Multi-Criterion Evolutionary Design of Deep Convolutional Neural Networks

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

***Abstract*—Convolutional neural networks (CNNs) are the backbones of deep learning paradigms for numerous vision tasks. Early advancements in CNN architectures are primarily driven by human expertise and elaborate design. 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 under limited computational budget for two reasons: (1) the obtained architectures are either solely optimized for classification performance or only for one targeted resource requirement; (2) the search process requires vast computational resources in most approaches. To overcome this limitation, we propose an evolutionary algorithm for searching neural architectures under multiple objectives, such as classification performance and 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 the computation efficiency by carefully down-scaling the architectures during the search as well as reinforcing the patterns commonly shared among the past successful architectures through Bayesian Learning. The integration of these two main contributions allows an efficient design of architectures that are competitive and in many cases outperform both manually and automatically designed architectures on benchmark image classification datasets, CIFAR, ImageNet and human chest X-ray. The flexibility provided from simultaneously obtaining multiple architecture choices for different compute requirements further differentiates our approach from other methods in the literature.**

***Index Terms*—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. These developments are the fruits of years of painstaking efforts and human ingenuity.

The authors are with Michigan State University, East Lansing, MI, 48824 USA, E-mail: (luzhicha@msu.edu).

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 indicate 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 component of the entire Deep Neural Network (DNN) development process, and is a fundamental step towards automated machine learning.

In general, the problem of designing CNN architectures for a target dataset $\mathcal{D} = \{\mathcal{D}_{trn}, \mathcal{D}_{vld}, \mathcal{D}_{tst}\}$ can be viewed as a bi-level optimization problem [7]. It can be mathematically formulated as,

$$\begin{aligned} \underset{\alpha \in \mathcal{A}}{\text{minimize}} \quad & \mathcal{L}_{vld}(\alpha, \omega^*), \\ \text{subject to} \quad & \omega^* \in \underset{\omega \in \Omega}{\operatorname{argmin}}\, \mathcal{L}_{trn}(\omega, \alpha), \end{aligned} \tag{1}$$

where the upper-level variable α defines CNN architectures, and the lower-level variable ω defines the associated weights. $\mathcal{L}_{trn}$ and $\mathcal{L}_{vld}$ denote the losses on the training data $\mathcal{D}_{trn}$ and the validation data $\mathcal{D}_{vld}$, respectively. Note that the testing data $\mathcal{D}_{tst}$ is used only for the purpose of reporting final results.

The problem in Eq. (1) poses several challenges to conventional optimization methods. First, the upper-level problem is not differentiable, as the gradient of the validation loss $\mathcal{L}_{trn}$ cannot be reliably computed or even approximated due to the categorical nature of the upper-level variable α involving the choice of layer operations, activation functions, etc. Secondly, evaluating the validation loss for a given architecture involves another prolonged optimization process, as the relationship between an architecture α and its corresponding optimal weights ω^* cannot be analytically computed due to the non-linear nature of modern CNNs.

Early methods for NAS relied on Reinforcement Learning (RL) to navigate and search for architectures with high performance. A major limitation of these approaches [8], [9] is the steep computational requirement for the search process itself, often requiring weeks of wall clock time on hundreds of GPU cards. Recent *relaxation*-based methods [10]–[13] 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 the need for high-performance results from NAS models, real-world applications of these models demand finding network architectures with different complexities for different deployment scenarios—e.g., IoT systems, mobile devices, automotives, cloud servers, etc. These computing devices are often constrained by a variety of hardware resources, such as power consumption, available memory, and latency constraints, to name a few. Even though there are methods with more advanced techniques like weight sharing [14] to improve RL's search efficiency, and binary gating [15] to reduce the GPU memory footprint of relaxation-based NAS methods, most existing RL and relaxation-based methods are not readily applicable for multi-objective NAS.

Evolutionary algorithms (EAs), due to their population-based nature and flexibility in encoding, 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 [16]–[21] are still single-criteria driven. Even under the explosive growth of general interest in NAS, EA-based NAS approaches have not been well perceived outside the EA community, primarily due to the following two reasons: (i) existing EA-based methods [18], [19] that produce competitive results are extremely computationally inefficient (e.g., one run of [19] takes 7 days on 450 GPUs); or (ii) results from existing EA-based methods [16], [20]–[22] that use limited search budgets are far from state-of-the-art performance and only demonstrated on small-scale datasets.

In this paper, we present NSGANet, a multi-objective evolutionary algorithm for NAS to address the aforementioned limitations of current approaches. The salient features of the proposed algorithm are summarized follows:

1) Rooted in the framework of evolutionary algorithms, we extend our previous work [23] by (i) a more comprehensive search space that encodes both layer operations and connections; (ii) adjusted generic operators accompanying the modified search space; and (iii) a more thorough lower-level optimization process for weight learning, leading to a more reliable estimation of the classification performance of architectures.
2) NSGANet processes a set of architectures simultaneously. In each iteration, all candidate architectures are ranked into different levels of importance based on both classification performance and complexity in compute requirements. The exploration of the design space is carried out through recombining sub-structures between architectures, and mutating architectural components. Subsequently, a Bayesian Learning guided exploitation step expedites the convergence of the search by reinforcing the commonly shared patterns in generating new architectures.
3) As evidenced by our extensive experiments on benchmark image classification and human chest X-ray datasets, NSGANet efficiently finds architectures that are competitive with or in most cases outperform architectures that are designed both manually by human experts and automatically by algorithms. In particular, NSGANet outperforms the current state-of-the-art evolutionary NAS method [19] on CIFAR-10, CIFAR-100, and ImageNet, while using 100x less search expense. We further validate performance of NSGANet on extended versions of benchmark datasets designed to evaluate generalization performance and robustness to common observable corruptions and adversarial attacks.
4) By obtaining a set of architectures in one run, NSGANet allows the designers to choose a suitable network a-posteriori as opposed to a pre-defined preference weighting each objective prior to the search. Further post-optimal analysis of the set of non-dominated architectures oftentimes reveals valuable design principles, which stays as another benefit for posing the NAS problem as a multi-objective optimization problem, as in case of NSGANet.

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 neural architecture search. 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. Broadly speaking, these approaches can be divided into evolutionary algorithm (EA), reinforcement learning (RL), and relaxation-based approaches – with a few additional methods falling outside these categories. In this section, we provide a brief overview of these approaches and refer the readers to [24] and [25] for a more comprehensive survey of early and recent literature survey on the topic, respectively.

A. Evolutionary Algorithms

Designing neural networks through evolution, or *neuroevolution*, has been a topic of interest for a long time, first showing notable success in 2002 with the advent of the neuroevolution of augmenting topologies (NEAT) algorithm [26]. In its original form, NEAT evolves network topologies along with weights and hyper-parameters simultaneously and performs well on simple control tasks with comparatively small fully connected networks. Miikkulainen et al. [27] attempt to extend NEAT to deep networks with CoDeepNEAT, using a co-evolutionary

approach that achieves limited results on the CIFAR-10 dataset. CoDeepNEAT does, however, produce state-of-the-art results in the Omniglot multi-task learning domain [28].

More recent neuroevolution based approaches focus solely on evolving 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 [16] is one of the early studies that shows the promise of using EAs for NAS. Real et al. [17] introduced perhaps the first truly large scale application of a simple EA to NAS. The extension of this method presented in [19], called AmoebaNet, provides the first large scale comparison of EA and RL methods. Their EA, using an age-based selection similar to [29], searches over the same space as NASNet [9], and has demonstrated faster convergence to an accurate network when compared to RL and random search. Furthermore, AmoebaNet achieves state-of-the-art results on both CIFAR-10 and ImageNet datasets.

Concurrently, another streamlining of EA methods for use in budgeted NAS has emerged. Suganuma et al. [22] use Cartesian genetic programming to assemble existing block designs (e.g., Residual blocks) and show competitive results under a low complexity regime. Sun et al. in [21] use a random forest as an off-line 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 without loss of classification performance when compared to their previous work [20].

Evolutionary multi-objective optimization (EMO) approaches have infrequently been used for NAS. Kim et al. [30] present NEMO, one of the earliest EMO approaches to evolve CNN architectures. NEMO uses NSGA-II [31] to maximize classification performance and inference time of a network and searches over the space of the number of output channels from each layer within a restricted space of seven different architectures. 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 a custom-designed approximate network morphisms [33], which allows newly generated networks to share parameters with their forerunners, obviating the need to train new networks from scratch. However, LEMONADE still requires close to 100 GPU-days to search on the CIFAR datasets.

B. Reinforcement Learning

Q-learning [34] is a very popular value iteration method used for RL. The MetaQNN method [35] 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. [36] 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 [37].

A policy gradient method seeks to approximate some non-differentiable reward function to train a model that requires parameter gradients, like a neural network architecture. Zoph and Le [8] first apply this method in architecture search to train a recurrent neural network controller that constructs networks. The original method in [8] uses the controller to generate the entire network at once. This contrasts with its successor, NASNet [9], 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. Hsu et al. [38] extends the NASNet approach to a multi-objective domain to optimize multiple linear combinations of accuracy and energy consumption criteria using different scalarization parameters.

