NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm

Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar,
Kalyanmoy Deb, Erik Goodman and Wolfgang Banzhaf
Michigan State University
East Lansing, Michigan
{luzhicha,whalenia,vishnu,dhebarya,kdeb,goodman,banzhafw}@msu.edu

ABSTRACT

This paper introduces NSGA-Net – an evolutionary approach for neural architecture search (NAS). NSGA-Net is designed with three goals in mind: (1) a procedure considering multiple and conflicting objectives, (2) an efficient procedure balancing exploration and exploitation of the space of potential neural network architectures, and (3) a procedure finding a diverse set of trade-off network architectures achieved in a single run. NSGA-Net is a population-based search algorithm that explores a space of potential neural network architectures in three steps, namely, a population *initialization* step that is based on prior-knowledge from hand-crafted architectures, an *exploration* step comprising crossover and mutation of architectures, and finally an *exploitation* step that utilizes the hidden useful knowledge stored in the entire history of evaluated neural architectures in the form of a Bayesian Network. Experimental results suggest that combining the dual objectives of minimizing an error metric and computational complexity, as measured by FLOPs, allows NSGA-Net to find competitive neural architectures. Moreover, NSGA-Net achieves error rate on the CIFAR-10 dataset on par with other state-of-the-art NAS methods while using orders of magnitude less computational resources. These results are encouraging and shows the promise to further use of EC methods in various deep-learning paradigms.

KEYWORDS

Deep Learning, Image classification, Neural Architecture Search, multi objective, Bayesian Optimization

1 INTRODUCTION

Deep convolutional neural networks have been overwhelmingly successful in a variety of image analysis tasks. One of the key driving forces behind this success is the introduction of many CNN architectures, such as AlexNet [25], VGG [41], GoogLeNet [43], ResNet [14], DenseNet [19] etc. in the context of image classification. Concurrently, network designs such as MobileNet [16], XNOR-Net [38], BinaryNets [6], LBCNN [21] etc. have been developed with the goal of enabling real-world deployment of high performance models on resource constrained devices. These developments are the fruits of years of painstaking efforts and human ingenuity.

Neural architecture search (NAS) methods, on the other hand, seek to automate the process of designing network architectures. State-of-the-art reinforcement learning (RL) methods like [40] and [51] are inefficient in their use of their search space and require 3,150 and 2,000 GPU days, respectively. Gradient-based methods like [29] focus on the single objective of minimizing an error metric on a task and cannot be easily adapted to handle multiple conflicting objectives. Furthermore, most state-of-the-art approaches search over a single computation block, similar to an Inception block [43], and repeat it as many times as necessary to form a complete network.

In this paper, we present NSGA-Net, a multi-objective genetic algorithm for NAS to address the aforementioned limitations of current approaches. A pictorial overview of NSGA-Net is provided in Figure 1. The salient features of NSGA-Net are,

(1) **Multi-Objective Optimization**: Real-world deployment of NAS models demands small-sized networks, in addition the models being accurate. For instance, we seek to maximize performance on compute devices that are often constrained by hardware resources in terms of power consumption, available memory, available FLOPs, and latency constraints, to name a few. NSGA-Net is explicitly designed to optimize such competing objectives.

(2) **Flexible Architecture Search Space**: The search space for most existing methods is restricted to a block that is repeated as many times as desired. In contrast, NSGA-Net searches over the entire structure of the network. This scheme overcomes the limitations inherent in repeating the same computation block throughout an entire network, namely, that a single block may not be optimal for every application and it is desirable to allow NAS to discover architectures with different blocks in different parts of the network.

(3) **Non-Dominated Sorting**: The core component of NSGA-Net is the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [7], a multi-objective optimization algorithm that has been successfully employed for solving a variety of multi-objective problems [34, 44]. Here, we leverage its ability to maintain a diverse trade-off frontier between multiple conflicting objectives, thereby resulting in a more effective and efficient exploration of the search space.

(4) **Efficient Recombination**: In contrast to state-of-the-art evolution-based NAS methods [39, 40] in which only mutation is used, we employ crossover (in addition to mutation) to combine networks with desirable qualities across multiple objectives from the diverse frontier of solutions.

(5) **Bayesian Learning**: We construct and employ a Bayesian Network inspired by the Bayesian Optimization Algorithm (BOA) [35] to fully utilize the promising solutions present in our search history archive and the inherent correlations between the layers of the network architecture.

