

Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification

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

arXiv:1912.01369v3 [cs.CV] 15 Sep 2020

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; (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. Code is available at <https://github.com/mikelzc1990/nsganetv1>.

Index Terms—Neural architecture search (NAS), evolutionary deep learning, convolutional neural networks (CNNs), genetic algorithms (GAs)

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.

The authors are with Michigan State University, East Lansing, MI, 48824 USA, Corresponding author's e-mail: (luzhicha@msu.edu).

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.

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).

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 multi-objective optimization. Despite recent advances in RL and relaxation-based NAS methods, they are still not readily applicable for multi-objective 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 machine learning (ML)-oriented approaches, especially under the scope of multi-objective 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 paper, we present NSGANetV1, a multi-objective evolutionary algorithm 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 paper 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 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 pre-defined preference weighting of objectives prior to the search. Further post-optimal analysis of the set of non-dominated architectures often reveals valuable design principles, which is another benefit of posing NAS as a multi-objective optimization problem, as is done in NSGANetV1.
- 3) From an algorithmic perspective, we extend our previous work [20] in a number of ways: (i) an expanded search space to include five more layer operations and one more option that controls the width of the network, (ii) improved encoding, mutation and crossover operators accompanying the modified search space, and (iii) 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: (i) adding three more tasks, including medical imaging, robustness to adversarial attacks, and car key-point estimation; and (ii) 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 paper 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 deep neural networks 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 convolutional neural networks 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] introduce 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] evolve a hierarchical representation that allows non-modular 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] use Cartesian genetic programming to assemble an architecture from existing modular blocks (e.g., Residual blocks). Sun et al. in [19] use 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 $3\times$ 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.

Multi-objective NAS: In this work, the term *multi-objective* NAS refers to methods that simultaneously approximate the entire set of efficient trade-off architectures in one run [26], [27]. Kim et al. [28] presented NEMO, one of the earliest evolutionary multi-objective 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

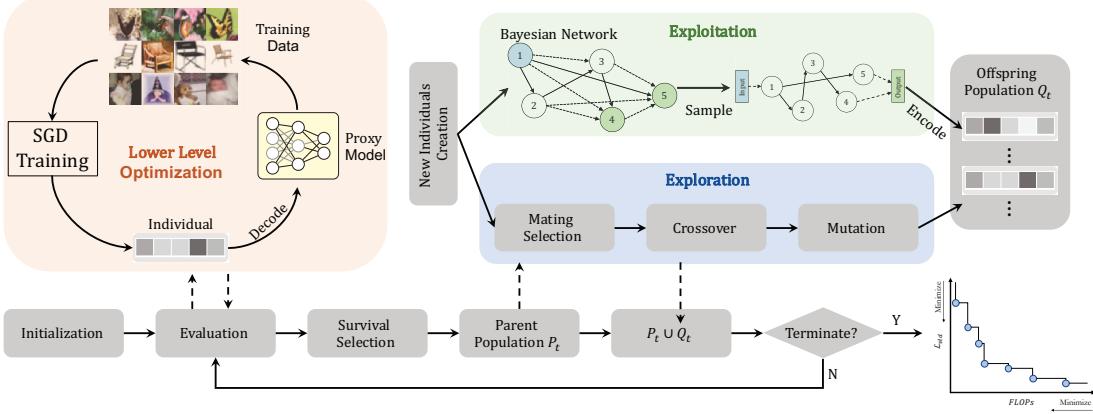


Fig. 1: Overview: Given a dataset and objectives, NSGANetV1 designs a set of custom architectures spanning the trading-off front. NSGANetV1 estimate the performance of an architecture through its *proxy model*, optimized by Stochastic Gradient Descent (SGD) in the lower-level. The search proceeds in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

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] present 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 re-ranking 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 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 reliabil-

ity aspects of the proxy model; through a series of systematic studies in a controlled setting, we empirically establish the trade-off 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.

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 multi-objective bilevel optimization problem* [36]. We mathematically formulate the problem as,

$$\begin{aligned} & \text{minimize } \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}; \mathbf{w}^*(\mathbf{x})), f_2(\mathbf{x}))^T, \\ & \text{subject to } \mathbf{w}^*(\mathbf{x}) \in \arg\min \mathcal{L}(\mathbf{w}; \mathbf{x}), \\ & \mathbf{x} \in \Omega_x, \quad \mathbf{w} \in \Omega_w, \end{aligned} \quad (1)$$

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

*See the supplement, Section III, for more about bilevel optimization.

Algorithm 1: General framework of NSGANetV1

Input : Complexity objective \tilde{f} (see Eq. (3)), 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  $\rho$  according to Eq. (2).
37   else
38      $\rho \leftarrow 1$  // remain unchanged from initial value.
39   end
40 end
41 Return parent population  $P$ .

```

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: (i) *exploration*, with the goal of discovering diverse ways to construct architectures, and (ii) *exploitation* that reinforces the emerging patterns commonly shared among the architectures successful during exploration. A set of architectures representing efficient trade-offs 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., # of channels), the depth (i.e., # 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], and 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], [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. 2a 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:

- identity
- 3x3 max pooling
- 3x3 average pooling
- squeeze-and-excitation [37]
- 3x3 local binary conv [6]
- 5x5 local binary conv [6]
- 3x3 dilated convolution
- 5x5 dilated convolution
- 3x3 depthwise-separable conv
- 5x5 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x7 then 7x1 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. 2b-2d for pictorial illustrations. The search space we consider in this paper is an expanded version of the *micro search space* used in our previous work [20]. Specifically, the current search space (i) search for a factor that gradually increments the channel size of each block with depth (see Fig. 2b) as opposed to sharply doubling the channel size when down-sampling. (ii) considers an expanded set of primitive operations to include both more

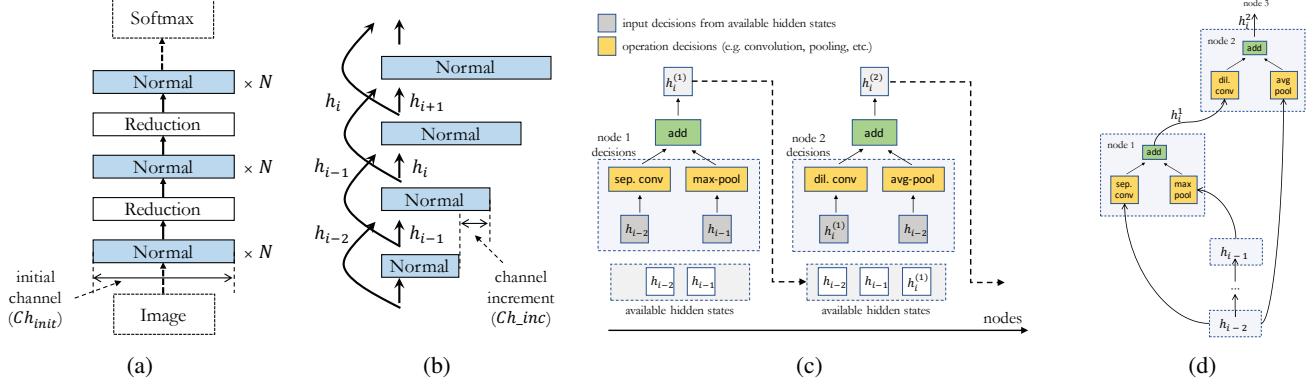


Fig. 2: Schematic of the NSGANetV1 search space motivated from [8]: (a) An architecture is composed of stacked blocks. (b) The number of channels in each block is gradually increased with depth of the network. (c) Each block is composed of five nodes, where each node is a two-branched computation applied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

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 towards finding more accurate and efficient architectures, we consider two metrics as objectives, namely, classification accuracy and architecture complexity. 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

Algorithm 2: Performance Evaluation of a CNN