C. 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 [10]. DARTS seeks to improve search efficiency by fixing the weights while updating the architectures, showing convergence on both CIFAR-10 and Penn Treebank [39] within one day in wall clock time on a single GPU card. Subsequent approaches in this line of research include [11]–[13], [40]. 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 NSGANet 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. [41] 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 [42] 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 [43] show that an augmented random search approach is an effective alternative to NAS. Kandasamy et al. [44] present a Gaussian-process-based approach to optimize network architectures, viewing the process through a Bayesian optimization lens.

III. Proposed Approach

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 [45]. We mathematically formulate the problem as,

$$\begin{aligned} \underset{\alpha\in\mathcal{A}}{\text{minimize}} \quad & \big\{\mathcal{L}_{vld}(\alpha,\omega^*),\mathcal{C}(\alpha)\big\}, \\ \text{subject to} \quad & \omega^* \in \underset{\omega\in\Omega}{\operatorname{argmin}}\, \mathcal{L}_{trn}(\omega,\alpha), \end{aligned} \tag{2}$$

where $\mathcal{C}(\cdot)$ measures the complexity of architectures and the other associated quantities are the same as in Eq. (1).

Our proposed algorithm, NSGANet, 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

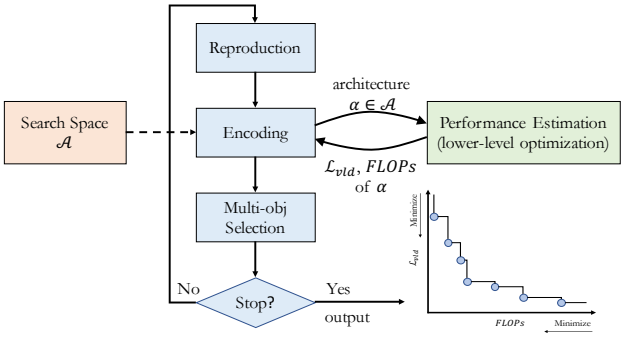


Fig. 1: **Overview:** NSGANet's search procedure consists of three main components: search space (red), search process (blue), and performance estimation (green).

through crossover and mutation to the more promising of the architectures already found, also known as *parents*, from the population. Every member in the population (including both parents and offspring) compete for survival and reproduction (becoming a parent) in each iteration. The initial population may be generated randomly or guided by prior-knowledge, i.e. seeding the past successful architectures directly into the initial population. Subsequent to initialization, NSGANet proceeds 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 flowchart of the overall approach is shown in Fig. 1. NSGANet consists of three main components: (i) the *search space* that defines all feasible architectures, (ii) the *performance estimation* strategy that assesses the performance (via lower-level optimization) and complexity of each architecture candidate, and (iii) the multi-objective search strategy (blue boxes in Fig. 1) that iteratively selects and reproduces promising architectures. A set of architectures representing efficient trade-offs between network performance and complexity is obtained at the end of evolution. In the remainder of this section, we provide a detailed description of the 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 [16], [21], [22], [32] search only one aspect of the architecture space—e.g., the connections and/or hyper-parameters. In contrast, NSGANet 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 [9], [10], [19], [41] 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 [46]
- 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 [23]. Specifically, the current search space (i) 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 more recent layers such as squeeze-and-excitation [46] and more computationally efficient layers like local binary conv [6].

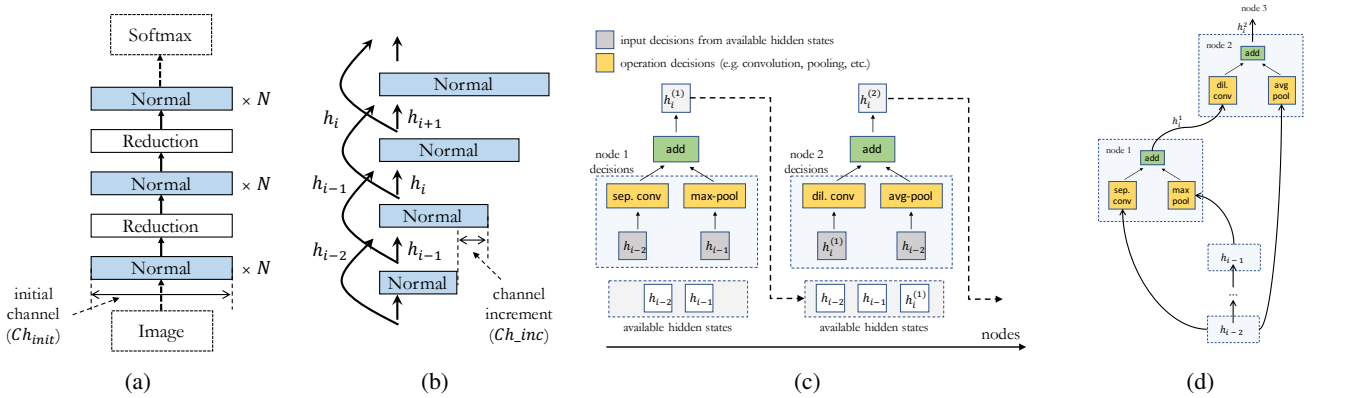
With the above-mentioned search space, there are in total 20 decisions to constitute a block structure, i.e. choose two pairs of input and operation for each node, and repeat for five nodes. The resulting number of combinations for a block structure is:

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

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 NSGANet 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



(a) (b) (c) (d)

Fig. 2: Schematic of the NSGANet search space motivated from [9]: (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).

Algorithm 1: Performance Evaluation of a CNN

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

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

search requires another optimization to first identify the optimal values of the associated weights (see Algorithm 1). Even though there exists well-established gradient descent algorithms to efficiently solve this optimization, repeatedly executing this algorithm for every candidate architecture renders the overall process computationally very prohibitive. Therefore, to overcome this computational bottleneck, we adopt the *proxy model* [9], [19] and *early stopping* [47] techniques as in previous work. See Section V-C for more details.

A number of metrics can serve as proxies for complexity, including: the number of active nodes, number of active connections between the nodes, number of parameters, inference time and number of floating-point operations (FLOPs) needed to execute the forward pass of a given architecture. Our initial experiments considered each of these metrics in turn. We concluded from extensive experimentation that inference time cannot be estimated reliably due to differences and inconsistencies in the computing environment, GPU manufacturer, ambient temperature, etc. Similarly, the number of parameters, active connections or active nodes only relate to one aspect of 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 "crowded-ness" concepts proposed in [31].

C. Reproduction

The process of generating new architectures is referred to as reproduction in NSGANet. 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 [48]. To address this problem, we use binary tournament selection [49] 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 selection criterion described in Section III-B becomes one of the parents. This process is repeated to select a second parent architecture; the two parent architectures then undergo a crossover operation.

In NSGANet, 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 we exchange a subset of the Normal and Reduction blocks between the parents; and 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. Figure 3 illustrates an example

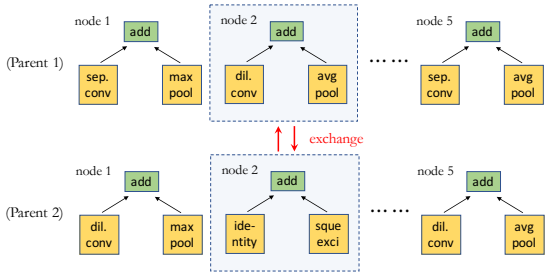


Fig. 3: Illustration of node-level crossover.

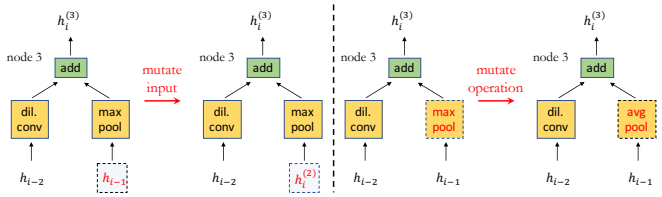


Fig. 4: **Input and Operation Mutation:** Dashed line boxes with red color highlight the mutation. h_{i-2}, h_{i-1} are outputs from previous-previous and previous blocks, respectively. $h_i^{(3)}$ indicates output from node 3 of the current block.

of node-level crossover. Note that two offspring architectures are generated after each crossover operation, and an offspring population of the same size as the parent population is generated in each iteration.

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 [50] subsequent to crossover. We allow mutations to be applied on both the input hidden states and the choice of operations. Figure 4 shows an example of each type of mutation. The PM operator inherits the parent-centric convention, 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 h_{i-2} by PM. A similar logic is applied in case of mutation on layer operations. The crossover and mutation operators jointly drive the exploration aspect of the search.

