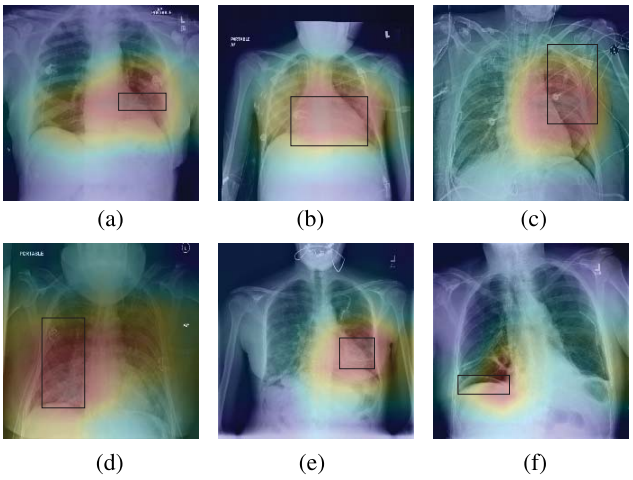
(a) (b)

Fig. 13. (a) NSGANetV1-X multilabel classification performance on ChestX-Ray14 and (b) class-wise mean test AUROC comparison with peer methods.

frontal-view chest X-ray images from 30 805 patients, and each image is labeled with one or multiple common thorax diseases, or "Normal," otherwise. More details are provided in supplementary materials under Section V-C. Past approaches [56]–[58] typically extend from existing architectures, and the current state-of-the-art method [58] uses a variant of the DenseNet [3] architecture, which is designed manually by human experts. For reference purpose, we call the obtained architecture NSGANetV1-X, and we retrain the weights thoroughly from scratch with an extended number of epochs. The learning rate is gradually reduced when the AUROC on the validation set plateaus.

Table III compares the performance of NSGANetV1-X with peer methods that are extended from existing manually designed architectures. This includes architectures used by the authors who originally introduced the ChestX-Ray14 dataset [56], and the CheXNet [58], which is the current state-of-the-art on this dataset. We also include results from commercial AutoML systems, i.e., Google AutoML [59], and LEAF [60], as comparisons with NAS-based methods. The setup details of these two AutoML systems are available in [60]. Noticeably, the performance of NSGANetV1-X exceeds Google AutoML's by a large margin of nearly 4 *AUROC points*. In addition, NSGANetV1-X matches the state-of-the-art results from human engineered CheXNet, while using 3.2× *fewer parameters*. For completeness, we also include the result from NSGANetV1-A3, which is evolved on CIFAR-100, to demonstrate the transfer learning capabilities of NSGANetV1.

More detailed results showing the disease-wise ROC curve of NSGANetV1-X and disease-wise AUROC comparison with other peer methods are provided in Fig. 13(a) and (b), respectively. To understand the pattern behind the disease classification decisions of NSGANetV1-X, we visualize the class activation map (CAM) [61], which is commonly adopted



(a) (b) (c)
(d) (e) (f)

Fig. 14. Examples of CAM [61] of NSGANetV1-X, highlighting the class-specific discriminative regions. The ground truth bounding boxes are plotted over the heatmaps. (a) Atelectasis. (b) Cardiomegaly. (c) Effusion. (d) Infiltrate. (e) Pneumonia. (f) Pneumothorax.

for localizing the discriminative regions for image classification. In the examples shown in Fig. 14(a)–(f), stronger CAM areas are covered with warmer colors. We also outline the bounding boxes provided by the ChestX-Ray14 dataset [56] as references.

These results further validate the ability of our proposed algorithm to generate task-dependent architectures automatically. Conventional approaches, e.g., transfer learning from existing architectures, can be effective in yielding similar performance, however, as demonstrated by NSGANetV1, simultaneously considering complexity along with performance in an algorithmic fashion allows architectures to be practically deployed in resource-constrained environments. We observe this phenomenon in another application of NSGANetV1 to keypoint prediction on cars (see the supplementary materials under Section V-D).