```

Input : The architecture  $\alpha$ , training data  $\mathcal{D}_{trn}$ , validation data  $\mathcal{D}_{vld}$ , number of epochs  $T$ , weight decay  $\lambda$ , initial learning rate  $\eta_{max}$ .
1  $\omega \leftarrow$  Randomly initialize the weights in  $\alpha$ ;
2  $t \leftarrow 0$ ;
3 while  $t < T$  do
4    $\eta \leftarrow \frac{1}{2}\eta_{max}(1 + \cos(\frac{t}{T}\pi))$ ;
5   for each data-batch in  $\mathcal{D}_{trn}$  do
6      $\mathcal{L} \leftarrow$  Cross-entropy loss on the data-batch;
7      $\nabla\omega \leftarrow$  Compute the gradient by  $\partial\mathcal{L}/\partial\omega$ ;
8      $\omega \leftarrow (1 - \lambda)\omega - \eta\nabla\omega$ ;
9   end
10   $t \leftarrow t + 1$ ;
11 end
12  $acc \leftarrow$  Compute accuracy of  $\alpha(\omega)$  on  $\mathcal{D}_{vld}$ ;
13 Return the classification accuracy  $acc$ .

```

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 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 non-dominated ranking and the “crowdedness” 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; (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 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 multi-objective 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.

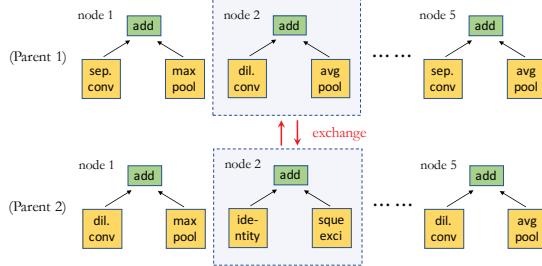


Fig. 3: Illustration of node-level crossover.

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.

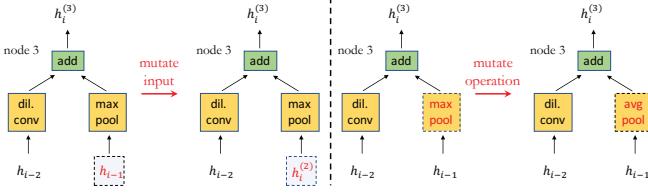


Fig. 4: Input and Operation Mutation: Dashed line boxes with red color highlight the mutation. h_{i-2} and h_{i-1} are outputs from previous-previous and previous blocks, respectively. $h_i^{(3)}$ indicates output from node 3 of the current block.

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. Figure 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 case of mutation on layer operations.

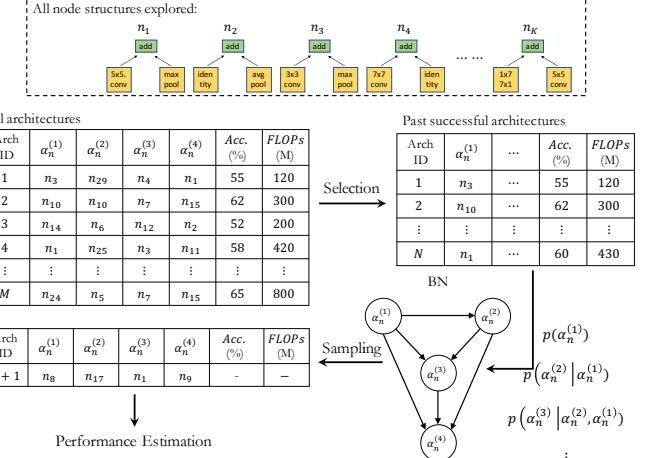


Fig. 5: Illustrative example of BN-based exploitation step in NSGANetV1: given past successful architectures, we construct a BN relating the dependencies between the four nodes inside the Normal block. A new architecture is then sampled from this BN and proceeds forward for performance estimation.

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 Bayesian Network (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:

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

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 non-dominated 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: architectures manually designed by human experts, non-EA- (mainly RL or relaxation)-based, and EA-based. Human engineered architectures include ResNet [2], ResNeXt [42], and DenseNet [3], etc. The second and third groups range from earlier methods [7], [14], [15] that are oriented towards "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.

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 (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

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 re-train the selected models from scratch. We also incorporate a data pre-processing 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. 6a. 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 non-dominated solutions (outlined by red box markers in Fig. 6a), we select five architectures based on the ratio of the gain on accuracy over the sacrifice on FLOPs. For reference 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 re-train 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 number of parameters to prevent potential discrepancies from re-implementation. Fig. 6b

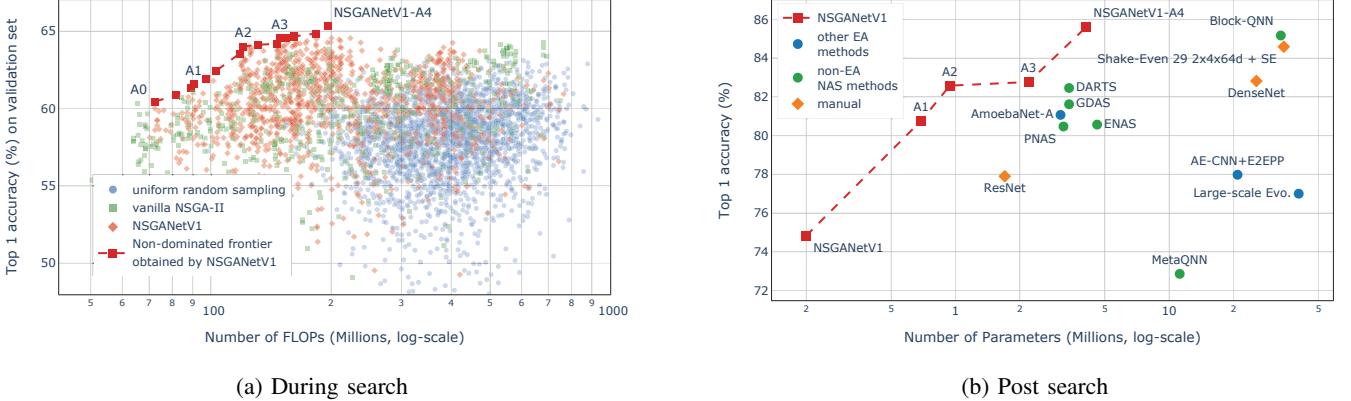


Fig. 6: (a) Accuracy vs. FLOPs of all architectures generated by NSGANetV1 during the course of evolution on CIFAR-100. A subset of non-dominated architectures (see text), named NSGANetV1-A0 to A4, are re-trained thoroughly and compared with other peer methods in (b).

shows the post-search architecture comparisons, NSGANetV1-A0 to A4, i.e., the algorithms derived in this paper, 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 **30x more efficient** in network parameters; NSGANetV1-A2 achieves better performance than AmoebaNet [17] and NSGA-Net [20] with **3x fewer parameters**. Furthermore, NSGANetV1-A4 exceeds the classification accuracy of *Shake-Even 29 2x4x64d + SE* [37] using **8x fewer parameters**. More comparisons can be found in Table IIb.

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 Figs. 7a, the trade-off 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 **27x 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 122x and 51x in parameters**. NSGANetV1-A2 uses **4x 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 **1.4x more compact**. Refer to Table IIa 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 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. 7b compares the objective space with the other peer methods. Clearly, NSGANetV1 can achieve a better trade-off 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 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 hyperparameters 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 25-percentile, median and 75-percentile of the normalized HV (NHV) that we obtain. The NHV measurements shown in Fig. 8a 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 Graphics

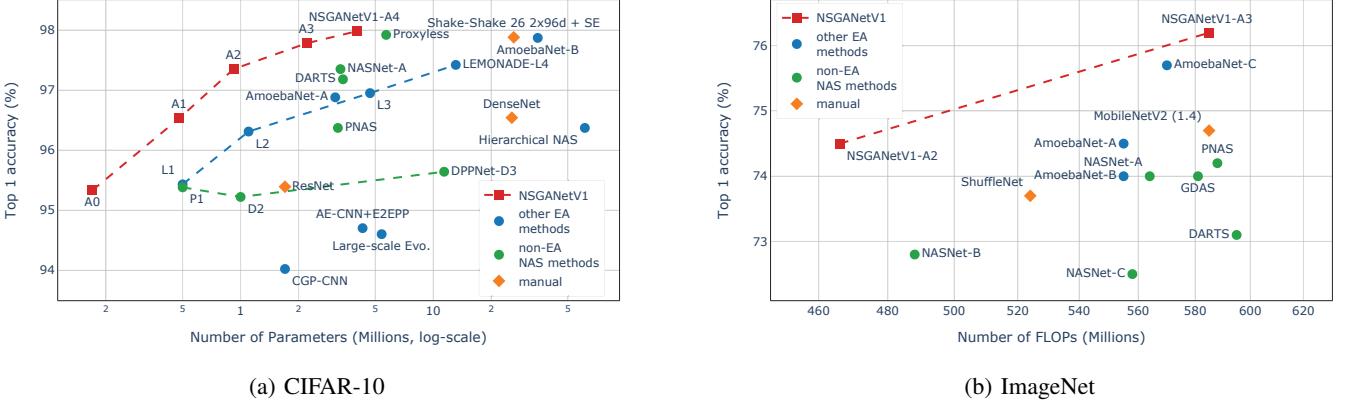


Fig. 7: Transferability of the NSGANetV1 architectures to (a) CIFAR-10, and (b) ImageNet. We compare Top-1 Accuracy vs. Computational Complexity. Architectures joined by dashed lines are from multi-objective algorithms.

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 re-training 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

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) CIFAR-100

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) ImageNet

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 29x4x64d + SE.

‡ The CIFAR-100 accuracy and #params for ENAS [44] and DARTS [9] are from [11]. #Params for AE-CNN+E2EPP are from [18].

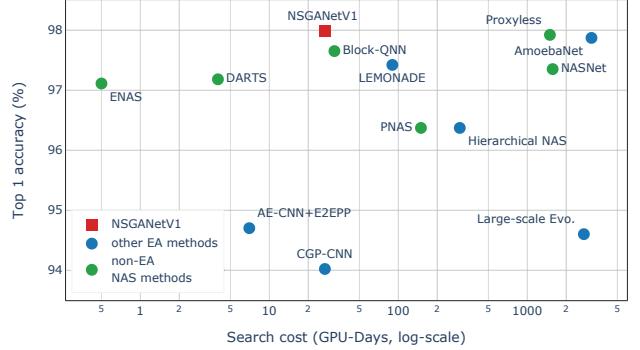
Processing Units (GPUs), *GPU-Days*, by each approach to arrive at the reported architectures. The number of GPU-Days is calculated by multiplying the number of employed GPU cards by the execution time (in units of days).

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. 8b, we observe that our proposed algorithm is more



(a) HV



(b) Search cost vs. Accuracy

Fig. 8: Search efficiency comparison between NSGANetV1 and other baselines in terms of (a) HV, and (b) the 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.

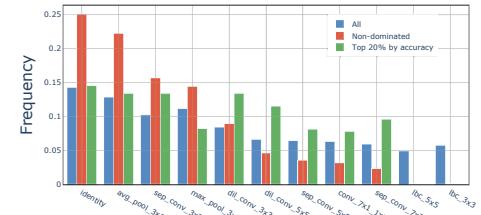
efficient at identifying a set of architectures than a number of other approaches, and the set of architectures obtained has higher performance. More specifically, NSGANetV1 simultaneously finds multiple architectures while using **10x to 100x less GPU-days** in searching than most of the considered 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 multi-objective NAS method, LEMONADE [32], NSGANetV1 manages to obtain a better (in the Pareto dominance sense) set of architectures than LEMONADE with **3x fewer GPU-Days**. Further experiments and comparisons are provided in the supplementary materials under Section VII-A.

E. Observations on Evolved Architectures

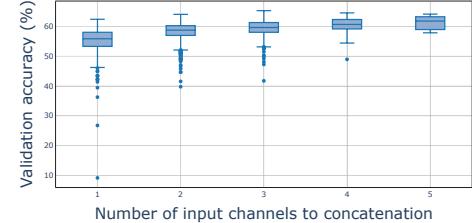
Population-based approaches with multiple conflicting objectives often lead to a set of diverse solution candidates, which can be “mined” for commonly shared design principles [50]. In order to discover any patterns for more efficient architecture design, we analyzed the entire history of architectures generated by NSGANetV1. We make the following observations:

- Non-parametric 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 non-dominated architectures are non-parametric, as shown in Fig. 9a (see also supplementary materials under Section VII-B for our follow-up study).
- Larger kernel size and parallel operations improve classification accuracy, as shown in Fig. 9a and Fig. 9b respectively. In particular, the frequencies of convolutions with large kernel sizes (e.g., *dil_conv_5x5*, *conv_7x1_1x7*) are significantly higher in the top-20% most accurate architectures than in non-dominated 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 non-dominated solutions stay as important knowledge for future applications. It is noteworthy that such a post-optimal knowledge extraction process is possible only from a multi-objective



(a) Frequency of operation choices.



(b) Concatenation dimension vs. Accuracy

Fig. 9: **Post-Search Analysis:** (a) Frequency of each operation selected during the search. (b) Effect of 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.

optimization study, another benefit that we enjoy for posing NAS as a multi-objective 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 paper—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

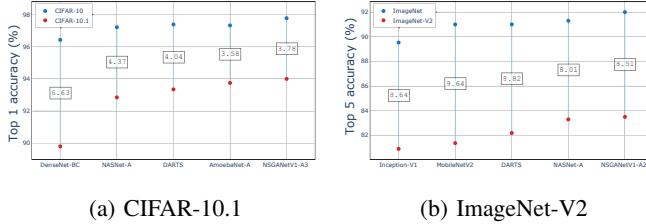


Fig. 10: Generalization: We evaluate the models on new and extended test sets for (a) CIFAR-10, and (b) ImageNet. Numbers in the boxes indicate absolute drop in accuracy (%).

these excessively re-used 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] propose 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 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 Figs. 10a and 10b 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 vs. 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.

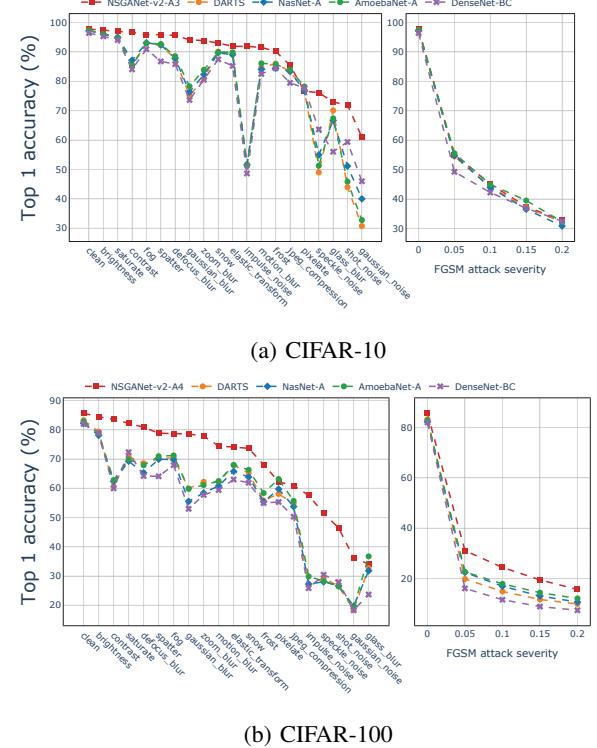


Fig. 11: Robustness: Effect of commonly observable corruptions and adversarial attacks on (a) CIFAR-10, and (b) CIFAR-100. Higher values of ϵ indicate more severe adversarial attacks. We use prediction accuracy on the corrupted test images (from each dataset) as a measurement of robustness.

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:

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

where \mathbf{x} is the original image, \mathbf{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 Figs. 11a and 11b 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 towards 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 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. 12a. Empirically, we observe that the CIFAR-100 dataset is challenging enough for architectural differences to affect predicted performance. This can be observed in Fig. 12a 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 are small-scale versions of the intended architectures, where the number of layers (N in Fig. 2a) and the number of channels (Ch_{init} in Fig. 2a) in each layer are reduced. Due to the downscaling, proxy models typically require much less compute time to train[†]. However, there exists a trade-off 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.