After a sufficient number of architectures has been explored (consuming 2/3 of the total computational budget), 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. The exploitation step in NSGANet is heavily inspired by the Bayesian Optimization Algorithm (BOA) [51], which is explicitly designed for problems with inherent correlations between the optimization variables. In the context of our NAS encoding, this translates to correlations in the nodes within a block and connections across blocks. Exploitation uses past information across all networks evaluated to guide the final part of the search. More specifically, say

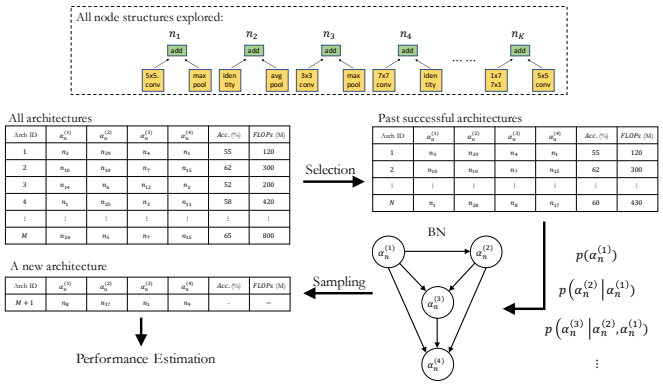


Fig. 5: Illustrative example of BN-based exploitation step in NSGANet: given past successful architectures (based on both classification accuracy and complexity in FLOPs), we construct a BN relating the dependencies between the four nodes inside the Normal block. A new architecture is then sampled from this BN and proceed forward for performance estimation.

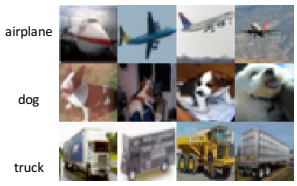
we are designing a Normal block with three nodes, namely $\alpha_n^{(1)}$, $\alpha_n^{(2)}$, and $\alpha_n^{(3)}$. We would like to know the relationship among these three nodes. For this purpose, we construct a Bayesian Network (BN) relating these variables, modeling the probability of Normal blocks 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 $p\left(\alpha_n^{(1)}\right)$, $p\left(\alpha_n^{(2)}|\alpha_n^{(1)}\right)$, and $p\left(\alpha_n^{(3)}|\alpha_n^{(2)},\alpha_n^{(1)}\right)$ 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 assign 25% of the offspring to be created by BN and we adaptively update this probability based on the ratio between the offspring created by BN and by genetic operators that survived into the next iteration.

IV. Experimental Setup and Results

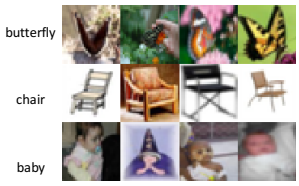
In this section, we will evaluate the efficacy of NSGANet on multiple benchmark image classification datasets.

A. Baselines

To demonstrate the effectiveness of the proposed algorithm, we compare the non-dominated architectures achieved at the conclusion of NSGANet'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 [52], and DenseNet [3], etc. The second and third groups range from earlier methods [8], [16], [17] that are oriented towards "proof-of-concept" for NAS, to more recent methods [9], [10], [15], [19], many of which improve state-of-the-art results on



(a) CIFAR-10



(b) CIFAR-100

(c) ImageNet

Fig. 6: 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.

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. CIFAR-10 and -100 are similar, as each of them consists of 50,000 images for training and 10,000 images for testing, with each image being 32 x 32 pixels. CIFAR-100 extends CIFAR-10 by adding 90 more classes resulting in 10$\times$ fewer training examples per class. ImageNet is a large-scale database containing more than 1.4 million images from 1,000 different classes and each image is of a much bigger size. ImageNet is significantly more challenging than the CIFAR datasets, and the cumulative progress is often cited as one of the main breakthroughs in computer vision and machine learning [53]. A gallery of examples from these three datasets is provided in Fig. 6.

B. Implementation Details

Motivated by efficiency and practicality considerations most existing NAS methods, including [9], [14], [19], [20], 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 improves its generalization. We emphasize that the original testing set is *never* used to guide the selection of architectures in any form during the search.

The search itself is repeated five times with different initial random seeds. We select and report the performance of the median run as measured by hypervolume (HV). Such a procedure ensures the reproducibility of our NAS experiments and mitigates the concerns that have arisen in recent NAS studies [43], [54]. We use the standard SGD [55] 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 using Pytorch [56].

TABLE I: Summary of Hyper-parameter Settings.

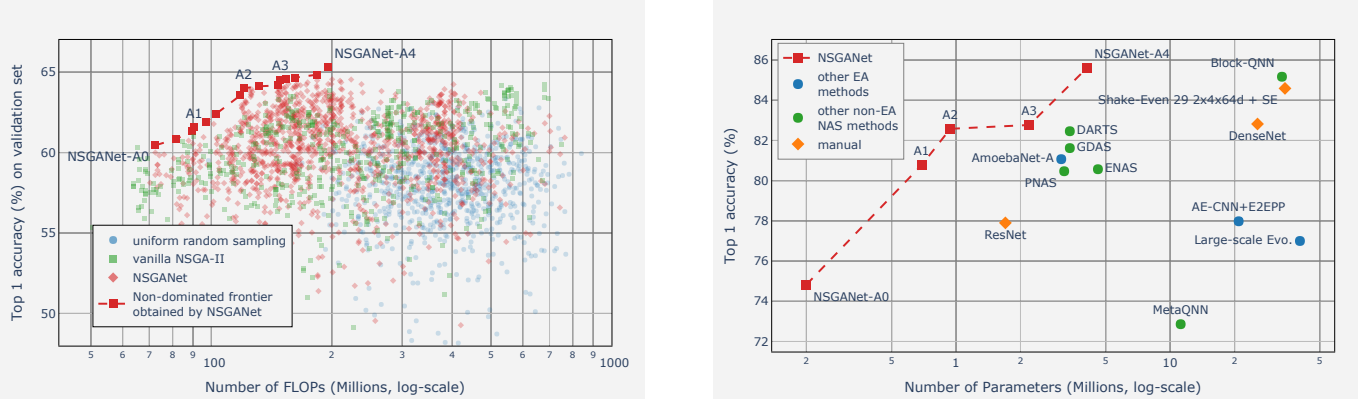
Categories	Parameters	Settings
search space	# of initial channels (Ch_{init})	32
	# of channel increments (Ch_{inc})	6
	# of repetitions of Normal blocks ($\mathcal{N}$)	4/5/6
gradient descent	batch size	128
	weight decay (L_2 regularization)	5.00E-04
	epochs	36/600
	learning rate schedule	Cosine Annealing [57]
search strategy	population size	40
	# of generations	30
	crossover probability	0.9
	mutation probability	0.1

C. Effectiveness of NSGANet

We first present the objective space distribution of all architectures generated by NSGANet during the course of evolution on CIFAR-100, in Fig. 7a. 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. 7a), 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 NSGANet-A0 to -A4 in ascending FLOPs order. See Fig.8 for a visualization and comparison of architectures.

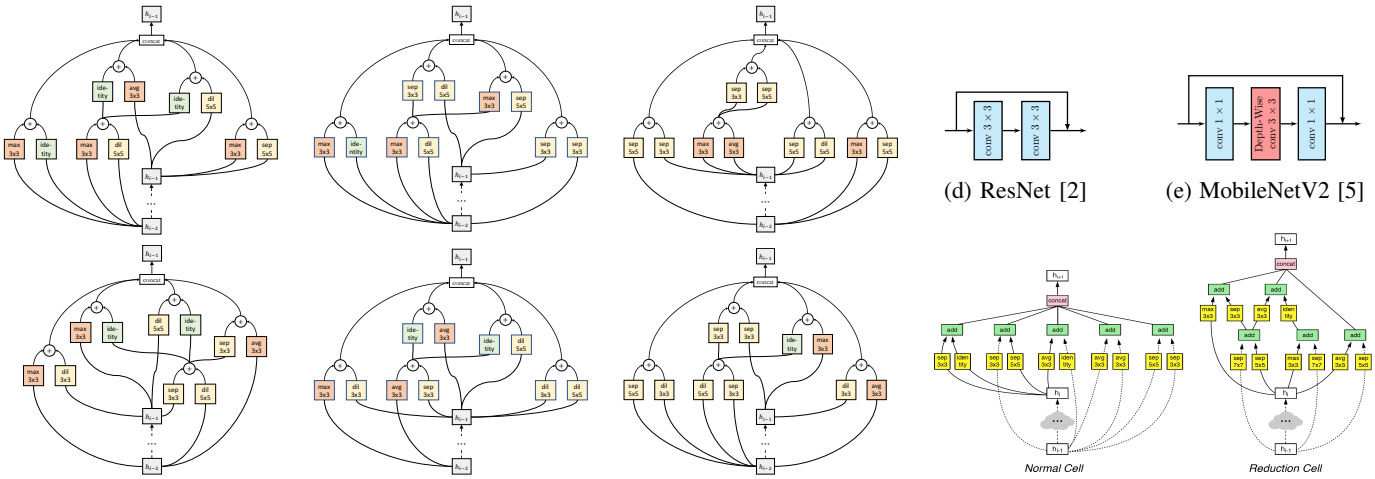
For comparison with other peer methods, we follow the training procedure in [10] and re-train the weights of NSGANet-A0 to -A4 thoroughly on CIFAR-100. We would like to mention that since most existing approaches do not report the number of FLOPs for the architectures used on the CIFAR-100 dataset, we instead compare their computational complexity through number of parameters to prevent potential discrepancies from re-implementation. Figure 7b shows the post-search architecture comparisons, NSGANet-A0 to A4, i.e., the algorithms derived in this paper, jointly dominate all other considered peer methods with a clear margin. More specifically, NSGANet-A1 is more accurate than the recently-published peer EA method, AE-CNN-E2EPP [21], while using **30x** fewer parameters; NSGANet-A2 surpasses the performance of the state-of-the-art EA method, AmoebaNet [19], while at the same time saving **3.4x** parameters. Lastly, NSGANet-A4 exceeds the classification accuracy of *Shake-Even 29 2x4x64d + SE* [46] with **8.4x** fewer parameters. More comparisons can be found in Table IIb.