VII. CONCLUSION

In this article, we have presented NSGANetV1, an evolutionary multiobjective algorithm for NAS. NSGANetV1 explores the design space of architectures through recombining and mutating architectural components. NSGANetV1 further improves the search efficiency by exploiting the patterns among the past successful architectures via distribution estimation through a BN model. Experiments on CIFAR-10, CIFAR-100, and ImageNet datasets have demonstrated the effectiveness of NSGANetV1. Further analysis toward validating the generalization and robustness aspects of the obtained architectures is also provided along with an application to common thorax disease classification on human chest X-rays. We believe these results are encouraging and demonstrate the importance of customized and efficient EAs for NAS in achieving superior performance compared to other contemporary ML methods.

ACKNOWLEDGMENT

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and

do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, 2015, pp. 1–9.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 2261–2269.

[4] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 6848–6856.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.

[6] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 4284–4293.

[7] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[8] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 8697–8710.

[9] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[10] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[11] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 1761–1770.

[12] B. Wu *et al.*, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 10734–10742.

[13] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 6105–6114.

[14] L. Xi and A. Yuille, "Genetic CNN," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, 2017, pp. 1388–1397.

[15] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 2902–2911.

[16] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

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

[18] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

[19] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.

[20] Z. Lu *et al.*, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2019, pp. 419–427.

[21] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[22] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[23] G. S. Hornby, "ALPS: The age-layered population structure for reducing the problem of premature convergence," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2006, pp. 815–822.

[24] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2017, pp. 5369–5373.

[25] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[26] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 3, pp. 397–415, May 2008.

[27] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1310–1322, Apr. 2020.

[28] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *Proc. Int. Conf. Mach. Learn. (ICML) AutoML Workshop*, 2017.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[30] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 540–555.

[31] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–35.

[32] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[33] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 564–572.

[34] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[35] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 2423–2432.

[36] G. Eichfelder, "Multiobjective bilevel optimization," *Math. Program.*, vol. 123, no. 2, pp. 419–449, 2010.

[37] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 7132–7141.

[38] Y. Leung, Y. Gao, and Z.-B. Xu, "Degree of population diversity—A perspective on premature convergence in genetic algorithms and its markov chain analysis," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1165–1176, Sep. 1997.

[39] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.

[40] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 2, pp. 115–148, 1995.

[41] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 1999, pp. 840–851.

[42] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 5987–5995.

[43] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[44] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4095–4104.

[45] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," 2019. [Online]. Available: arXiv:1902.07638.

[46] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Seoul, South Korea, 2019, pp. 1284–1293.

[47] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017. [Online]. Available: arXiv:1708.04552.

[48] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[49] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[50] K. Deb and A. Srinivasan, "Innovization: Innovating design principles through optimization," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2006, pp. 1629–1636.

[51] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do ImageNet classifiers generalize to ImageNet?" 2019. [Online]. Available: arXiv:1902.10811.
[52] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.
[53] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014. [Online]. Available: arXiv:1412.6572.
[54] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS) Workshop Deep Learn. Unsupervised Feature Learn.*, 2011.
[55] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017. [Online]. Available: arXiv:1708.07747.
[56] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "ChestX-ray8: Hospital-scale chest X-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 3462–3471.
[57] L. Yao, E. Poblenz, D. Dagunts, B. Covington, D. Bernard, and K. Lyman, "Learning to diagnose from scratch by exploiting dependencies among labels," 2017. [Online]. Available: arXiv:1710.10501.
[58] P. Rajpurkar *et al.*, "CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning," 2017. [Online]. Available: arXiv:1711.05225.
[59] *Automl For Large Scale Image Classification and Object Detection*, Google Blog, Mountain View, CA, USA, 2017. [Online]. Available: https://research.googleblog.com/2017/11/automl-for-large-scaleimage.html
[60] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural AutoML for deep learning," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2019, pp. 401–409.
[61] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 2921–2929.

Zhichao Lu (Student Member, IEEE) received the bachelor's and Ph.D. degrees in electrical and computer engineering from Michigan State University, East Lansing, MI, USA, in 2013 and 2020, respectively.

His research interests are in the field of evolutionary machine learning, notably machine learning assisted evolutionary algorithms, automated machine learning, and in particular evolutionary neural architecture search.

Dr. Lu received the Best Paper Award at GECCO 2019 under evolutionary machine learning track.