We demonstrate the efficacy of NSGA-Net on the CIFAR10 [24] image classification task by minimizing two objectives: classification
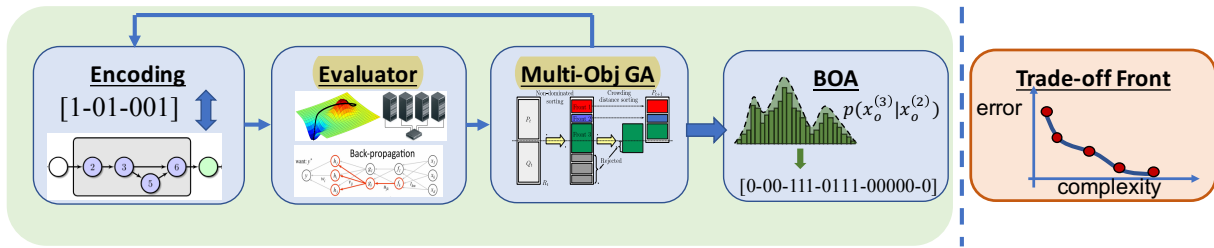
Figure 1: Overview of the stages of NSGA-Net. Networks are represented as bit strings, trained through gradient descent, ranking and selection by NSGA-II, search history exploitation through BOA. Output is a set of networks that span a range of complexity and error objectives.

error and computational complexity. Here, computational complexity is defined by the number of floating-point operations (FLOPs) that a network carries out during a forward pass. Experimentally, we observe that NSGA-Net can find a set of network architectures containing solutions that are significantly better than hand-crafted methods in both objectives, while being competitive with single objective state-of-the-art NAS approaches. Furthermore, by fully utilizing a population of networks through recombination and utilization of the search history, NSGA-Net explores the search space efficiently and requires less computational time for search than other competing methods. The implementation of NSGA-Net is available here*.

2 RELATED WORK

Recent research efforts in NAS have produced a plethora of methods to automate the design of networks. Broadly speaking, these methods can be divided into evolutionary algorithm (EA) and reinforcement learning (RL) based approaches – with a few methods falling outside these two categories. The main motivation of EA methods is to treat structuring a network as a combinatorial optimization problem. EAs operate with a population that makes small changes (mutation) and mixes parts (crossover) of solutions selected by consideration of multiple objectives to guide its search toward the optimal solutions. RL, on the other hand, views the construction of a network as a decision process. Usually, an agent is trained to optimally choose the pieces of a network in a particular order. We briefly review a few existing methods here.

Reinforcement Learning: Q-learning [45] is a widely popular value iteration method used for RL. The MetaQNN method [1] 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. [49] 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. This method allows for a much more general network and achieves better results than its predecessor on CIFAR-10 [24].

A policy gradient method seeks to approximate some not differentiable reward function to train a model that requires parameter gradients, like a neural network architecture. Zoph and Le [50] first applied this method in architecture search to train a recurrent neural

network controller that constructs networks. The original method in [50] uses the controller to generate the entire network at once. This contrasts from its successor, NASNet [51], 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 error rate on CIFAR-10. NSGA-Net differs from RL methods by using more than one selection criteria. More specifically, networks are selected for their accuracy on a task, rather than an approximation of accuracy, along with computational complexity. Furthermore, the most successful RL methods search only a computational block that is repeated to create a network, NSGA-Net allows for search across computational blocks and combinations of blocks. Hsu et al. [17] extends the NASNet approach to multi-objective domain to optimize multiple linear combinations of accuracy and energy consumption criteria for different scalarization parameters. However, multiple generative applications of a scalarized objectives was shown to be not as efficient as simultaneous approaches [7].

Evolutionary Algorithms: Designing neural networks through evolution, or *neuroevolution*, has been a topic of interest for some time, first showing popular success in 2002 with the advent of the neuroevolution of augmenting topologies (NEAT) algorithm [42]. In its original form, NEAT only performs well on comparatively small networks. Miikkulainen et al. [32] attempted to extend NEAT to deep networks with CoDeepNEAT using a co-evolutionary approach that achieved limited results on the CIFAR-10 dataset. CoDeepNEAT does, however, produce state-of-the-art results in the Omniglot multi-task learning domain [26].

Real et al. [40] introduced perhaps the first truly large scale application of a simple evolutionary algorithm. The extension of this method presented in [39], called AmoebaNet, provided the first large scale comparison of EC and RL methods. Their simple EA searches over the same space as NASNet [51] and has shown to have a faster convergence to an accurate network when compared to RL and random search. Furthermore, AmoebaNet produced one of the best state-of-the-art results on CIFAR-10 data-set.