Following the practice adopted in most previous approaches [9], [10], [14], [19], [41], 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.



(a) During search (b) Post search

Fig. 7: (a) Accuracy vs. FLOPs of all architectures generated by NSGANet during the course of evolution on CIFAR-100. A subset of non-dominated architectures (see text), named NSGANet-A0 to A4, are re-trained thoroughly and compared with other peer methods in (b).



(a) NSGANet-A1 (b) NSGANet-A2 (c) NSGANet-A4 (d) ResNet [2] (e) MobileNetV2 [5] (f) NASNet-A [9]

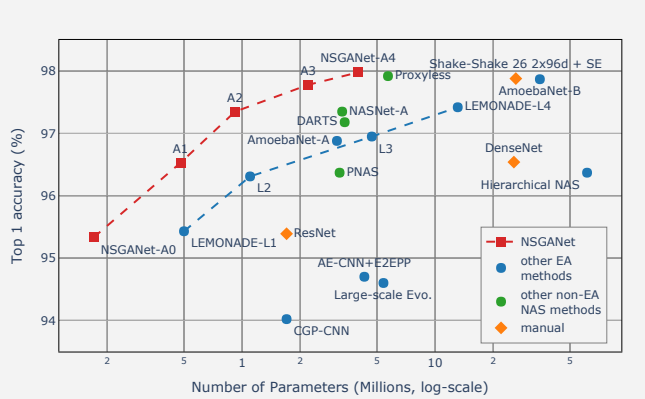
Fig. 8: Visualization of block-level structures for different architectures. The Normal and Reduction blocks are shown in the first and second rows, respectively for NSGANet architectures. Examples of blocks that are designed manually by experts [2], [5] and from other peer methods [9] are also included in (d) - (f) for comparison.

The effectiveness of NSGANet is further validated by the performance of transferred architectures on the CIFAR-10 dataset. As we show in Figs. 9a, the trade-off frontier established by NSGANet-A0 to -A4 completely dominates the frontiers obtained by the peer EMO method, LEMONADE [32], as well as those obtained with other single-objective peer methods. More specifically, NSGANet-A0 uses **27x** fewer parameters and achieves higher classification accuracy than Large-scale Evo. [17]. NSGANet-A1 outperforms Hierarchical NAS [18] and DenseNet [3] in classification, while saving **122x** and **51x** in parameters. Furthermore, NSGANet-A4 exceeds previous state-of-the-art results reported by Proxyless NAS [15] while being 1.4x more compact. Refer to Table IIa for more comparisons.

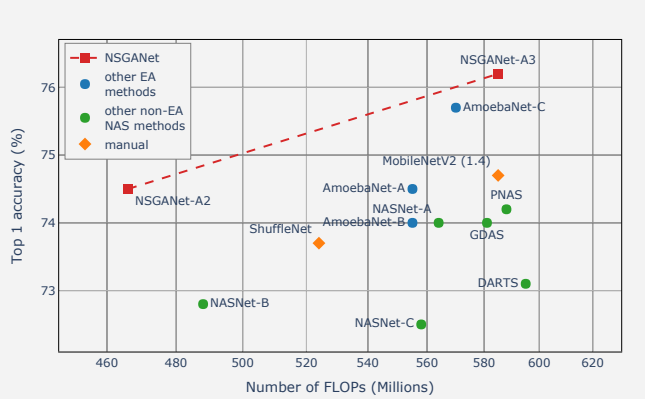
For transfer performance comparison on the ImageNet dataset, we follow previous work [5], [9], [10], [14] and use the ImageNet-mobile setting, i.e., the setting where number of FLOPs is less than 600M. The NSGANet-A0 is too simple for the ImageNet dataset and NSGANet-A4 exceeds the 600M FLOPs threshold for the mobile setting, so we provide results only for NSGANet-A1, -A2 and -A3. Figure 9b shows the objective space Pareto comparison with the other peer methods. Clearly, NSGANet can achieve a better trade-off between the objectives. NSGANet-A2 dominates a wide range of peer methods including ShuffleNet [4] by human experts, NASNet-A [9] by RL, DARTS [10] by relaxation-based methods, and AmoebaNet-A [19] by EA. Moreover, NSGANet-A3 surpasses previous state-of-the-art performance reported by MobileNet-V2 [5] and AmoebaNet-C [19] on mobile-setting with a marginal overhead in FLOPs (1% - 3%).

D. Efficiency of NSGANet

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 NSGANet.



(a) CIFAR-10



(b) ImageNet

Fig. 9: Transferability of the NSGANet 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 NSGANet and other baseline methods. NSGANet architectures are obtained by searching on CIFAR-100. NSGANet results on CIFAR-10 and ImageNet are obtained by re-training the weights with images from their respective datasets. Ratio-to-NSGANet 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-NSGANet
NSGANet-A0	**EA**	**27**	**95.33%**	**0.2M**	**1x**
CGP-CNN [22]	EA	27	94.02%	1.7M	8.5x
Large-scale Evo. [17]	EA	2,750	94.60%	5.4M	27x
AE-CNN+E2EPP [21]	EA	7	94.70%	4.3M	21x
ResNet [2]	manual	-	95.39%	1.7M	8.5x
NSGANet-A1	**EA**	**27**	**96.51%**	**0.5M**	**1x**
Hierarchical NAS [18]	EA	300	96.37%	61.3M	122x
PNAS [41]	SMBO	150	96.37%	3.2M	6.4x
DenseNet [3]	manual	-	96.54%	25.6M	51x
NSGANet-A2	**EA**	**27**	**97.35%**	**0.9M**	**1x**
AmoebaNet-A [19]	EA	3,150	96.88%	3.1M	3.4x
DARTS [10]	relaxation	1	97.18%	3.4M	3.8x
NSGANet-A3	**EA**	**27**	**97.78%**	**2.2M**	**1x**
NASNet-A [9]	RL	1,575	97.35%	3.3M	1.5x
LEMONADE [32]	EA	90	97.42%	13.1M	6.0x
NSGANet-A4	**EA**	**27**	**97.98%**	**4.0M**	**1x**
AmoebaNet-B [19]	EA	3,150	97.87%	34.9M	8.7x
Proxyless NAS [15]	RL	1,500	97.92%	5.7 M	1.4x

(b) CIFAR-100

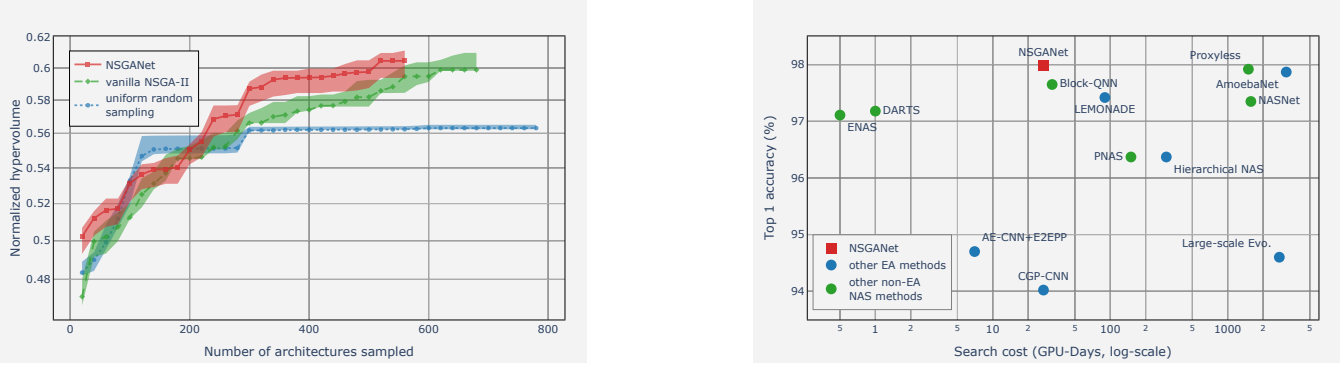
Architecture	Search Method	GPU-Days	Top-1 Acc.	#Params	Ratio-to-NSGANet
NSGANet-A0	**EA**	**27**	**74.83%**	**0.2M**	**1x**
Genetic CNN [16]	EA	17	70.95%	-	-
MetaQNN [35]	RL	90	72.86%	11.2M	56x
NSGANet-A1	**EA**	**27**	**80.77%**	**0.7M**	**1x**
Large-scale Evo. [17]	EA	2,750	77.00%	40.4M	58x
ResNet [2]	manual	-	77.90%	1.7M	2.4x
AE-CNN+E2EPP [21]	EA	10	77.98%	20.9M	30x
PNAS [41]	SMBO	150	80.47%	3.2M	4.6x
ENAS [14]	RL	0.5	80.57%	4.6M	6.6x
NSGANet-A2	**EA**	**27**	**82.58%**	**0.9M**	**1x**
AmoebaNet-A [19]	EA	3,150	81.07%	3.1M	3.4x
GDAS [12]	relaxation	0.2	81.62%	3.4M	3.8x
DARTS [10]	relaxation	1	82.46%	3.4M	3.8x
NSGANet-A3	**EA**	**27**	**82.77%**	**2.2M**	**1x**
NSGANet-A4	**EA**	**27**	**85.62%**	**4.1M**	**1x**
DenseNet [3]	manual	-	82.82%	25.6M	6.2x
SENet [46]	manual	-	84.59%	34.4M	8.4x
Block-QNN [36]	RL	32	85.17%	33.3M	8.1x