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

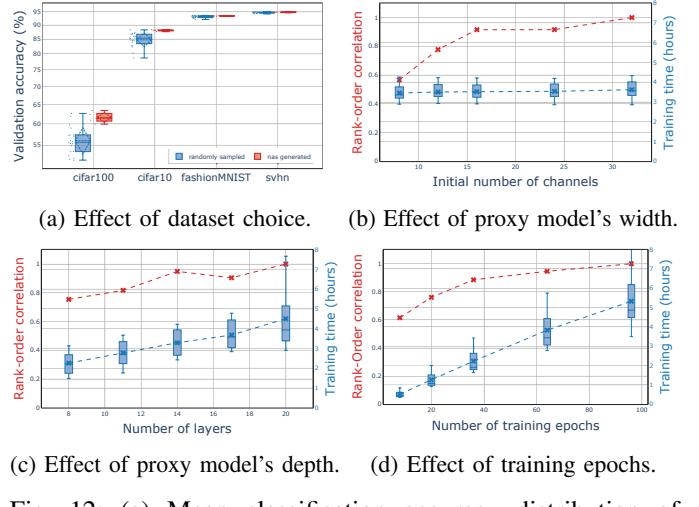


Fig. 12: (a) Mean classification accuracy distribution of randomly generated architectures and architectures from peer NAS methods on four datasets. Correlation in performance (red lines) vs. Savings in gradient descent wall time (blue boxes) by reducing (b) the number of channels in layers, (c) the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

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. Figures 12b, 12c and 12d 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 trade-offs from the plots, for our proxy model, we set the number of channels to 36 (maximum desired), number of epochs to 36, and number of layers to 14. Empirically, we found that this choice of parameters offers a good trade-off between practicality of search and reliability of proxy models.

VI. AN APPLICATION TO CHEST X-RAY CLASSIFICATION

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

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.

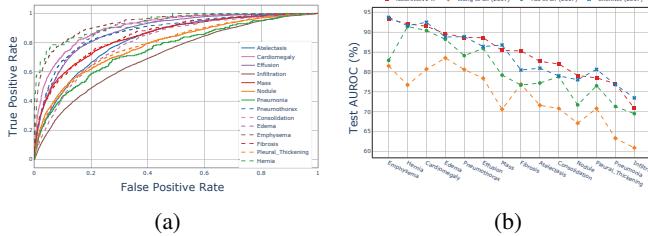


Fig. 13: (a) NSGANetV1-X multi-label classification performance on ChestX-Ray14 and (b) the class-wise mean test AUROC comparison with peer methods.

human experts. For reference purpose, we call the obtained architecture NSGANetV1-X, and we re-train 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 **3.2x 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 Figs. 13a and 13b, 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 for localizing the discriminative regions for image classification. In the examples shown in Fig. 14a - 14f, 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

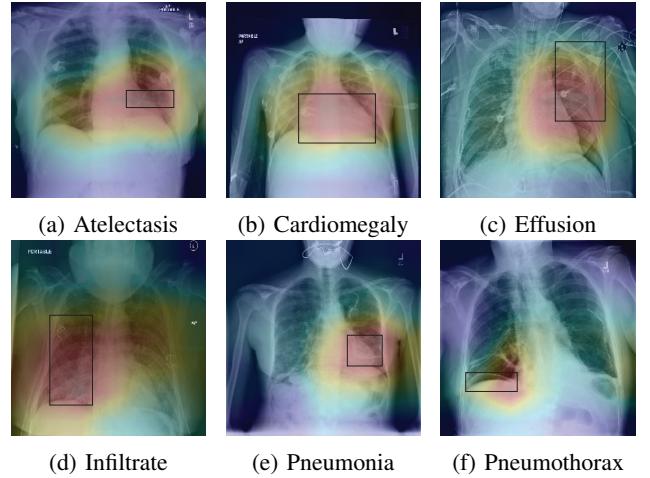


Fig. 14: Examples of class activation map [61] of NSGANetV1-X, highlighting the class-specific discriminative regions. The ground truth bounding boxes are plotted over the heatmaps.

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. CONCLUSIONS

In this paper, we have presented NSGANetV1, an evolutionary multi-objective algorithm for neural architecture search. 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 Bayesian Network model. Experiments on CIFAR-10, CIFAR-100, and ImageNet datasets have demonstrated the effectiveness of NSGANetV1. Further analysis towards 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 evolutionary algorithms for neural architecture search in achieving superior performance compared to other contemporary machine learning methods.

ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454. 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, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [4] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [6] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, “Local binary convolutional neural networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [7] B. Zoph and Q. V. Le, “Neural Architecture Search with Reinforcement Learning,” in *International Conference on Learning Representations (ICLR)*, 2016.
- [8] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [9] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in *International Conference on Learning Representations (ICLR)*, 2019.
- [10] S. Xie, H. Zheng, C. Liu, and L. Lin, “SNAS: stochastic neural architecture search,” in *International Conference on Learning Representations (ICLR)*, 2019.
- [11] X. Dong and Y. Yang, “Searching for a robust neural architecture in four GPU hours,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [12] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [13] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International Conference on Machine Learning (ICML)*, 2019.
- [14] L. Xie and A. Yuille, “Genetic CNN,” in *International Conference on Computer Vision (ICCV)*, 2017.
- [15] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, “Large-scale evolution of image classifiers,” in *International Conference on Machine Learning (ICML)*, 2017.
- [16] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search,” in *International Conference on Learning Representations (ICLR)*, 2018.
- [17] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in *AAAI Conference on Artificial Intelligence*, 2019.
- [18] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, “Completely automated CNN architecture design based on blocks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1242–1254, 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 Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 350–364, 2020.
- [20] 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,” in *Genetic and Evolutionary Computation Conference (GECCO)*, 2019.
- [21] X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
- [22] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, 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 *Genetic and Evolutionary Computation Conference (GECCO)*, 2006.
- [24] M. Suganuma, S. Shirakawa, and T. Nagao, “A genetic programming approach to designing convolutional neural network architectures,” in *Genetic and Evolutionary Computation Conference (GECCO)*, 2017.
- [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 Transactions on Cybernetics*, pp. 1–15, 2020.
- [26] Y. Jin and B. Sendhoff, “Pareto-based multiobjective machine learning: An overview and case studies,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 3, pp. 397–415, 2008.
- [27] H. Zhu and Y. Jin, “Multi-objective evolutionary federated learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1310–1322, 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 *International Conference on Machine Learning (ICML) AutoML Workshop*, 2017.
- [29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 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 *European Conference on Computer Vision (ECCV)*, 2018.
- [31] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, “Progressive neural architecture search,” in *European Conference on Computer Vision (ECCV)*, 2018.
- [32] T. Elsken, J. H. Metzen, and F. Hutter, “Efficient multi-objective neural architecture search via lamarckian evolution,” in *International Conference on Learning Representations (ICLR)*, 2019.
- [33] T. Wei, C. Wang, Y. Rui, and C. W. Chen, “Network morphism,” in *International Conference on Machine Learning (ICML)*, 2016.
- [34] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.
- [35] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, “Practical block-wise neural network architecture generation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [36] G. Eichfelder, “Multiobjective bilevel optimization,” *Mathematical Programming*, vol. 123, no. 2, pp. 419–449, 2010.
- [37] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [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 Transactions on Neural Networks*, vol. 8, no. 5, pp. 1165–1176, 1997.
- [39] B. L. Miller, D. E. Goldberg *et al.*, “Genetic algorithms, tournament selection, and the effects of noise,” *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.
- [40] K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.
- [41] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, “BOA: The bayesian optimization algorithm,” in *Genetic and Evolutionary Computation Conference (GECCO)*, 1999.
- [42] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [43] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct neural architecture search on target task and hardware,” in *International Conference on Learning Representations (ICLR)*, 2019.
- [44] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in *International Conference on Machine Learning (ICML)*, 2018.
- [45] L. Li and A. Talwalkar, “Random search and reproducibility for neural architecture search,” *arXiv preprint arXiv:1902.07638*, 2019.
- [46] S. Xie, A. Kirillov, R. Girshick, and K. He, “Exploring randomly wired neural networks for image recognition,” in *International Conference on Computer Vision (ICCV)*, 2019, pp. 1284–1293.
- [47] T. DeVries and G. W. Taylor, “Improved regularization of convolutional neural networks with cutout,” *arXiv preprint arXiv:1708.04552*, 2017.
- [48] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” in *International Conference on Learning Representations (ICLR)*, 2017.
- [49] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2017.
- [50] K. Deb and A. Srinivasan, “Innovization: Innovating design principles through optimization,” in *Genetic and Evolutionary Computation Conference (GECCO)*, 2006.
- [51] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do imagenet classifiers generalize to imagenet?” *arXiv preprint arXiv:1902.10811*, 2019.

- [52] D. Hendrycks and T. Dietterich, “Benchmarking neural network robustness to common corruptions and perturbations,” in *International Conference on Learning Representations (ICLR)*, 2019.
- [53] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [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 *Advances in Neural Information Processing Systems (NeurIPS) Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- [55] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
- [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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [57] L. Yao, E. Poblenz, D. Dagunts, B. Covington, D. Bernard, and K. Lyman, “Learning to diagnose from scratch by exploiting dependencies among labels,” *arXiv preprint arXiv:1710.10501*, 2017.
- [58] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya *et al.*, “CheXNet: Radiologist-level pneumonia detection on chest x-rays with deep learning,” *arXiv preprint arXiv:1711.05225*, 2017.
- [59] G. R. Blog, “Automl for large scale image classification and object detection,” *Google Research*, <https://research.googleblog.com/2017/11/automl-for-large-scaleimage.html>, Blog, 2017.
- [60] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, “Evolutionary neural automl for deep learning,” in *Genetic and Evolutionary Computation Conference (GECCO)*, 2019.
- [61] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [62] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf
- [63] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, “The penn treebank: annotating predicate argument structure,” in *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, 1994, pp. 114–119.