Conceptually, NSGA-Net is closest to the Genetic CNN [47] algorithm. It uses a binary encoding that corresponds to connections in convolutional blocks. In NSGA-Net, we augment the original encoding and genetic operations by (1) adding an extra bit for a residual connection, and (2) introducing phase-wise crossover. We also introduce a multi-objective based selection scheme. Moreover, we also diverge from Genetic CNN by incorporating a Bayesian

---
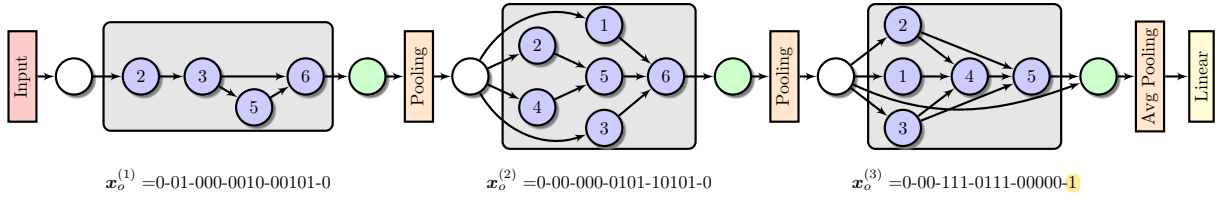
Figure 2: Encoding: Illustration of a classification network encoded by $\mathbf{x} = \mathbf{x_o}$, where x_o is the operations at a phase (gray boxes, each with a possible maximum of 6 nodes). In this example the spatial resolution changes (orange boxes that connect the phases) are fixed based on prior knowledge of successful approaches. The phases are described by the bit string $\mathbf{x_o}$ which is formatted for readability above. The bits are grouped by dashes to describe what node they control. See Section 3.1 for detailed description of the encoding schemes.

network in our search to fully utilize past population history as learned knowledge.

Evolutionary multi-objective optimization (EMO) approaches have been scarcely used for NAS. Kim et al. [23] presents an algorithm utilizing NSGA-II [7], however their method only searches over hyper-parameters and a small fixed set of architectures. The evolutionary method shown in [10] uses weight sharing through network morphisms [46] and approximate morphisms as mutations and uses a biased sampling to select for novelty from the objective space rather than a principled selection scheme, like NSGA-II [7]. Network morphisms allow for a network to be "widened" or "deepened" in a manner that maintains functional equivalence. For architecture search, this allows for easy parameter sharing after a perturbation in a network's architecture.

Other Methods: Methods that do not subscribe to either an EA or RL paradigm have also shown success in architecture search. Liu et al. [27] presents 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. Dong et al. [9] extended this method to use a multi-objective approach, selected the K networks based on their Pareto-optimality when compared to other networks. Hsu et al. [18] also presents a meta-modeling approach that generates models with state-of-the-art accuracy. This approach may be ad-hoc as no analysis is presented on how the progressive search affects the trade-off frontier. Elsken et al. [11] use a simple hill climbing method along with a network morphism [46] approach to optimize network architectures quickly on limited resources. Chen et al. [5] combine the ideas of RL and EA. A population of networks is maintained and are selected for mutation with tournament selection [13]. A recurrent network is used as a controller to learn an effective strategy to apply mutations to networks. Networks are then trained and the worst performing network in the population is replaced. This approach generates state of the art results for the ImageNet classification task. Chen et al. [3] presented an augmented random search approach to optimize networks for a semantic segmentation application. Kandasamy et al. [22] presents a Gaussian process based approach to optimize network architectures, viewing the process through a Bayesian optimization lens.

3 PROPOSED APPROACH

Compute devices are often constrained by hardware resources in terms of their power consumption, available memory, available FLOPs, and latency constraints. Hence, real-world design of DNNs are required to balance these multiple objectives (e.g., predictive performance and computational complexity). Often, when multiple design criteria are considered simultaneously, there may not exist a single solution that performs optimally in all desired criteria, especially with competing objectives. Under such circumstances, a set of solutions that provide representative trade-off information between the objectives is more desirable. This enables a practitioner to analyze the importance of each criterion, depending on the application, and to choose an appropriate solution on the trade-off frontier for implementation. We propose NSGA-Net, a genetic algorithm based architecture search method to automatically generate a set of DNN architectures that approximate the Pareto-front between performance and complexity on an image classification task. The rest of this section describes the encoding scheme, and main components of NSGA-Net in detail.

3.1 Encoding

Genetic algorithms, like any other biologically inspired search methods, often do not directly operate on *phenotypes*. From the biological perspective, we may view the DNN architecture as a phenotype, and the representation it is mapped from as its *genotype*. As in the natural world, genetic operations like crossover and mutation are only carried out in the genotype space; such is the case in NSGA-Net as well. We refer to the interface between the genotype and the phenotype as *encoding* in this paper.

Most existing CNN architectures can be viewed as a composition of computational blocks that define the layer-wise computation (e.g. ResNet blocks [14], DenseNet block [19], and Inception block [43], etc.) and a scheme that specifies the spatial resolution changes. For example, down-sampling is often used after computational blocks to reduce the spatial resolution of information going into the next computational blocks in image classification DNNs. In NSGA-Net, each computational block, referred to as a *phase*, is encoded using the method presented by Xie and Yuille [47], with the small change of adding a bit to represent a skip connection that forwards the input information directly to the output bypassing the entire block. And we name it as the *Operation Encoding* $\mathbf{x_o}$ in this study.