(c) ImageNet

Architecture	Search Method	GPU-Days	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-NSGANet	#FLOPs	Ratio-to-NSGANet
NSGANet-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
NSGANet-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 [9]	RL	1,575	74.0%	91.3%	5.3M	1.3x	564M	1.2x
PNAS [41]	SMBO	150	74.2%	91.9%	5.1M	1.2x	588M	1.3x
AmoebaNet-A [19]	EA	3,150	74.5%	92.0%	5.1M	1.2x	555M	1.2x
DARTS [10]	relaxation	1	73.1%	91.0%	4.9M	1.2x	595M	1.3x
NSGANet-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 [19]	EA	3,150	75.7%	92.4%	6.4M	1.3x	**570M**	**1x**

† SMBO stands for sequential model-based optimization. SENet is the abbreviation for Shake-Even 29 2x4x64d + SE.
‡ The CIFAR-100 accuracy and #params for ENAS [14] and DARTS [10] are from [12]. #Params for AE-CNN+E2EPP are from [20].

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 [31] algorithm with all hyper-parameters set to default values. 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



(a) HV

(b) Search cost vs. Accuracy

Fig. 10: Search efficiency comparison between NSGANet 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 NSGANet and Block-QNN [36], where the CIFAR-100 dataset is used for.

hypervolume (HV) that we obtain. The HV measurements shown in Fig. 10a suggest that NSGANet 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 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 NSGANet 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 [36] which is measured on CIFAR-100. From the search cost comparison in Fig. 10b, we observe that our proposed algorithm is more efficient at identifying a set of architectures than a number of other approaches, and the set of architectures obtained has higher performance. More specifically, NSGANet simultaneously finds multiple architectures using **10x** to **100x** less search cost in GPU-days than most of the considered peer methods, including Hierarchical NAS [18], AmoebaNet [19], NasNet [9], and Proxyless NAS [15], all of which find a single architecture at a time. When compared to the peer multi-objective NAS method, LEMONADE [32], NSGANet 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 Appendix A-C1.

E. Observations on Evolved Architectures

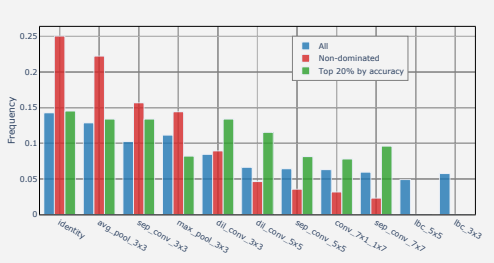
Population-based approaches with multiple conflicting objectives often lead to a set of diverse solution candidates, which can be "mined" for commonly shared design principles [58]. In order to discover any patterns for more efficient architecture design, we analyzed the entire history of architectures generated by NSGANet. 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. 11a (see also Appendix A-C2 for our follow-up study).
- Larger kernel size improves classification accuracy. As shown in Fig. 11a, 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.
- Parallel operations are more beneficial to classification performance than sequential operations. The concatenation dimensions shown in Fig. 11b indicate the number of parallel paths in a block structure. Clearly, the higher the number of inputs concatenated, the higher the validation accuracy, on average. Similar findings are also reported in previous work [19], [52].

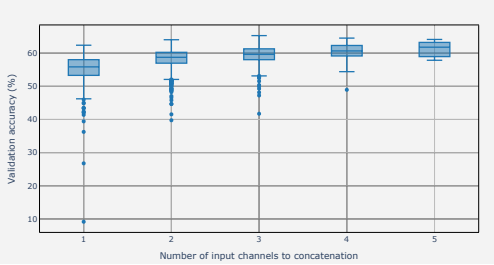
The above common properties of multiple final non-dominated solutions stay as important knowledge for future applications. It is noteworthy that such a post-optimal knowledge extraction process is possible only from a multi-objective optimization study, another benefit that we enjoy for posing NAS as 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 over-fitting of 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

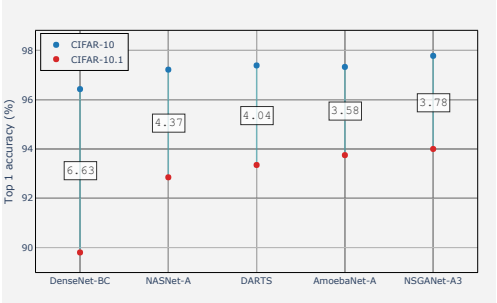


(a) Frequency of operation choices.

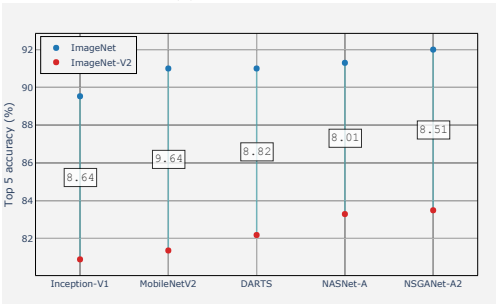


(b) Concatenation dimension vs. Accuracy

Fig. 11: **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 results in more reliable performance improvement but at the cost of computational efficiency.



(a) CIFAR-10.1



(b) ImageNet-V2

Fig. 12: **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 (%).

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. [59] propose two new testing sets, CIFAR-10.1 and ImageNet-V2. Refer to Appendix A-B1 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. 12a and 12b 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 architectures with highest accuracy on the original testing set (NSGANet in this case) is also the architectures with highest accuracy on the 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. 12). These results provide evidence that extensive benchmarking on the original testing sets is an effective way to measure the progress of architectures.

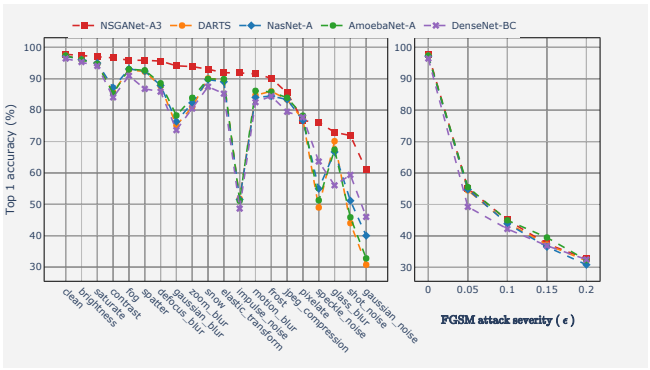
B. Robustness

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 [60] recently introduced two new testing datasets, CIFAR-10-C and CIFAR-100-C, by applying commonly observable corruptions (e.g., noise, weather, compression, etc.) to the clean images from the original datasets. Each dataset contains images perturbed by 19 different types of corruption at five different levels of severity. More details and visualizations are provided in Appendix **??**. In addition, we include adversarial images as examples of worst-case corruption. We use the fast gradient signed method (FGSM) [61] to construct adversarial examples for both the CIFAR-10 and -100 datasets. The severity of the attack is controlled via a hyper-parameter ϵ as shown below:

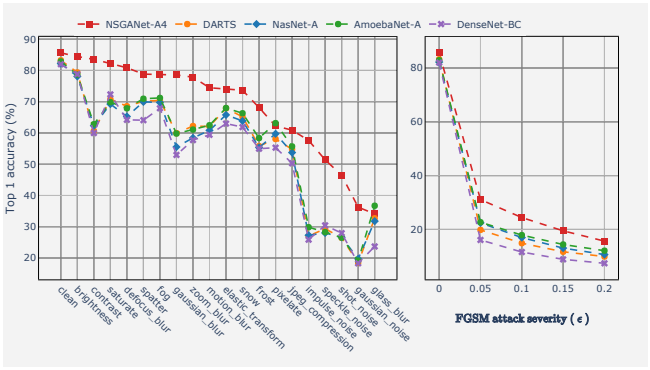
$$\boldsymbol{x}' = \boldsymbol{x} + \epsilon \ \text{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y_{true})),$$

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

Our empirical findings summarized in Figs. 13a and 13b 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



(a) CIFAR-10



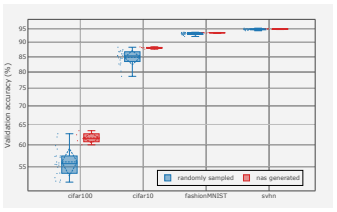
(b) CIFAR-100

Fig. 13: **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. Robustness to corruptions and adversarial attacks is correlated to prediction accuracy on the clean test images.