Ian Whalen received the bachelor's degree in computer science and mathematics and the master's degree in computer science from Michigan State University, East Lansing, MI, USA, in 2017 and 2018, respectively.

He is currently a Consultant Level Data Scientist with QuantumBlack, a McKinsey Company, New York, NY, USA. In industry, he has experience prototyping and assetization machine learning solutions to complex problems across sectors, including banking, energy, mining, and nonprofit organizations.

Yashesh Dhebar received the bachelor's and master's degrees in mechanical engineering from the Indian Institute of Technology Kanpur, Kanpur, India, in 2015, with his thesis on modular robotics and their operation in cluttered environments. He is currently pursuing the Ph.D. degree with the Department of Mechanical Engineering, Michigan State University, East Lansing, MI, USA.

He is currently actively pursuing his research in the field of explainable and interpretable artificial intelligence, machine learning, and knowledge discovery. He has worked on applied optimization and machine learning projects with industry collaborators under the guidance of Prof. K. Deb.

Kalyanmoy Deb (Fellow, IEEE) received the bachelor's degree in mechanical engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 1985, and the master's and Ph.D. degrees from the University of Alabama, Tuscaloosa, AL, USA, in 1989 and 1991, respectively.

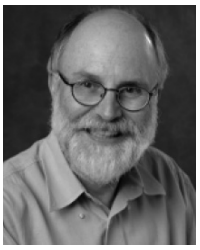
He is the Koenig Endowed Chair Professor with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, USA. He is largely known for his seminal research in evolutionary multicriterion optimization. He has published over 544 international journal and conference research papers. His current research interests include evolutionary optimization and its application in design, modeling, AI, and machine learning.

Dr. Deb is a recipient of the IEEE CIS EC Pioneer Award in 2018, the Lifetime Achievement Award from Clarivate Analytics in 2017, the Infosys Prize in 2012, and the Edgeworth-Pareto Award in 2008. He is a Fellow of ASME.

Erik D. Goodman received the Ph.D. degree in computer and communication sciences from the University of Michigan, Ann Arbor, MI, USA, 1972.

He is the PI and the Executive Director of the BEACON Center for the Study of Evolution in Action, an NSF Science and Technology Center headquartered with Michigan State University, East Lansing, MI, USA, funded at $47.5 million for 2010–2020. He has been an Assistant Professor through Full Professor in Electrical and Computer Engineering with Michigan State University since 1971, and also a Professor of Mechanical Engineering and Computer Science and Engineering. He was the Director, Case Center for Computer-Aided Engineering and Manufacturing from1983 to 2002; and the Director and Co-founder and VP Technology, Red Cedar Technology, Inc., East Lansing, which develops design optimization software, currently owned by Siemens.

Dr. Goodman was the Chair and a Senior Fellow of International Society for Genetic and Evolutionary Computation. He was the Founding Chair of ACM SIG on Genetic and Evolutionary Computation, in 2005.



Wolfgang Banzhaf received a "Diplom in Physik" degree in physics (equivalent to a M.Sc.) from the Ludwig-Maximilians-University, Munich, Germany, and the Dr.rer.nat (Ph.D.) degree from the Department of Physics, Technische Hochschule Karlsruhe (currently, Karlsruhe Institute of Technology), Karlsruhe, Germany.

He is the John R. Koza Chair for Genetic Programming and a Professor with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA. His research interests are in the field of bio-inspired computing, notably evolutionary computation and complex adaptive system, and in particular genetic programming.

Prof. Banzhaf won the EvoStar Award for sustained contributions to the field of Evolutionary Computation in Europe. He is the Founding Editor-in-Chief of *Genetic Programming and Evolving Machines* (Springer), He is a Senior Fellow of the International Society for Genetic and Evolutionary Computation.

Vishnu Naresh Boddeti (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2013.

He is an Assistant Professor with the Computer Science Department, Michigan State University, East Lansing, MI, USA. His research interests are in computer vision, pattern recognition, and machine learning.

Dr. Boddeti received the Best Paper Award at BTAS 2013, the Best Student Paper Award at ACCV 2018, and the Best Paper Award at GECCO 2019.