Operation Encoding $\mathbf{x_o}$: Unlike most of the hand-crafted and NAS generated architectures, we do not repeat the same phase (computational block) to construct a network. Instead, the operations of a network are encoded by $\mathbf{x_o} = \left(\mathbf{x_o^{(1)}}, \mathbf{x_o^{(2)}}, \ldots, \mathbf{x_o^{(n_p)}} \right)$ where n_p is
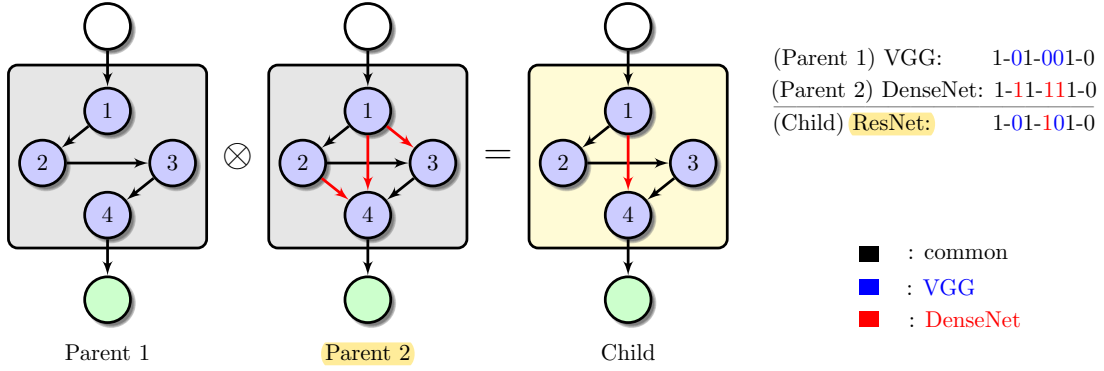
Figure 3: Crossover Example: A crossover (denoted by ⊗) of a VGG-like structure with a DenseNet-like structure may result in a ResNet-like network. In the figure, red and blue denotes connections that are unique to VGG and DenseNet respectively, and black shows the connections that are common to both parents. All black bits are retained in the final child encoding, and only the bits that are not common between the parents can potentially be selected at random from one of the parent.

the number of phases. Each $\mathbf{x_o^{(i)}}$ encodes a directed acyclic graph consisting of n_o number of nodes that describes the operation within a phase using a binary string. Here, a node is a basic computational unit, which can be a single operation like convolution, pooling, batch-normalization [20] or a sequence of operations. This encoding scheme offers a compact representation of the network architectures in genotype space, yet is flexible enough that many of the computational blocks in hand-crafted networks can be encoded, e.g. VGG [41], ResNet [14] and DenseNet [19]. Figure 2 and Figure 3 shows examples of the operation encoding.

Search Space: With a pre-determined scheme of spatial resolution reduction (similarly in [29, 39, 51]), the total search space in the genotype space is governed by our operation encoding $\mathbf{x_o}$:

$$\Omega_{\mathbf{x}} = \Omega_{\mathbf{x_o}} = n_p \times 2^{n_o(n_o-1)/2+1}$$

where n_p is the number of phases (computational blocks), and n_o is the number of nodes (basic computational units) in each phase. However, for computationally tractability, we constrain the search space such that each node in a phase carries the same sequence of operations, i.e. a 3×3 convolution followed by batch-normalization [20] and ReLU.

It is worth noting that, as a result of nodes in each phase having identical operations, the encoding between genotype and phenotype is a many-to-one mapping. Given the prohibitive computational expense required to train each network architecture before its performance can be assessed, it is essential to avoid evaluating genomes that decode to the same architecture. We develop an algorithm to quickly and approximately identify these duplicate genomes (see Appendix for details).

3.2 Search Procedure

NSGA-Net is an iterative process in which initial solutions are made gradually better as a group, called a *population*. In every iteration, the same number of offspring (new network architectures) are generated from parents selected from the population. Each population member (including both parents and offspring) compete for both survival and reproduction (becoming a parent) in the next iteration. The

initial population may be generated randomly or guided by prior-knowledge (e.g. seeding the hand-crafted network architectures into the initial population). Following initialization, the overall NSGA-Net search proceeds in two sequential stages, an *exploration* and *exploitation*.
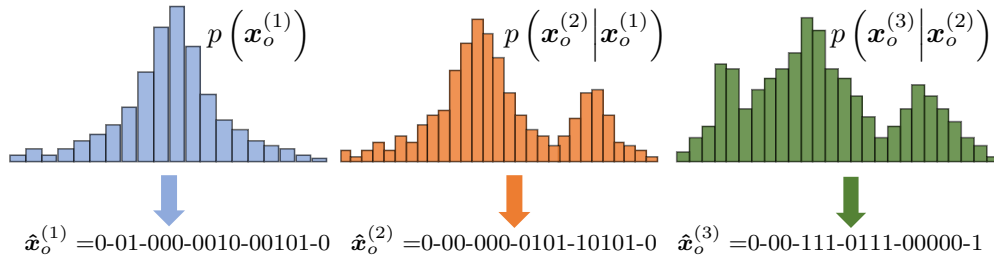
Exploration: The goal of this stage is to discover diverse ways of connecting nodes to form a phase (computational block). Genetic operations, crossover and mutation, offer an effective mean to realize this goal.