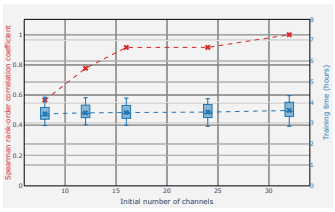
NSGANet architectures perform noticeably better than other peer methods' architectures on corrupted datasets even though the robustness measurement was never a part of the architecture selection process in NSGANet. However, we emphasize that no architectures are considered robust to corruption, especially 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 Appendix A-D.

C. Ablation Studies

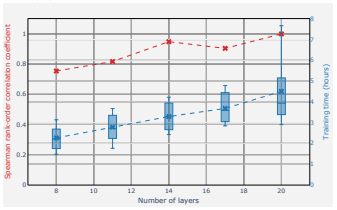
Dataset for Search: As previously mentioned in Section III-B, our proposed method differs from most of the existing peer methods in the choice of datasets on which the search is carried out. Instead of directly following the current practice of using the CIFAR-10 dataset, we investigated the utility 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 [62], fashionMNIST [63], CIFAR-10, and CIFAR-100 for comparison. The choice was based on a number of factors including the number



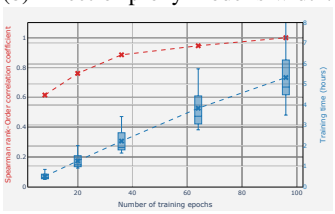
(a) Effect of dataset choice.



(b) Effect of proxy model's width.



(c) Effect of proxy model's depth.



(d) Effect of training epochs.

Fig. 14: (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.

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. 14a. Empirically, we observe that the CIFAR-100 dataset is challenging enough for architectural differences to effect predict performance. This can be observed in Fig. 14a 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 the 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 [9], [19]. 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

*Small architecture size allows larger batch size to be used and a lower number of epochs to converge under Algorithm 1.

necessary that the performance of an architecture measured at proxy-model scale can serve as a reliably indicator of the architecture's performance measured at the desired scale.

To determine the smallest proxy model that can provide a reliable estimate of performance at a larger scale, we conducted parametric studies that gradually reduced the sizes of the proxy models of 100 randomly sampled architectures from our search space. Then, we measured the rank-order correlation and the savings in lower-level optimization compute time between the proxy models and the same architectures at the full scale. Figures 14b, 14c and 14d show the effect of number of channels, layers and epochs respectively on the training time and Spearman rank-order correlation between the proxy and full scale models. We make the following observations, (1) increasing the 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 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 [64]. 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 Appendix A-B3. To cope with the multi-label nature of the dataset, we use a multi-task learning setup in which each disease is treated as an individual binary classification problem. We define a 14-dimensional label vector of binary values indicating the presence of one or more diseases, resulting in a regression loss as opposed to the standard cross-entropy for single-label problems. Past approaches [64]–[66] typically extend from existing architectures, and the current state-of-the-art method [66] uses a variant of the DenseNet [3] architecture, which is designed manually by human experts.

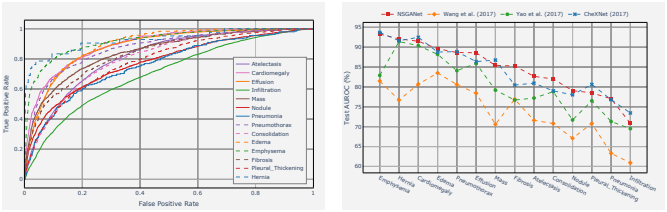
The search is setup along the same lines as that of CIFAR-100 described in Section IV, except that the classification accuracy objective is replaced by the average area under the ROC curve (AUROC) metric to guide the selection of architectures. We split the dataset into three parts with 70% for training, 10% for validation, and the remaining 20% for testing. All images are pre-processed to 224×224 pixels using ImageNet data augmentation techniques (e.g., normalization, random horizontal flipping, etc.). Once the non-dominated architectures are obtained after the convergence of evolution (28 generations averaged over 3 runs), we select the architecture that maximizes the ratio between the gain in AUROC over the sacrifice in FLOPs. We call this architecture NSGANet-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 NSGANet-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 [64], and the CheXNet [66], which is the current state-of-the-art on this dataset. We also include results from commercial AutoML systems, i.e. Google AutoML [67], and LEAF [68], as comparisons with NAS based methods. The setup details of these two AutoML systems are available in [68]. Noticeably, the performance of NSGANet-X exceeds Google AutoML's by a large margin of nearly **4** AUROC points. In addition, NSGANet-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 NSGANet-A3, which is evolved on CIFAR-100, to demonstrate the transfer learning capabilities of NSGANet.

TABLE III: AUROC on ChestX-Ray14 testing set.

Method	Type	#Params	Test AUROC (%)
Wang et al. (2017) [64]	manual	-	73.8
Yao et al. (2017) [65]	manual	-	79.8
CheXNet (2017) [66]	manual	7.0M	84.4
Google AutoML (2018) [67]	RL	-	79.7
LEAF (2019) [68]	EA	-	84.3
NSGANet-A3	**EA**	**5.0M**	**84.7**
NSGANet-X	**EA**	**2.2M**	**84.6**

† Google AutoML result is from [68].
‡ NSGANet-A3 represents results under the standard transfer learning setup.

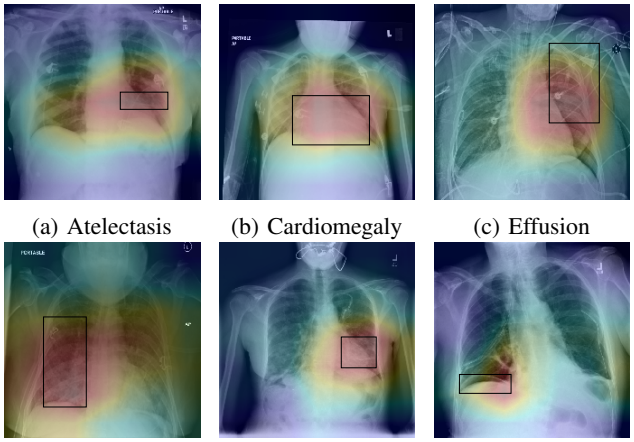


(a) (b)

Fig. 15: NSGANet-X multi-label classification performance on ChestX-Ray14 (a) and the class-wise mean test AUROC comparison with peer methods (b).

More detailed results showing the disease-wise ROC curve of NSGANet-X and disease-wise AUROC comparison with other peer methods are provided in Figs. 15a and 15b, respectively. To understand the pattern behind the disease classification decisions of NSGANet-X, we visualize the class activation map (CAM) [69], which is commonly adopted for localizing the discriminative regions for image classification. In the examples shown in Fig. 16a - 16f, stronger CAM areas are covered with warmer colors. We also outline the bounding boxes provided by the ChestX-Ray14 dataset [64] as references.

These results further validate the ability of our proposed algorithm to generate task-dependent architectures automatically. Conventional approaches, e.g., transfer learning from existing architectures, can be effective in yielding similar performance, however, as demonstrated by NSGANet, 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 NSGANet to keypoint prediction on cars (see Appendix A-D1).



(a) Atelectasis (b) Cardiomegaly (c) Effusion

(d) Infiltrate (e) Pneumonia (f) Pneumothorax

Fig. 16: Examples of class activation map [69] of NSGANet-X, highlighting the class-specific discriminative regions. The ground truth bounding boxes are plotted over the heatmaps.

VII. Conclusions

In this paper, we have presented NSGANet, an evolutionary multi-objective algorithm for neural architecture search. NSGANet explores the design space of architectures through recombining and mutating architectural components. NSGANet further improves the search efficiency by exploiting the patterns among the past successful architectures via Bayesian Learning. Experiments on CIFAR-10, CIFAR-100, ImageNet datasets have demonstrated the effectiveness of NSGANet. Further analysis towards validating the generalization and robustness aspects of the obtained architectures are also provided along with an application to common thorax diseases classification on human chest X-rays. We believe these results are encouraging and demonstrates the utility of evolutionary methods for automated machine learning.

Acknowledgments

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 *CVPR*, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.

[4] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, 2018.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.

[6] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *CVPR*, 2017.

[7] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.

[8] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv preprint arXiv:1611.01578*, 2016.

[9] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018.

[10] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *ICLR*, 2019.

[11] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *ICLR*, 2019.

[12] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *CVPR*, 2019.