- [64] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [65] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, “PPP-Net: Platform-aware progressive search for pareto-optimal neural architectures,” in *International Conference on Learning Representations (ICLR)*, 2018.
- [66] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, “Neural architecture search with bayesian optimisation and optimal transport,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [67] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [68] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” in *International Conference on Computer Vision (ICCV)*, 2019.
- [69] K. Yu, C. Sciuco, M. Jaggi, C. Musat, and M. Salzmann, “Evaluating the search phase of neural architecture search,” in *International Conference on Learning Representations (ICLR)*, 2020.
- [70] A. Brock, T. Lim, J. Ritchie, and N. Weston, “SMASH: One-shot model architecture search through hypernetworks,” in *International Conference on Learning Representations (ICLR)*, 2018.
- [71] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, “Understanding and simplifying one-shot architecture search,” in *International Conference on Machine Learning (ICML)*, 2018.
- [72] X. He, Y. Zhou, and Z. Chen, “Evolutionary bilevel optimization based on covariance matrix adaptation,” *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 258–272, 2019.
- [73] A. Sinha, P. Malo, and K. Deb, “Evolutionary algorithm for bilevel optimization using approximations of the lower level optimal solution mapping,” *European Journal of Operational Research*, vol. 257, no. 2, pp. 395 – 411, 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0377221716306634>
- [74] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *IEEE Symposium on Security and Privacy (SP)*, 2017.
- [75] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-scale Image Recognition,” in *International Conference on Learning Representations (ICLR)*, 2015.
- [76] V. Naresh Boddeti, T. Kanade, and B. Vijaya Kumar, “Correlation filters for object alignment,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [77] A. Newell, K. Yang, and J. Deng, “Stacked hourglass networks for human pose estimation,” in *European Conference on Computer Vision (ECCV)*, 2016.
- [78] J. J. Durillo, A. J. Nebro, C. A. C. Coello, J. Garcia-Nieto, F. Luna, and E. Alba, “A study of multiobjective metaheuristics when solving parameter scalable problems,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 618–635, 2010.
- [79] H. Zille, H. Ishibuchi, S. Mostaghim, and Y. Nojima, “A framework for large-scale multiobjective optimization based on problem transformation,” *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 260–275, 2018.
- [80] W. Hong, K. Tang, A. Zhou, H. Ishibuchi, and X. Yao, “A scalable indicator-based evolutionary algorithm for large-scale multiobjective optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 525–537, 2019.

APPENDIX

In this supplementary document, we provide additional details on (1) related works in Section A; (2) multi-objective related issues in NAS in Section B; (3) bilevel optimization in Section C; (4) layer operations in Section D; (5) datasets in Section E; (6) implementation in Section F; (7) other potential utilities of our proposed algorithm in Section G; (8) hypervolume in Section H.

A. Related Work Continued

Existing NAS approaches can be broadly classified into evolutionary algorithm (EA), reinforcement learning (RL), and relaxation-based approaches – with a few additional methods falling outside these categories.

Reinforcement Learning (RL): *Q*-learning [62] is a widely-used value iteration method used for RL. The MetaQNN method [49] employs an ϵ -greedy *Q*-learning strategy with experience replay to search connections between convolution, pooling, and fully connected layers, and the operations carried out inside the layers. Zhong et al. [35] extended this idea with the BlockQNN method. BlockQNN searches the design of a computational block with the same *Q*-learning approach. The block is then repeated to construct a network, resulting in a much more general network that achieves better results than its predecessor on CIFAR-10 [34]. A policy gradient method seeks to approximate non-differentiable reward functions to train a model that requires parameter gradients, like a neural network architecture. Zoph and Le [7] first apply this method in NAS to train a recurrent neural network controller that constructs networks. The original method in [7] uses the controller to generate the entire network at once. This contrasts with its successor, NASNet [8], which designs a convolutional and pooling block that is repeated to construct a network. NASNet outperforms its predecessor and produces a network achieving state-of-the-art performance on CIFAR-10 and ImageNet.

Relaxation-based Approaches and Others: Approximating the connectivity between different layers in CNN architectures by real-valued variables weighting the importance of each layer

is the common principle of relaxation-based NAS methods. Liu et al. first implement this idea in the DARTS algorithm [9]. DARTS seeks to improve search efficiency by fixing the weights while updating the architectures, showing convergence on both CIFAR-10 and Penn Treebank [63] within one day in wall clock time on a single GPU card. Subsequent approaches in this line of research include [10]–[12], [64]. The search efficiency of these approaches stems from weight sharing during the search process. This idea is complementary to our approach and can be incorporated into NSGANetV1 as well. However, it is beyond the scope of this paper and is a topic of future study.

Methods not covered by the EA-, RL- or relaxation-based paradigms have also shown success in architecture search. Liu et al. [31] proposed a method that progressively expands networks from simple cells and only trains the best K networks that are predicted to be promising by a RNN meta-model of the encoding space. PPP-Net [65] extended this idea to use a multi-objective approach, selecting the K networks based on their Pareto-optimality when compared to other networks. Li and Talwalkar [45] show that an augmented random search approach is an effective alternative to NAS. Kandasamy et al. [66] present a Gaussian-process-based approach to optimize network architectures, viewing the process through a Bayesian optimization lens.

Multi-obj NAS through Scalarization: A portfolio of works that aims to design hardware-specific network architectures emerges. This include, ProxylessNAS [43], MnasNet [67], FBNet [12], and MobileNetV3 [68] which use a scalarized objective that encourages high accuracy and penalizes compute inefficiency at the same time, e.g., maximize $Acc * (Latency/Target)^{-0.07}$. These methods require a pre-defined preference weighting of the importance of different objectives before the search, which in itself requires a number of trials.

Weight Sharing: Another recently proposed approach for improving the search efficiency of NAS is through *weight sharing*. Approaches in this category involve training a *supernet* that contains all searchable architectures as its subnets. They can be broadly classified into two categories depending on whether the supernet training is coupled with architecture search or decoupled into a two-stage process. Approaches of the former kind [9], [43], [44] are computationally efficient but return sub-optimal models. Numerous studies [45], [46], [69] allude to weak correlation between performance at the search and final evaluation stages. Methods of the latter kind [70], [71] use performance of subnets (obtained by sampling the trained supernet) as a metric to select architectures during search. However, training a supernet beforehand for each new task is computationally prohibitive.

B. Multi-objective Optimization in NAS

In addition to high predictive accuracy, real-world applications demand NAS algorithms to simultaneously balance a few other network complexity related objectives that are specific to the deployment scenarios. For instance, mobile or embedded devices often have restrictions in terms of model

size, multiply-adds, latency, power consumption, and memory footprint.

It has been a common observation in the Deep Learning literature that classification performance is positively correlated with the complexity of the neural network. Since we want to maximize one (performance) while minimizing the other (FLOPS), they constitute a conflicting scenario. Optimization of a single composite objective obtained by weighting two objectives into one will produce a neural architecture and weight combination which may be too complex (requiring more FLOPS) or too inaccurate (having less accuracy). A generative approach of simply applying a single-obj optimization does not solve the issue: (i) many common scalarization methods do not work if the interesting optimal solutions lie on the non-convex part of the efficient frontier, and (ii) Generative methods are more computationally expensive (due to the lack of any parallel search efforts) than simultaneous methods, such as the method used in this paper.

In particular, ResNet [2] showed the classification accuracy on ImageNet continuously to improve as the number of layers increases from 18 (2G FLOPs) to 152 (11G FLOPs). Similar trends are also observed in DenseNet [3], NASNet [8], EfficientNet [13], etc. The aforementioned observation implies the competing nature of these objectives of simultaneously maximizing classification performance and minimizing complexity in terms of FLOPs. Additionally, posing NAS as a multi-objective problem is beneficial from the decision-making perspective, as it allows designers to choose a suitable network architecture *a posteriori* as opposed to requiring a pre-defined preference weighting of each objective prior to the search. Empirically, we also observe that the type of diversity provided by multi-objective optimization contributes to its outperforming on the classification accuracy objective achieved relative to single-objective optimization. (This can be seen, for example, in comparing NSGANetV1-Ax from the main paper and NSGANetV1-Bx from Table V in this supplementary materials.

C. Bilevel Optimization in NAS

Recall that we formulate the problem of designing custom architectures for different deployment scenarios as a bilevel multi-objective NAS problem, mathematically as below:

$$\begin{aligned} & \text{minimize } \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}; \mathbf{w}^*(\mathbf{x})), f_2(\mathbf{x}))^T, \\ & \text{subject to } \mathbf{w}^*(\mathbf{x}) \in \operatorname{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \mathbf{x}), \\ & \quad \mathbf{x} \in \Omega_x, \quad \mathbf{w} \in \Omega_w, \end{aligned} \quad (3)$$

The bilevel formulation used above arises from the problem nature of NAS, where one objective function f_1 evaluation at the upper-level requires both an architecture \mathbf{x} and its weights \mathbf{w} . f_1 is *not* meaningfully defined at any arbitrary \mathbf{w} ; rather, it requires \mathbf{w} to be a member of the set that minimize the cross-entropy loss \mathcal{L} (in the case of image classification) on training data given \mathbf{x} , mathematically as $\mathbf{w}^*(\mathbf{x}) \in \operatorname{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \mathbf{x})$. For simplicity, we use $\mathbf{w}^*(\mathbf{x})$ to denote weights that satisfy the previously specified condition. However, $\mathbf{w}^*(\mathbf{x})$ is typically not analytically computable due to non-linearities in layers and

from activation functions in encoded by \boldsymbol{x} , requiring another (lower) level of optimization.

The principle of a bilevel optimization is that the upper-level objectives and constraints must be computed using the optimal lower level variables ($\boldsymbol{w}^*(\boldsymbol{x})$) for the corresponding upper-level variables (\boldsymbol{x}). Thus, in an implicit manner, f_1 becomes a function of \boldsymbol{x} alone, making $f_1(\boldsymbol{x}; \boldsymbol{w}^*(\boldsymbol{x}))$. Using a (1, 1) evolution strategy search as an illustration, when $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \Delta\boldsymbol{x}$ is altered at iteration t in the upper-level, we first optimize the corresponding lower-level problem of $\mathcal{L}(\boldsymbol{w}; \boldsymbol{x}_{t+1})$ to find $\boldsymbol{w}^*(\boldsymbol{x}_{t+1})$ (through SGD). We then compute $f_1(\boldsymbol{x}_{t+1}, \boldsymbol{w}^*(\boldsymbol{x}_{t+1}))$, and update the current $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t+1}$ if $f_1(\boldsymbol{x}_{t+1}, \boldsymbol{w}^*(\boldsymbol{x}_{t+1}))$ is better than $f_1(\boldsymbol{x}_t, \boldsymbol{w}^*(\boldsymbol{x}_t))$, i.e., $f_1(\boldsymbol{x}_{t+1}, \boldsymbol{w}^*(\boldsymbol{x}_{t+1})) < f_1(\boldsymbol{x}_t, \boldsymbol{w}^*(\boldsymbol{x}_t))$. A bilevel problem allows a more convenient way to optimize a hierarchical problem having two distinct hierarchical variable sets (such as, a weight vector only makes sense when an architecture is provided) than a single level in which both \boldsymbol{x} and \boldsymbol{w} are considered in the same level. First, the search space becomes huge and second, a good \boldsymbol{x} may be deleted simply because the respective \boldsymbol{w} is not good.

Our NSGANetV1 algorithm has two types of optimization problems put together:

- 1) A bilevel optimization having a upper-level optimization problems in which architectures (\boldsymbol{x}) are decision variables and a lower level problem in which weight vector (\boldsymbol{w}) for the given architecture is the decision variable.
- 2) upper-level problem uses two conflicting objectives providing a Pareto-optimal front and the lower level problem uses a single objective of minimizing the cross-entropy loss on the validation data.

Thus, the final outcome of our approach is a set of trade-off architectures and their associated weight vectors, thereby completely specifying neural networks. For upper-level, we employ a customized and advanced NSGA-II-like evol. multi-objective algorithm, so that a set of non-dominated trade-off solutions is obtained at each iteration. The lower level uses the stochastic gradient based back-propagation algorithm for weight learning. Despite the fact that our approach also hybridizes a global search (EMO algorithm at the upper-level) and a local search (SGD) into a unified paradigm, which is also done in memetic algorithms, our bilevel approach is conceptually very different from memetic computing. To be more specific, the local search used in memetic algorithms is mainly to improve an individual's fitness within its local neighborhood, while the local search (SGD) used in our approach is to find the remaining set of variables (lower-level variables; weights), which jointly with the upper-level variables (architectures) compute the objective function (classification error).

Our bilevel approach is nested in nature, meaning that for each \boldsymbol{x} at the upper-level, a respective optimized \boldsymbol{w}^* is found by using the back-propagation method. However, the NAS at the upper-level is expedited by using a Bayesian learning method of already found good solutions and by using customized coding and genetic operators. Although more sophisticated surrogate-assisted bilevel algorithms, such as BLEAQ or BLEAQ2 [72], [73] can be used, in this work we keep the methods relatively

simple and use learning-assisted EMO and use only 1,200 architecture evaluations to achieve the results.

D. Details of the Considered Layer Operations

As described in Section III-A, we form a operation pool consists of 12 different choices of convolution, pooling and etc., based on their prevalence in the CNN literature. Most of these operations can be directly called from standard Deep Learning libraries, like Pytorch, TensorFLow, Caffe, etc. Here we provide demo Pytorch codes for less commonly used[‡] operations, including *depth-wise separable convolutions*, *local binary convolutions* and *1x7 then 7x1 convolution*.

E. Datasets Details

Examples from CIFAR-10, CIFAR-100, and ImageNet are provided in Fig. 16.

1) *CIFAR-10.1 and ImageNet-V2*: In this work, we use the *MatchedFrequency* version of the ImageNet-V2 dataset. The curation details along with the discussion of the difference among the three versions are available in [51]. Examples randomly sampled from these two new testing sets are provided in Figs. 17a and 17b, repectively. The CIFAR-10.1 is available for download at <https://github.com/modestyachts/CIFAR-10.1>. And the ImageNet-V2 is available at <https://github.com/modestyachts/ImageNetV2>.

2) *CIFAR-10-C and CIFAR-100-C*: There are in total 19 different commonly observable corruption types considered in both CIFAR-10-C and CIFAR-100-C, including Gaussian noise, shot noise, impulse noise, de-focus blur, frosted glass blur, motion blur, zoom blur, snow, frost, fog, brightness, contrast, elastic, pixelate, jpeg, speckle noise, Gaussian blur, spatter and saturate. Fig. 18a provides examples for visualization. For every corruption type, there are five different levels of severity, see Fig. 18b for visualization. Both datasets are available from the original authors' GitHub page at <https://github.com/hendrycks/robustness>. A demo visualization of adversarial examples created by applying FGSM [53] on MNIST dataset is provided in Fig. 18c.

3) *ChestX-Ray14*: ChestX-Ray 14 are hospital-scale Chest X-ray database containing 112,120 frontal-view X-ray images of size 1,024 x 1,024 pixels from 30,805 unique patients. The database is labeled using natural language processing techniques from the associated radiological reports stored in hospitals' Picture Archiving and Communication Systems (PACS). Each image can have one or multiple common thoracic diseases, or "Normal" otherwise. Visualization of example X-ray images from the database is provided in Fig. 19. The dataset is publicly available from NIH at <https://nihcc.app.box.com/v/ChestXray-NIHCC>. We follow the *train_val_list.txt* and *test_list.txt* provided along with the X-ray images to split the database for training, validation and testing.

[‡]refer to both less frequently used operations and operations under less commonly followed setups.

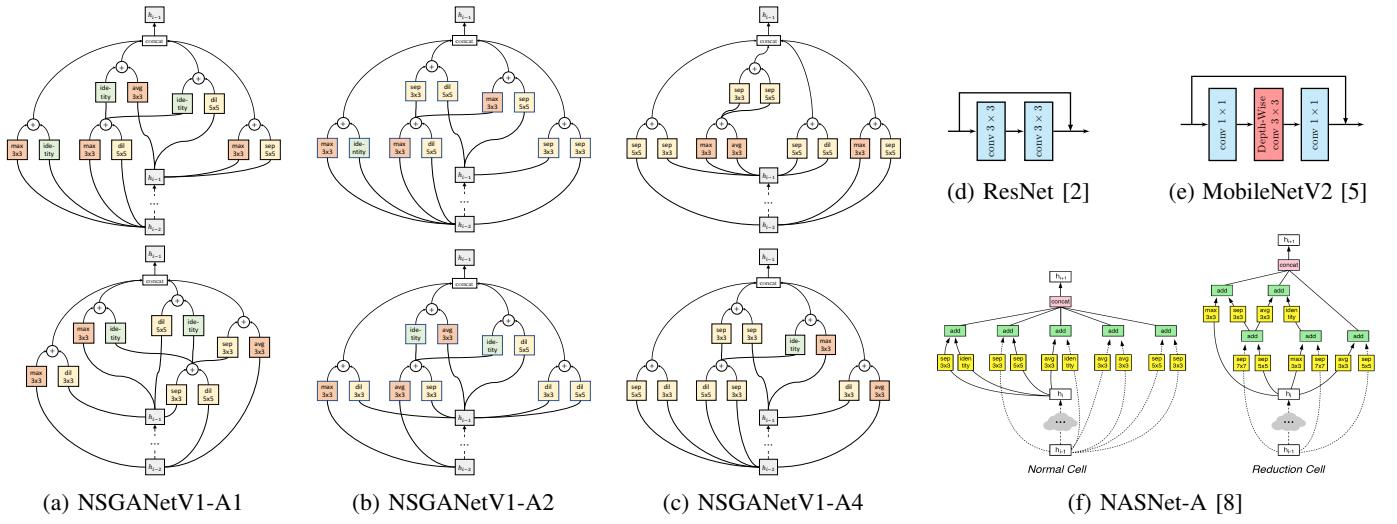
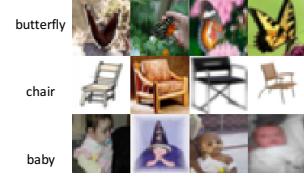


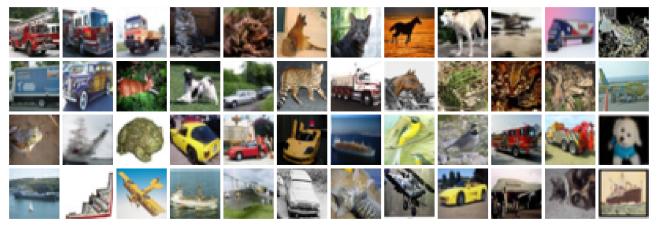
Fig. 15: Visualization of block-level structures for different architectures. The Normal and Reduction blocks are shown in the first and second rows, respectively for NSGANetV1 architectures. Examples of blocks that are designed manually by experts [2], [5] and from other peer methods [8] are also included in (d) - (f) for comparison.



(a) CIFAR-10



(b) CIFAR-100



(c) ImageNet

Fig. 16: Examples from CIFAR-10, CIFAR-100 and ImageNet datasets. Images in each row belong to the same class with label names shown to the left.

F. Implementation Details Continued

Evaluating a neural network architecture’s performance is computationally expensive—e.g., one evaluation on the CIFAR-10 dataset takes more than 30 minutes. In general, our (GA-related) hyper-parameter choices represent the minimal number of function evaluations required to reproduce the claimed performance. To be more specific, in our proposed algorithm, each architecture is encoded with a 40-position, integer-valued string. Our choice of population size at 40 corresponds to one individual per variable dimension, which follows one of the common suggestions in the GA literature on minimal required population size. Empirically, we observed that the hypervolume stabilized by generation 30 (see Fig.9a in the revised main paper), hence, we chose to terminate the proposed algorithm at generation 30. Other hyper-parameter choices are discussed in Section V.C of the revised main paper.

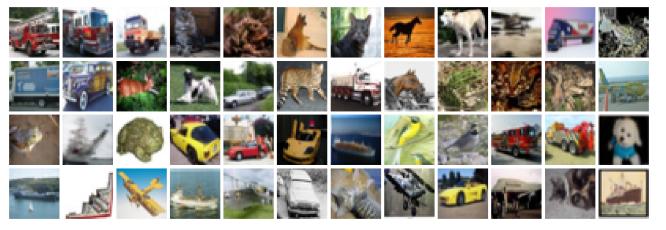


Fig. 17: Visualization of CIFAR-10.1 (a) and ImageNet-V2 (b). Examples are randomly sampled from the datasets.

G. Follow-up Studies

1) *Single-Objective NSGANetV1*: Despite the superior effectiveness and efficiency of the proposed algorithm, the computation overheads of 27 GPU-days of NSGANetV1 can be infeasible for users with few GPU cards. Towards understanding of the overall search wall time limit of NSGANetV1, as well as comparison to the peer methods that use less GPU-days to execute the search, the following experiment has been performed. We minimized the search setup differences by dropping the second objective of minimizing FLOPs and changing the search dataset to CIFAR-10. We also reduce the population size by half and perform early-termination at one and four GPU-days. The obtained architectures are named as NSGANetV1-B0 and NSGANetV1-B1, respectively.

TABLE IV: Demo Pytorch implementation of separable convolution (a), local binary convolution (b) and 1x7 then 7x1 convolution (c) used in NSGANetV1.