Crossover: The *implicit* parallelism of population-based search approaches can be unlocked when the population members can effectively share (through crossover) building-blocks [15]. In the context of NAS, a phase or the sub-structure of a phase can be viewed as a building-block. We design a homogeneous crossover operator, which takes two selected population members as parents, to create offspring (new network architectures) by inheriting and recombining the building-blocks from parents. The main idea of this crossover operator is to 1) preserve the common building-blocks shared between both parents by inheriting the common bits from both parents' binary bit-strings; 2) maintain, relatively, the same complexity between the parents and their offspring by restricting the number of "1" bits in the offspring's bit-string to lie between the number of "1" bits in both parents. The proposed crossover allows selected architectures (parents) to effectively exchange phases or sub-structures within a phase. An example of the crossover operator is provided in Figure 3.

Mutation: To enhance the diversity (having different network architectures) of the population and the ability to escape from local optima, we use a bit-flipping mutation operator, which is commonly used in binary-coded genetic algorithms. Due to the nature of our encoding, a one bit flip in the genotype space could potentially create a completely different architecture in the phenotype space. Hence, we restrict the number of bits that can be flipped to be at most one for each mutation operation. As a result, only one of the phase architectures can be mutated at one time.

Exploitation: The exploitation stage follows the exploration stage in NSGA-Net. The goal of this stage is to exploit and reinforce the

$$p\left(\boldsymbol{x}_o^{(1)}\right) \qquad p\left(\boldsymbol{x}_o^{(2)}\middle|\boldsymbol{x}_o^{(1)}\right) \qquad p\left(\boldsymbol{x}_o^{(3)}\middle|\boldsymbol{x}_o^{(2)}\right)$$

$\hat{\boldsymbol{x}}_o^{(1)} =$0-01-000-0010-00101-0 $\quad \hat{\boldsymbol{x}}_o^{(2)} =$0-00-000-0101-10101-0 $\quad \hat{\boldsymbol{x}}_o^{(3)} =$0-00-111-0111-00000-1

Figure 4: Exploitation: Sampling from the Bayesian Network (BN) constructed by NSGA-Net. The histograms represent estimates of the conditional distributions between the network structure between the phases explored during the exploration step and updated during the exploitation step (i.e., using the population archive). During exploitation, networks are constructed by sampling phases from the BN. Fig. 2 shows the architectures that the sampled bit strings, $\{\hat{\boldsymbol{x}}_o^{(1)}, \hat{\boldsymbol{x}}_o^{(2)}, \hat{\boldsymbol{x}}_o^{(3)}\}$ decode to.

patterns commonly shared among the past successful architectures explored in the previous stage. The exploitation step in NSGA-Net is heavily inspired by the Bayesian Optimization Algorithm (BOA) [35] 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 blocks and paths across the different phases. Exploitation uses past information across all networks evaluated to guide the final part of the search. More specifically, say we have a network with three phases, namely $\mathbf{x_o^{(1)}}$, $\mathbf{x_o^{(2)}}$, and $\mathbf{x_o^{(3)}}$. We would like to know the relationship of the three phases. For this purpose, we construct a Bayesian Network (BN) relating these variables, modeling the probability of networks beginning with a particular phase $\mathbf{x_o^{(1)}}$, the probability that $\mathbf{x_o^{(2)}}$ follows $\mathbf{x_o^{(1)}}$, and the probability that $\mathbf{x_o^{(3)}}$ follows $\mathbf{x_o^{(2)}}$. In other words, we estimate the distributions $p\left(\mathbf{x_o^{(1)}}\right)$, $p\left(\mathbf{x_o^{(2)}}|\mathbf{x_o^{(1)}}\right)$, and $p\left(\mathbf{x_o^{(3)}}|\mathbf{x_o^{(2)}}\right)$ by using the population history, and update these estimates during the exploitation process. New offspring solutions are created by sampling from this BN. Figure 4 shows a pictorial depiction of this process.

4 EXPERIMENTS

In this section, we explain the experimental setup and implementation details of NSGA-Net, followed by the empirical results to demonstrate the efficacy of NSGA-Net to automate the NAS process on image classification task.