[13] 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 *CVPR*, 2019.

[14] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018.

[15] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.

[16] L. Xie and A. Yuille, "Genetic CNN," in *ICCV*, 2017.

[17] 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 *ICML*, 2017.

[18] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *ICLR*, 2018.

[19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.

[20] 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*, 2019.

[21] 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*, 2019.

[22] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *GECCO*, 2017.

[23] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: Neural architecture search using multi-objective genetic algorithm," in *GECCO*, 2019.

[24] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[25] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

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

[27] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[28] J. Liang, E. Meyerson, and R. Miikkulainen, "Evolutionary architecture search for deep multitask networks," in *GECCO*, 2018.

[29] G. S. Hornby, "Alps: the age-layered population structure for reducing the problem of premature convergence," in *GECCO*, 2006.

[30] 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 *ICML AutoML Workshop*, 2017.

[31] 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.

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

[33] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *ICML*, 2016.

[34] 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

[35] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2017.

[36] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," 2018.

[37] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (Canadian Institute for Advanced Research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[38] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[39] 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.

[40] 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 *CVPR*, 2019.

[41] 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 *ECCV*, 2018.

[42] 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 *ICLR*, 2018.

[43] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," *arXiv preprint arXiv:1902.07638*, 2019.

[44] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *NeurIPS*, 2018.

[45] G. Eichfelder, "Multiobjective bilevel optimization," *Mathematical Programming*, vol. 123, no. 2, pp. 419–449, 2010.

[46] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *CVPR*, 2018.

[47] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *CVPR*, 2017.

[48] 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.

[49] 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.

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

[51] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Boa: The bayesian optimization algorithm," in *GECCO*, 1999.

[52] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[53] J. Malik, "Technical perspective: What led computer vision to deep learning?" *Communications of the ACM*, vol. 60, no. 6, pp. 82–83, 2017.

[54] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," *arXiv preprint arXiv:1904.01569*, 2019.

[55] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[56] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NeurIPS Autodiff Workshop*, 2017.

[57] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[58] K. Deb and A. Srinivasan, "Innovization: Innovating design principles through optimization," in *GECCO*, 2006.

[59] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?" *arXiv preprint arXiv:1902.10811*, 2019.

[60] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *ICLR*, 2019.

[61] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[62] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[63] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[64] 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 *CVPR*, 2017.

[65] 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.

[66] 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.

[67] 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.

[68] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning," in *GECCO*, 2019.

[69] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *CVPR*, 2016.

[70] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy (SP)*, 2017.

[71] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," in *ICLR*, 2015.

[72] V. Naresh Boddeti, T. Kanade, and B. Vijaya Kumar, "Correlation filters for object alignment," in *CVPR*, 2013.

[73] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *ECCV*, 2016.

APPENDIX A

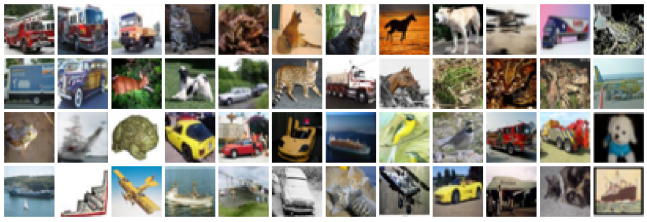
In this appendix, we provide additional details of the layer operations in Section A-A, the datasets in Section A-B. Other potential utilities of our proposed algorithm are demonstrated in Section A-C. The code will be made publicly available at https://github.com/ianwhale/nsga-net.

A. Details of the Considered Layer Operations

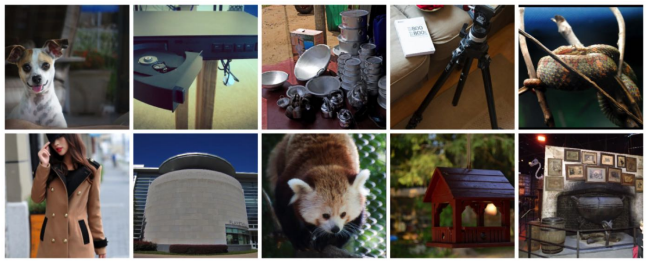
As described in Section III-A, we form a operation pool consists of 12 different choices of convolution, pooling and etc., based on their prevalence in the CNN literature. Most of these operations can be directly called from standard Deep Learning libraries, like Pytorch, TensorFLow, Caffe, etc. Here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

B. Datasets Details

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 [59]. 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.



(a) CIFAR-10.1



(b) ImageNet-V2

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

†refer to both less frequently used operations and operations under less commonly followed setups.

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

```
class SepConv(nn.Module)
# depth-wise separable convolution in NSGANet
# 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 LBConv(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
```

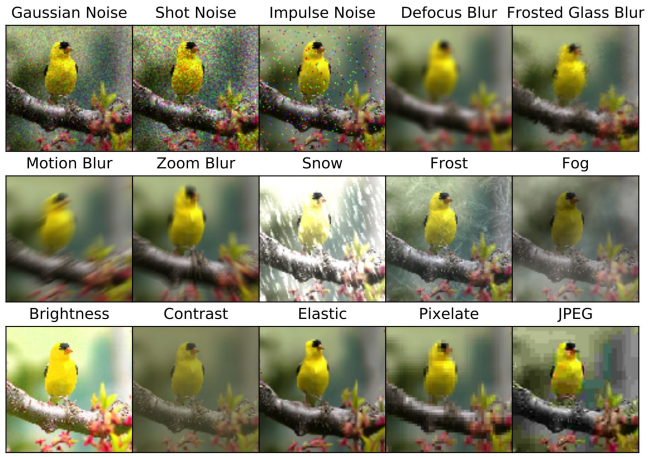
(b) Local Binary Convolution

```
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)
```

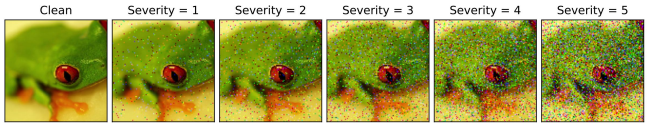
(c) 1x7 convolution then 7x1 convolution

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 [61] on

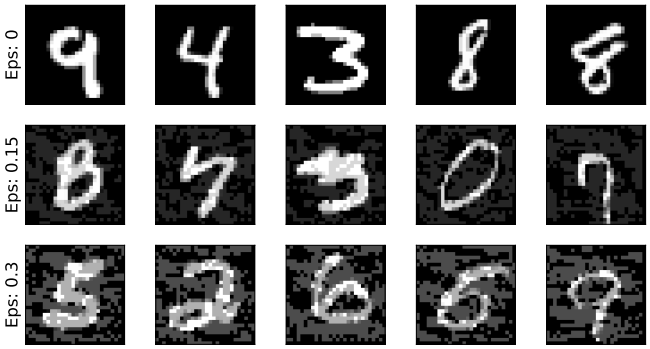
MNIST dataset is provided in Fig. 18c.



(a) Types of corruptions



(b) Severity of corruptions



(c) Adversarial examples from FGSM [61] on MNIST.

Fig. 18: Visualization of different types of corruptions and different levels of severity. Examples are from [60]. 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).

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.

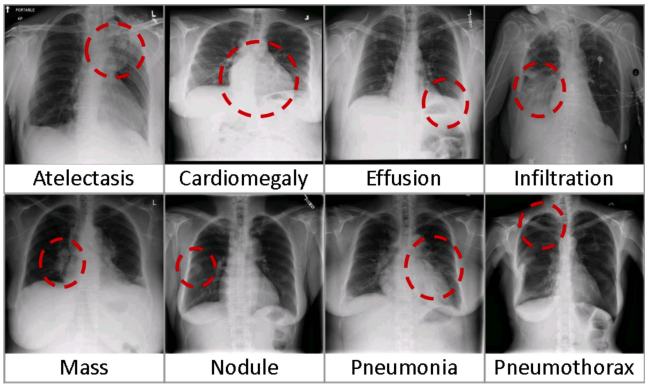


Fig. 19: Visualization of ChestXray14 datasets. Examples showing eight common thoracic diseases are from [64].

C. Follow-up Studies

1) Single-Objective NSGANet: Despite the superior effectiveness and efficiency of the proposed algorithm, the computation overheads of 27 GPU-days of NSGANet can be infeasible for users with few GPU cards. Towards understanding of the overall search wall time limit of NSGANet, 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 NSGANet-B0 and NSGANet-B1, respectively.

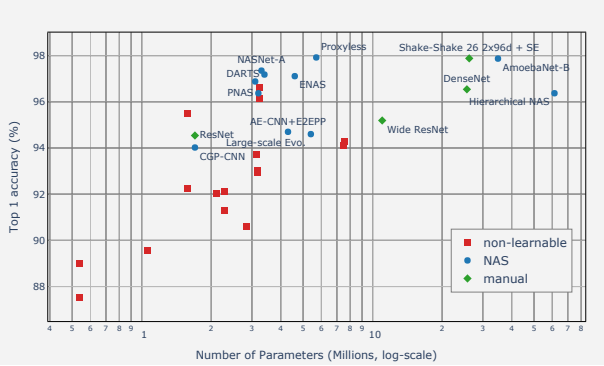
TABLE V: NSGANet with single objective of maximizing classification accuracy on CIFAR-10 and early terminations.