```
class SepConv(nn.Module)
# depth-wise separable convolution in NSGANetV1
# consists of two regular depth-wise separable convolutions in series.
def __init__(self, C_in, C_out, kernel_size, stride, padding, affine=True)
    super(SepConv, self).__init__()
    self.op = nn.Sequential(
        nn.ReLU(inplace=False),
        nn.Conv2d(C_in, C_in, kernel_size=kernel_size, stride=stride,
                  padding=padding, groups=C_in, bias=False),
        nn.Conv2d(C_in, C_in, kernel_size=1, padding=0, bias=False),
        nn.BatchNorm2d(C_in, affine=affine),
        nn.ReLU(inplace=False),
        nn.Conv2d(C_in, C_in, kernel_size=kernel_size, stride=1,
                  padding=padding, groups=C_in, bias=False),
        nn.Conv2d(C_in, C_out, kernel_size=1, padding=0, bias=False),
        nn.BatchNorm2d(C_out, affine=affine),
    )
    def forward(self, x)
        return self.op(x)
```

(a) Separable Convolution

```
# The weight values of local binary convolution filters
# either -1, 1, or 0, and kept fixed during back-propagation.
# Number of 0-valued weights are controlled by sparsity argument.
def LBCConv(in_planes, out_planes, kernel_size=3, stride=1,
            padding=1, dilation=1, groups=1, bias=False, sparsity=0.5)
    conv2d = nn.Conv2d(
        in_planes, out_planes, kernel_size=kernel_size,
        stride=stride, padding=padding, dilation=dilation,
        groups=groups, bias=bias,
    )
    conv2d.weight.requires_grad = False
    conv2d.weight.fill_(0.0)
    num = conv2d.weight.numel()
    shape = conv2d.weight.size()
    index = torch.Tensor(math.floor(sparsity * num)).random_(num).int()
    conv2d.weight.resize_(num)
    for i in range(index.numel()):
        conv2d.weight[index[i]] = torch.bernoulli(torch.Tensor([0.5])) * 2 - 1
    conv2d.weight.resize_(shape)
    return conv2d
```

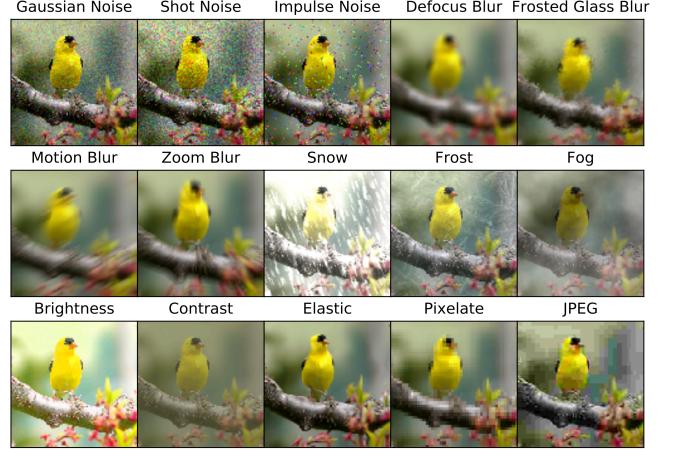
(b) Local Binary Convolution