4.1 Performance Metrics

We consider two objectives to guide NSGA-Net based NAS, namely, classification error and computational complexity. A number of metrics can serve as proxies for computational complexity: 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 network. Our initial experiments considered each of these different metrics. We concluded from extensive experimentation that inference time cannot be estimated reliably due differences and inconsistencies in computing environment, GPU manufacturer, temperature, etc. Similarly, the number of parameters, active connections or active nodes only relate to one aspect of computational complexity. In

contrast, we found an estimate of FLOPs to be a more accurate and reliable proxy for network complexity. See Appendix for more details. Therefore, classification error and FLOPs serve as the twin objectives for selecting networks.

For the purpose of quantitatively comparing different multi - objective search methods or different configuration setups of NSGA-Net, we use the hypervolume (HV) performance metric, which calculates the dominated area (hypervolume in the general case) from the a set of solutions (network architectures) to a reference point which is usually an estimate of the nadir point—a vector concatenating worst objective values of the Pareto-frontier. It has been proven that the maximum HV can only be achieved when all solutions are on the Pareto-frontier [12]. Hence, the higher the HV measures, the better solutions that are being found in terms of both objectives.

4.2 Implementation Details

Dataset: We consider the CIFAR-10 [24] dataset for our classification task. We split the original training set (80%-20%) to create our training and validation sets for architecture search. The original CIFAR-10 testing set is only utilized at the conclusion of the search to obtain the test accuracy for the models on the final trade-off front.

NSGA-Net hyper-parameters: We set the number of phases n_p to three and the number of nodes in each phase n_o to six. We also fix the spatial resolution changes scheme similarly as in [51], in which a max-pooling with stride 2 is placed after the first and the second phase, and a global average pooling layer after the last phase. The initial population is generated by uniform random sampling. The probabilities of crossover and mutation operations are set at 0.9 and 0.02 respectively. The population size is 40 and the number of generations is 20 for the exploration stage. And another ten generations for exploitation. Hence, a total of 1,200 network architectures are searched by NSGA-Net.

Network training during searching: During architecture search, we limit the number of filters (channels) in any node to 16 for each one of the generated network architecture. We then train them on our training set using standard stochastic gradient descent (SGD) back-propagation algorithm and a cosine annealing learning rate schedule [30]. Our initial learning rate is 0.025 and we train for 25 epochs, which takes about 9 minutes on a NVIDIA 1080Ti GPU

implementation in PyTorch [33]. Then the classification error is measured on our validation set.

4.3 Architecture Validation

For comparing with other single-objective NAS methods, we adopt the training procedure used in [29] and a quick summary is given as follows.

We extend the number of epochs to 600 with a batch-size of 96 to train the final selected models (could be the entire trade-off frontier architectures or a particular one chosen by the decision-maker). We also incorporate a data pre-processing technique *cutout* [8], and a regularization technique *scheduled path dropout* introduced in [51]. In addition, to further improve the training process, an auxiliary head classifier is appended to the architecture at approximately 2/3 depth (right after the second resolution-reduction operation). The loss from this auxiliary head classifier, scaled by a constant factor 0.4, is aggregated with the loss from the original architecture before back-propagation during training. Other hyper-parameters related to the back-propagation training remain the same as during the architecture search.

For the fairness of the comparison among various NAS methods, we incorporate the NASNet-A cell [51], the AmoebaNet-A cell [39] and the DARTS(second order) cell [29] into our training procedures and report their results under the same settings as NSGA-Net found architectures.

4.4 Results Analysis

We first present the overall search progression of NSGA-Net in the objective-space. Figure 5 shows the bi-objective frontiers obtained by NSGA-Net through the various stages of the search, clearly showcasing a gradual improvement of the whole population. Figure 6 shows two metrics: normalized HV and offspring survival rate, through the different generations of the population. The monotonic increase in the former suggests that a better set of trade-off network architectures have been found over the generations. The monotonic decrease in the latter metric suggests that, not surprisingly, it is increasingly difficult to create better offspring (than their parents). We can use a threshold on the offspring survival rate as a potential criterion to terminate the current stage of the search process and switch between the exploration and exploitation.