Method	Type	#Params	Top-1 Acc.	GPU-Days
Genetic CNN [16]	EA	-	92.90%	17
AE-CNN+E2EPP [21]	EA	4.3M	94.70%	7
ENAS [14]	RL	4.6M	97.11%	0.5
DARTS [10]	differential	3.3M	97.24%	1
NSGANet-B0	**EA**	**3.3M**	**96.15%**	**1**
NSGANet-B1	**EA**	**3.3M**	**97.25%**	**4**

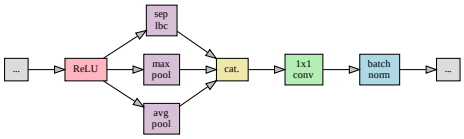
Results in Table V confirm that our proposed algorithm can be more efficient in GPU-days than the other two EA-based peer methods, Genetic CNN [16] and AE-CNN-E2EPP [21]. Specifically, NSGANet obtains the architecture B1 in 3 less GPU-days than AE-CNN-E2EPP, in addition to the B1 architecture being more 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 [14] and DARTS [10] are still more efficient in GPU-days than our proposed method. The weight sharing method could in principle be applied to NSGANet 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,

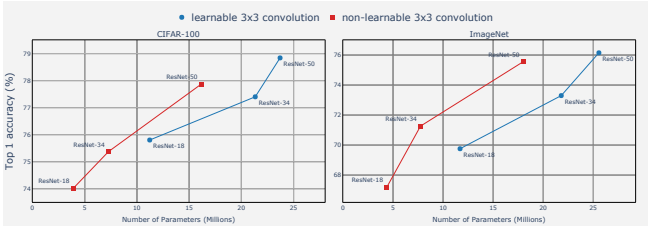
e.g. identity mapping, average/max pooling, etc., in trading off classification performance for architectural complexity. To further validate this observation, we consider a expanded range of operations including both non-parametric and weight-fixed operations, which we name as *non-learnable operations* 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.



(a)



(b)



(c)

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.

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.

D. 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 [61] 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.

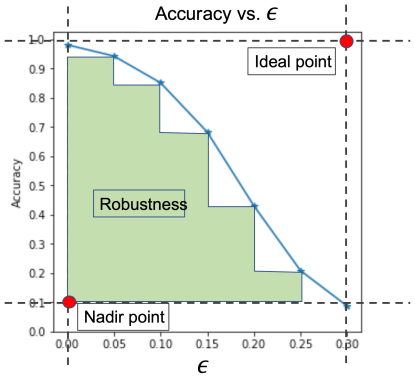


Fig. 21: **Robustness Objective:** We define a robustness objective under the FGSM [61] attack as follows: 1) obtain classification accuracy on adversarial images generated by FGSM as we vary ϵ, 2) compute the area under the curve (blue line), approximated by the area of the green region; 2) normalize 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).

Designing a measure/objective for robustness against adversarial robustness is an area of active research (e.g., [70]). For our purposes, we present a possible measure here, illustrated in Fig. 21. Using the FSGM presented by [61], this robustness objective progressively increases noise produced by FSGM. The ϵ axis in Fig. 21 refers to the hyper-parameter in the FSGM equation,

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

where $\boldsymbol{x}$ is the original image, $\boldsymbol{x}'$ 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 [23]. In each phase a maximum of size nodes may be active—where the computation at each node is 3x3 convolution followed by ReLU and batch normalization. Each network is trained for 20 epochs with SGD on a cosine annealed learning rate 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, NSGANet 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.

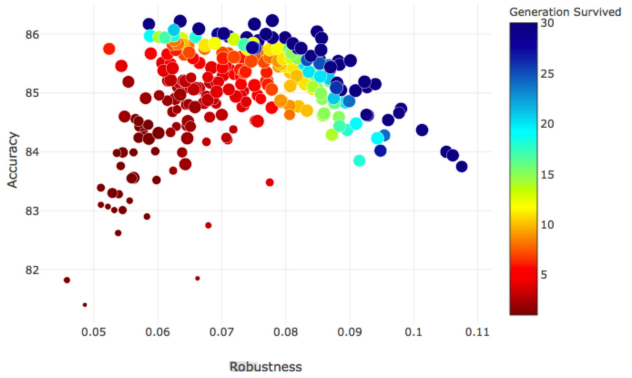


Fig. 22: Trade-off frontier of the robustness experiments. Color indicates the generation (iteration) at which a network architecture is eliminated from the surviving parent population. The size of each point is proportional to the network architecture's number of trainable parameters. We note that networks 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, NSGANet is useful in capturing patterns that differentiate architectures that are good for competing objectives. We find that the "wide" networks (like ResNeXt [52] or Inception blocks [1]) appear to provide good accuracy on standard benchmark images, but are fragile to the FSGM attack. On the other hand, "deep" networks (akin to ResNet [2] or VGG [71]) 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.

1) 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

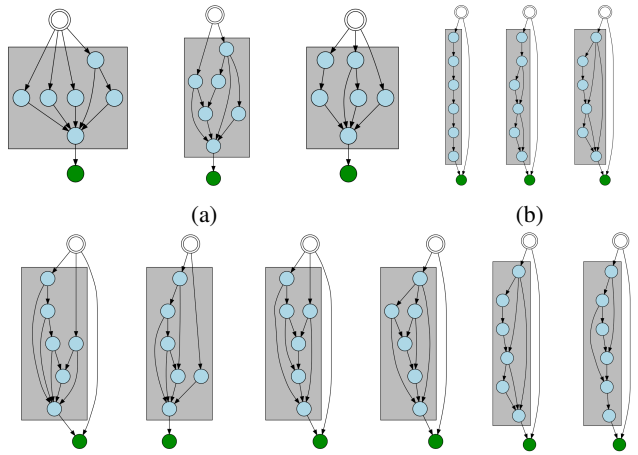


Fig. 23: (a) Examples of the computational blocks discovered with high classification accuracy. For these networks, the mean accuracy and robustness objectives are 0.8543 and 0.0535, respectively; (b) Examples of the computational blocks discovered with high robustness against FGSM attack, the mean accuracy 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.

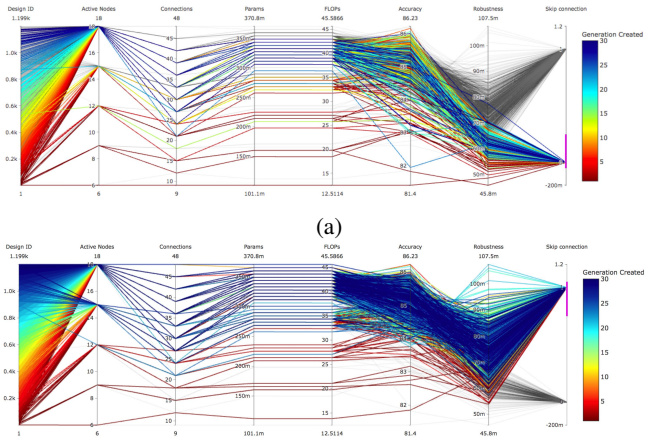


Fig. 24: Parallel coordinate plots of the 1,200 network architectures sampled by NSGANet. Each line represents a network architecture, each vertical line is an attribute associated with the network. (a) Networks that have the skip connection bit inactive, we can see that none of them have good measurement 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.

one label to the entire image in case of classification. In this section, we apply NSGANet to the problem of multi-view car key-points alignment.

We use the CMU-Car dataset originally introduced in [72]. 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 [40]. 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 [72]. We use FLOPs as the second objective for architecture complexity measurement. The obtained architectures are named as NSGANet-C0 and -C1. The obtained results are provided in Table VI and the visualization of the architectures is provided in Fig. 25.

TABLE VI: Preliminary results on the CMU-Car alignment [72]. Notably, our proposed algorithm is able to find architectures with competitive performance while having 2x less parameters when compared to human-designed architecture [73].

Architectures	Params. (M)	FLOPs (M)	Regression Error (%)
Hourglass[73]	3.38	3613	7.80
NSGANet-C0	1.53	2584	8.66
NSGANet-C1	1.61	2663	8.64

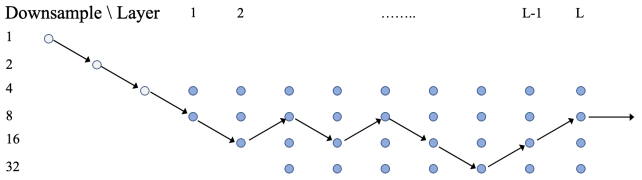


Fig. 25: Spatial-resolution-change path of the NSGANet-C0 architecture. Each circle encodes a residual module [2]. Circles colored in white are always executed at the beginning. The arrows and blues circles are parts of the search decisions. The total number of layers (L) are set to be nine.