```
class Conv1x7Then7x1
def __init__(self, C, stride, affine=True)
    super(Conv1x7Then7x1, self).__init__()
    self.op = nn.Sequential(
        nn.ReLU(inplace=False),
        nn.Conv2d(C, C, (1, 7), stride=(1, stride), padding=(0, 3), bias=False),
        nn.Conv2d(C, C, (7, 1), stride=(stride, 1), padding=(3, 0), bias=False),
        nn.BatchNorm2d(C, affine=affine)
    )
    def forward(self, x)
        return self.op(x)
```

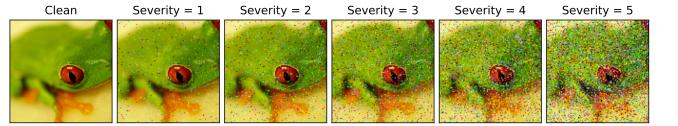
(c) 1x7 convolution then 7x1 convolution

TABLE V: NSGANetV1 with single objective of maximizing classification accuracy on CIFAR-10 and early terminations.

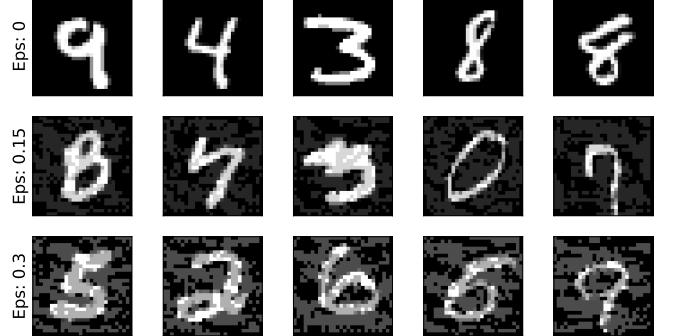
Method	Type	#Params	Top-1 Acc.	GPU-Days
Genetic CNN [14]	EA	-	92.90%	17
AE-CNN+E2EPP [19]	EA	4.3M	94.70%	7
ENAS [44]	RL	4.6M	97.11%	0.5
DARTS [9]	differential	3.3M	97.24%	1
NSGANetV1-B0	EA	3.3M	96.15%	1
NSGANetV1-B1	EA	3.3M	97.25%	4



(a) Types of corruptions



(b) Severity of corruptions



(c) Adversarial examples from FGSM [53] on MNIST.

Fig. 18: Visualization of different types of corruptions and different levels of severity. Examples are from [52]. Both CIFAR-10-C and CIFAR-100-C are constructed by applying corruptions to the original testing sets. A demo visualization of adversarial examples from FGSM on MNIST is shown in (c).

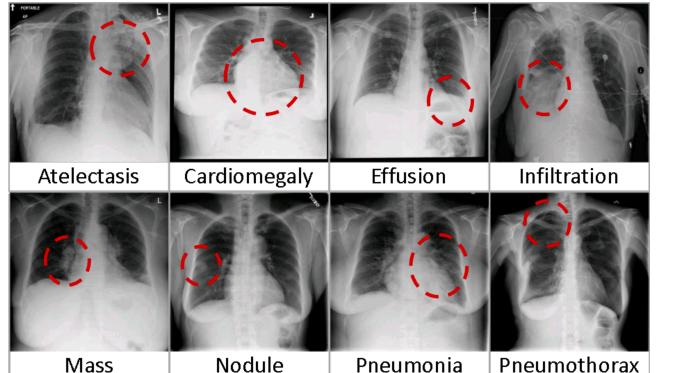


Fig. 19: Visualization of ChestXray14 datasets. Examples showing eight common thoracic diseases are from [56].

Results in Table V confirm that our proposed algorithm can be more efficient in GPU-days than the other two EA-based peer methods, Genetic CNN [14] and AE-CNN-E2EPP [19]. Specifically, NSGANetV1 obtains the architecture B1 in 3 less GPU-days than AE-CNN-E2EPP, in addition to the B1 architecture being more accurate in CIFAR-10 classification and less complex in number of parameters. Due to the use of weight sharing that partially eliminates the back-propagation weight learning process, ENAS [44] and DARTS [9] are still more efficient in GPU-days than our proposed method. The weight sharing method could in principle be applied to NSGANetV1 as well, however this is beyond the scope of this paper.

2) *Effectiveness of Non-learnable Operations:* Our post-optimization analysis on the evolved architectures, shown in Section IV-E, has revealed some interesting findings, one of which being the effectiveness of non-parametric operations, e.g. identity mapping, average/max pooling, etc., in trading off classification performance for architectural complexity. To further validate this observation, we consider a expanded range of operations including both non-parametric and weight-fixed operations, which we name as *non-learnable operations* in this paper. We manually construct such layers by concatenate multiple non-learnable operations in parallel. The obtained results are shown in Figs. 20a - 20c.

Our preliminary results on manual construction of non-learnable layers are very encouraging. In additional to the comparative performance to regular fully learned layers, non-learnable layers offer unique advantages in terms of re-usable weights for multi-tasking network architectures, as the weights are agnostic (not specifically learned on a particular task). We believe designing dedicated search algorithm to shape the construction of these non-learnable layers is a promising direction for NAS towards automatic design for multi-tasking architectures.

3) *Robustness Against Adversarial Attacks:* Based on our analysis in Section V-B, years of architectural advancements have translated to minuscule improvements in robustness against adversarial examples. Simple one-iteration attack strategy like FGSM [53] is enough to constructing examples that turn many modern DNN classifiers to random-guess (see Fig. 12 for examples). In this section, we make an effort to improve adversarial robustness from the architectural perspective. The search space used in the main paper searches over both layer operations and layer connections (see Section III-A). To isolate the effect of these two aspects to the adversarial robustness, we fix the layer operation to basic residual module [2] and search over the connections among these modules to improve both classification accuracy on clean images and robustness against adversarial examples.

Designing a measure/objective for robustness against adversarial robustness is an area of active research (e.g., [74]). For our purposes, we present a possible measure here, illustrated in Fig. 21. Using the FSGM presented by [53], this robustness objective progressively increases noise produced by FSGM. The ϵ axis in Fig. 21 refers to the hyper-parameter in the FSGM equation,

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

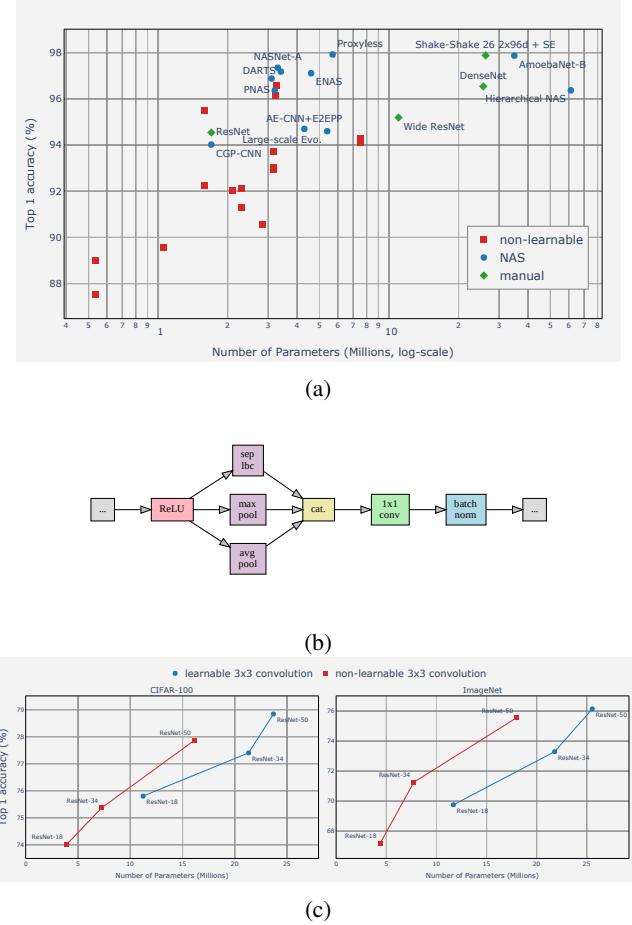


Fig. 20: Preliminary experiment on constructing DNN architectures using layers with non-learnable weights. Each layer is composed of several non-learnable operations in parallel. We manually constructed a handful of such layers and evaluate them on CIFAR-10 (a). An example configuration, based on the trade-off between accuracy and the number of the parameters, is shown in (b). We validate the effectiveness of non-learnable layers by replacing the original 3×3 convolution in different ResNet models with the chosen configuration on both CIFAR-100 and ImageNet (c). Evidently, layer with non-learnable weights is capable of yielding competitive classification performance while being computational efficient as opposed to conventional learnable convolutions.

where \mathbf{x} is the original image, \mathbf{x}' is adversarial image, y_{true} is true class label, and \mathcal{L} cross-entropy loss. Therefore, for this experiment, we seek to maximize two objectives, namely, classification accuracy and the robustness objective defined above.

The setup for the robustness experiment is as follows. For training we use 40,000 CIFAR-10 images from the official CIFAR-10 training data, 10,000 of which are reserved for validation. Each network is encoded with three blocks using the macro space encoding from our previous work [20]. In each phase a maximum of size nodes may be active—where the computation at each node is 3×3 convolution followed by ReLU and batch normalization. Each network is trained for 20 epochs with SGD on a cosine annealed learning rate

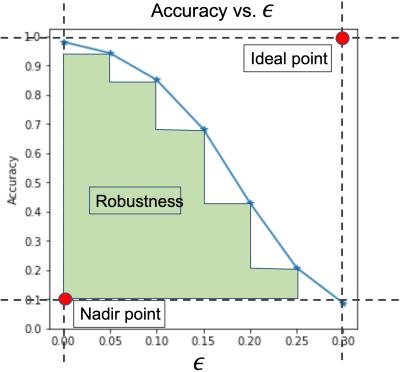


Fig. 21: **Robustness Objective:** We define a robustness objective under the FGSM [53] attack as follows: 1) obtain classification accuracy on adversarial images generated by FGSM as we vary ϵ , 2) compute the area under the curve (blue line), approximated by the area of the green region; 3) normalize the robustness value to the rectangular area formed by the *Ideal point* and *Nadir point*; 3) Ideal point is defined at 100% accuracy at pre-defined maximum ϵ value, and the nadir point is defined as the accuracy of random guessing at $\epsilon = 0$ (clean images).

schedule. The epsilon values used in the FSGM robustness calculation are $[0.0, 0.01, 0.03, 0.05, 0.07, 0.1, 0.15]$. As before, NSGANetV1 initiates the search with 40 randomly created network architecture, and 40 new network architectures are created at each generation (iteration) via genetic operations (see main paper for details). The search is terminated at 30 generations.

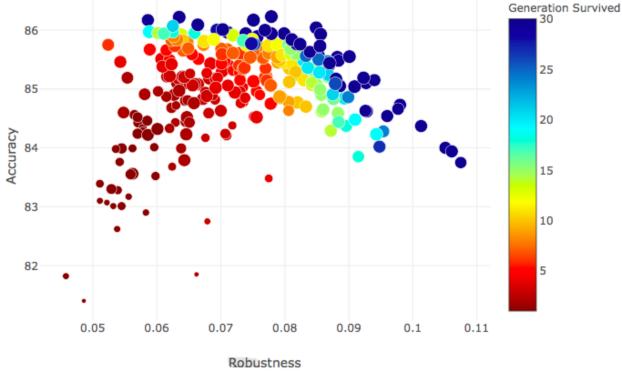


Fig. 22: Trade-off frontier of the robustness experiments. Color indicates the generation (iteration) at which a network architecture is eliminated from the surviving parent population. The size of each point is proportional to the network architecture's number of trainable parameters. We note that networks for latter generations form the Pareto front (dark blue points).