To compare the network architecture obtained from NSGA-Net to other hand-crafted and search-generated architectures, we pick the network architectures with the lowest classification error from the final frontier (the dot in the lower right corner on the green curve in Figure 5) and extrapolate (by following the setup as explained in Section 4.3) the network by increasing the number of filters of each node in the phases, and train with the entire official CIFAR-10 training set. The chosen network architecture, shown in Figure 2, results in 3.85% classification error on the CIFAR-10 testing set with 3.34 Millions of parameters and 1290 MFLOPs. Table 1 provides a summary that compares NSGA-Net with other multi-objective NAS methods. Unfortunately, hypervolume comparisons between these multi-objective NAS methods are not feasible due to the following two reasons: 1) The entire trade-off frontiers obtained by the other multi-objective NAS methods are not reported and 2) different objectives were used to estimate the complexity of the architectures. Due
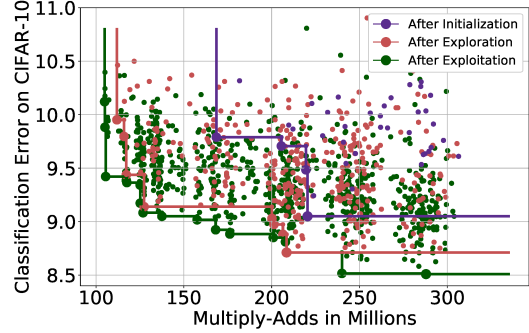

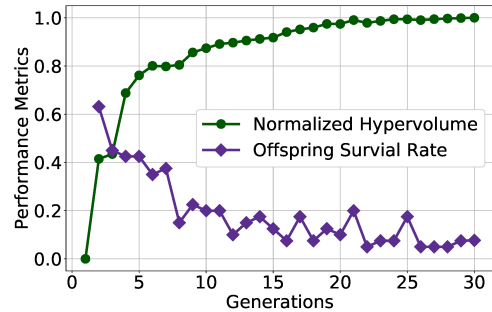**Figure 5: Progression of trade-off frontiers after each stage of NSGA-Net.**

Figure 6: Generational normalized hypervolume and survival rate of the offspring network architectures.

Table 1: Multi-objective methods for CIFAR-10 (best accuracy for each method)

Method	Error (%)	Other Objective	Compute
PPP-Net [9]	4.36	FLOPs or Params or Inference Time	Nvidia Titan X
MONAS [17]	4.34	Power	Nvidia 1080Ti
NSGA-Net	3.85	FLOPs	Nvidia 1080Ti 8 GPU Days

to space limitation, the other architectures on the trade-off frontier found by NSGA-Net are reported in Appendix.

The CIFAR-10 results comparing state-of-the-art CNN architectures from both human-designed and search-generated are presented in Table 2. NSGA-Net achieves comparable results with state-of-the-art architectures designed by human experts [19] while having order of magnitude less parameters in the obtained network architecture. When compared with other state-of-the-art RL- and evolution-based NAS methods [39, 51], NSGA-Net achieves similar performance by using two and half orders of magnitude less computation resources (GPU-days). Even though NSGA-Net falls short in search efficiency when compared to the gradient-based NAS method DARTS [29]
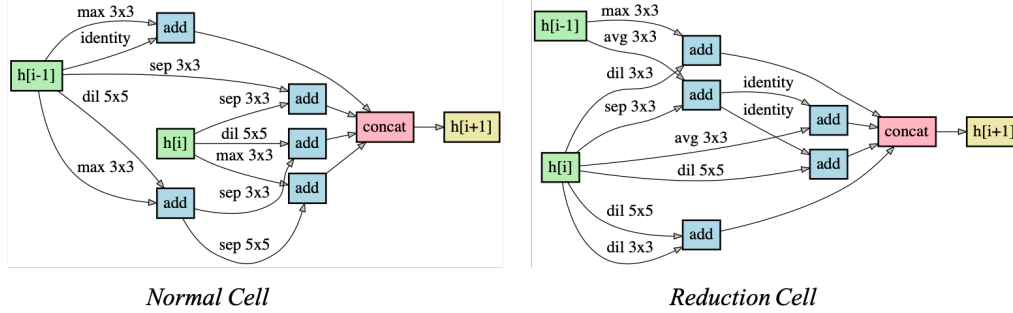
Normal Cell   *Reduction Cell*

Figure 7: Normal and reduction convolutional cell architectures found by NSGA-Net applied to NASNet micro search space. The inputs (green) are from previous cells' output (or input image). The output (yellow) is the results of a concatenation operation across all resulting branches. Each edge (line with arrow) indicates an operation with operation name annotated above the line.

Table 2: Comparison of NSGA-Net with baselines on CIFAR-10 image classification. In this table, the first block presents state-of-the-art architectures designed by human experts. The second block presents NAS methods that design the entire network. The last block presents NAS methods that design modular blocks which are repeatedly combined to form the final architecture. We use (N @ F) to indicate the configuration of each model, where N is the number of repetition and F is the number of filters right before classification. Results marked with † are obtained by training the corresponding architectures with our setup (refers to Section 4.3 for details).