Empirically, we observe a clear trade-off between accuracy and robustness, as shown in Fig. 22. Visualization of the non-dominated architectures are provided in Fig. 23c. In our opinion, NSGANetV1 is useful in capturing patterns that differentiate architectures that are good for competing objectives. We find that the “wide” networks (like ResNeXt [42] or Inception blocks [1]) appear to provide good accuracy on standard benchmark

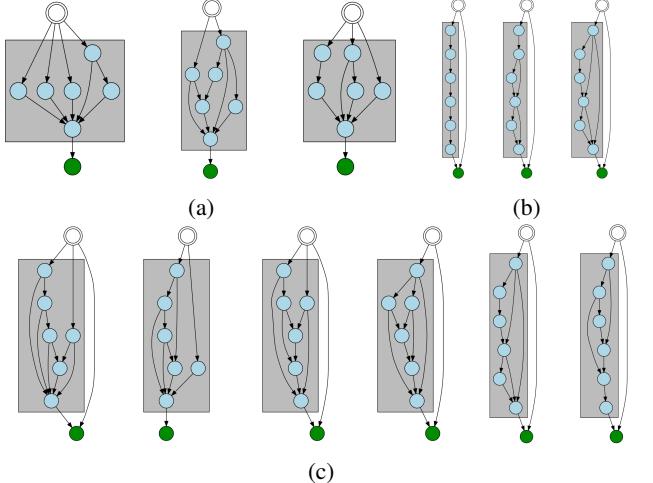


Fig. 23: (a) Examples of the computational blocks discovered with high classification accuracy. For these networks, the mean accuracy and robustness objectives are 0.8543 and 0.0535, respectively; (b) Examples of the computational blocks discovered with high robustness against FGSM attack, the mean accuracy and robustness objectives are 0.8415 and 0.1036, respectively; (c) Examples of the computational blocks discovered along the pareto-front that provides an efficient trade-off between classification accuracy and adversarial robustness. They are arranged in the order of descending accuracy and ascending robustness.

images, but are fragile to the FSGM attack. On the other hand, “deep” networks (akin to ResNet [2] or VGG [75]) are more robust to FSGM attack, while having less accuracy. This phenomenon is illustrated with examples in Figs. 23a and 23b, respectively. Furthermore, the skip connection of skipping the entire block’s computation appears to be critical in obtaining a network that is robust to adversarial attacks; see Fig. 24a and 24b.

4) An Application to Multi-view Car Alignment: In addition to object classification, dense image prediction (e.g. object alignment, human body pose estimation and semantic segmentation, etc.) is another class of problems that is of great importance to computer vision. Dense image prediction assigns a class label to each pixel in the query images, as opposed to one label to the entire image in case of classification. In this section, we apply NSGANetV1 to the problem of multi-view car key-points alignment.

We use the CMU-Car dataset originally introduced in [76]. The dataset contains around 10,000 car images in different orientations, environments, and occlusion situations. In this case, we search for the path of image resolution changes, similar to [64]. The node-level structure is kept fixed, using the basic residual unit [2]. The performance of architectures in this case is calculated using the root mean square (RMS) error between the predicted heatmap and ground truth for each key-point, more details are available in [76]. We use FLOPs as the second objective for architecture complexity measurement. The obtained architectures are named as NSGANetV1-C0 and -C1. The obtained results are provided in Table VI and the

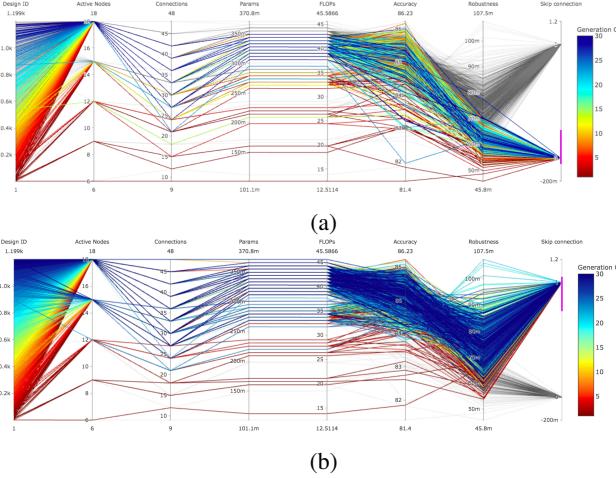


Fig. 24: Parallel coordinate plots of the 1,200 network architectures sampled by NSGANetV1. Each line represents a network architecture, each vertical line is an attribute associated with the network. (a) Networks that have the skip connection bit inactive, we can see that none of them have good measurement on robustness against adversarial attacks. (b) Networks that have the skip connection bit active. This skip connection bit refers to the connection that goes past all computation within a phase, as a normal residual connection would. When the skip connection is active, the networks cover the full range of adversarial robustness.

visualization of the architectures is provided in Fig. 25.

TABLE VI: Preliminary results on the CMU-Car alignment [76]. Notably, our proposed algorithm is able to find architectures with competitive performance while having 2x less parameters when compared to human-designed architecture [77].

Architectures	Params. (M)	FLOPs (M)	Regression Error (%)
Hourglass[77]	3.38	3613	7.80
NSGANetV1-C0	1.53	2584	8.66
NSGANetV1-C1	1.61	2663	8.64

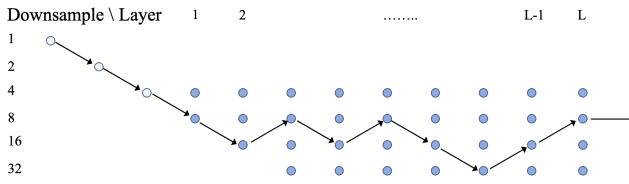


Fig. 25: Spatial-resolution-change path of the NSGANetV1-C0 architecture. Each circle encodes a residual module [2]. Circles colored in white are always executed at the beginning. The arrows and blues circles are parts of the search decisions. The total number of layers (L) are set to be nine.

5) *Ablation Study on Exploitation Operator:* Recall that we use Bayesian Network (BN) as a probabilistic model to estimate the distribution of the Pareto set (of architectures). In this section, we first explain the connection of our proposed

BN-based distribution estimation operator to the existing works [78]–[80] of large-scale multi-objective optimization algorithms for general numerical problems. The common theme behind these works is to learn the correlation among decision variables to reduce the dimension either through grouping (optimize a subset of variables at a time) or embedding (projection to lower-dimensional space). In our work, we exploit the problem information (i.e., network architectures are variants of directed acyclic graphs) explicitly in the form of a BN to learn the correlations (i.e., BN edge weights) among architectural variables. The learned BN is then used (as a probabilistic model) to generate the remaining variables given the observed variables, as a form of dimension reduction. Thus, in this work, we take advantage of learning algorithms to capture the properties of good solutions to deal with the large dimensionality of the problem. In short, our approach in NAS application and general-purpose EMO algorithms shares a similar concept of dimensional reduction to handle large-scale problems.

Secondly, we study the effectiveness of the proposed BN-based exploitation operator. The experimental setup follows a two objective NAS optimization to maximize top-1 validation accuracy on the FashionMNIST dataset [55] and minimize #FLOPs simultaneously. We study five different settings of the proposed exploitation operator, namely:

- 1) No exploitation
- 2) Exploitation activate after 1/3 computation budget spent.
- 3) Exploitation activate after 1/2 computation budget spent.
- 4) Exploitation activate after 2/3 computation budget spent.
- 5) Exploitation activate after 3/4 computation budget spent.

We use the same population size of 40 and a maximum number of 30 generations for each of the considered settings. And we repeat 11 runs with different random seeds to capture the variance from different initial population. The obtained results are provided in Fig. 26.

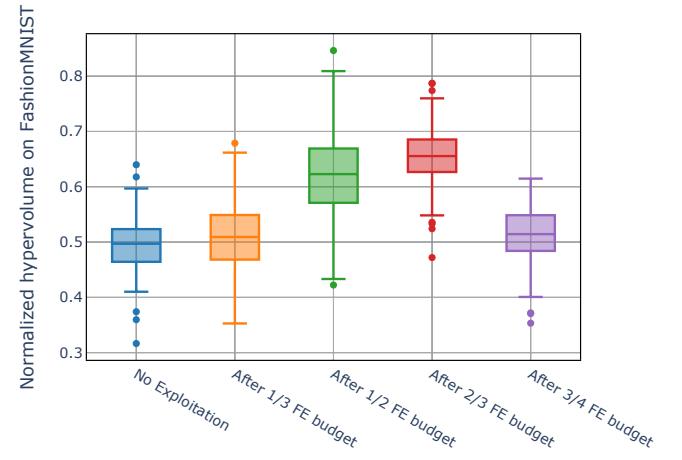


Fig. 26: Ablation study on effectiveness of our proposed exploitation operator under different settings.

Empirically, we observe that our proposed exploitation operator provides a noticeable improvement to the overall algorithm's performance, measured by hypervolume. However, the margin of improvement quickly diminishes as we activate

the operator too early (i.e. before 1/3 of the computation budget spent) or too close to the total budget (i.e. after 3/4 of the computation budget spent).

H. Hypervolume Calculation

For the two-objective experiments presented in Section IV of the main paper, the reference point used in computing the hypervolume metric is (100, 1,000), where 100 is the worst error rate in percentage, and 1,000 is the highest #FLOPs, in millions, of any architecture that our search space can encode. We then normalize the hypervolume by the rectangular area formed by the reference point and the ideal point—i.e. (0, 0).