Architectures	Params (M)	Test Error (%)	$\times +$ (M)	Search Cost (GPU-days)	Search Method
Wide ResNet [48]	36.5	4.17	-	-	human experts
DenseNet-BC (k = 40) [19]	25.6	3.47	-	-	human experts
NAS [50]	7.1	4.47	-	3150	RL
NAS + more filters[50]	37.4	3.65	-	3150	RL
ENAS + macro search space [36]	21.3	4.23	-	0.5	RL + weight sharing
ENAS + macro search space + more channels [36]	38.0	3.87	-	0.5	RL + weight sharing
NSGA-Net + macro search space	**3.3**	**3.85**	**1290**	**8**	**evolution**
DARTS second order + cutout [29]	3.3	2.76	-	4	gradient-based
DARTS second order (6 @ 576) + cutout [29] †	3.3	2.76	547	4	gradient-based
NASNet-A + cutout [51]	3.3	2.65	-	2,000	RL
NASNet-A (6 @ 660) + cutout [51] †	3.2	2.91	532	2,000	RL
ENAS + cutout [36]	4.6	2.89	-	0.5	RL + weight sharing
ENAS (6 @ 660) + cutout [36] †	3.3	2.75	533	0.5	RL + weight sharing
AmoebaNet-A [39]	3.2	3.34	-	3,150	evolution
AmoebaNet-A (6 @ 444) + cutout [39] †	3.3	2.77	533	3,150	evolution
NSGA-Net (6 @ 560) + cutout	**3.3**	**2.75**	**535**	**4**	**evolution**
NSGA-Net (7 @ 1536) + cutout	**26.8**	**2.50**	**4147**	**4**	**evolution**

despite a slight advantage in test error, it's worth noting that NSGA-Net inherently delivers many other architectures from the trade-off frontier at no extra cost. The corresponding architectures found by NSGA-Net are provided in Figure 2 and Figure 7 for macro search space and NASNet micro search space respectively.

4.5 Transferability

We consider CIFAR-100 dataset [24] for evaluating the tranferability of the found architecture by NSGA-Net. We use the same training setup as explained in Section 4.3 on CIFAR-10 dataset. The training takes about 1.5 days on a single 1080Ti GPU. Results shown in Table 3 suggest that the learned architecture from searching on CIFAR-10 is transferable to CIFAR-100. The architecture found by NSGA-Net achieves comparable performance to both the human-designed and RL-search generated architectures [48, 49] with 10x and 2x less number of parameters respectively.
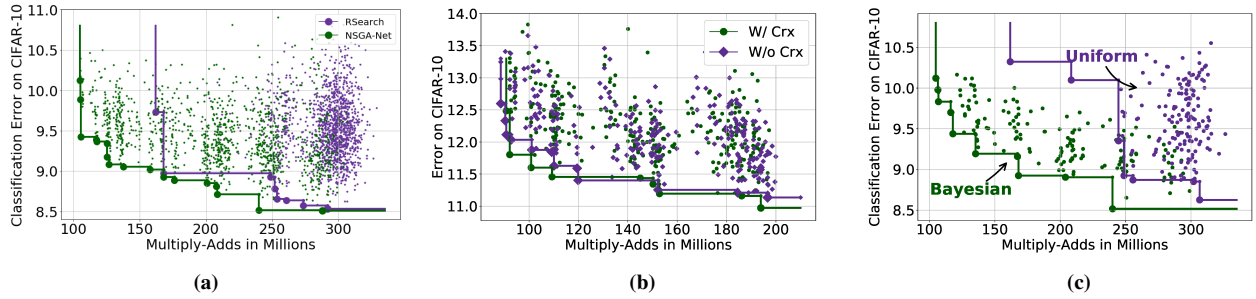
Figure 8: (a) Trade-off frontier comparison between random search and NSGA-Net. (b) Trade-off frontier comparison with and without crossover. (c) Comparison between sampling from uniformly from the encoding space and the Bayesian Network constructed from NSGA-Net exploration population archive.

Table 3: Comparison with different classifiers on CIFAR-100.

Architectures	Params (M)	Test Error (%)	GPU Days	Search Method
Wide ResNet [48]	36.5	20.50	-	manual
Block-QNN-S [49]	6.1	20.65	96	RL
Block-QNN-S* [49]	39.8	18.06	96	RL
NSGA-Net	3.3	20.74	8	evolution
NSGA-Net*	11.6	19.83	8	evolution

*denotes architectures with extended number of filters

4.6 Ablation Studies

Here, we first present results comparing NSGA-Net with uniform random sampling (RSearch) from our encoding as a sanity check. It's clear from Figure 8a that much better set of network architectures are obtained using NSGA-Net. Then we present additional results to showcase the benefits of the two main components of our approach: crossover and Bayesian network based offspring creation.

Crossover Operator: Current state-of-the-art NAS search results [28, 39] using evolutionary algorithms use mutation alone with enormous computation resources. We quantify the importance of crossover operation in an EA by conducting the following small-scale experiments on CIFAR-10. From Figure 8b, we observe that crossover helps achieve a better trade-off frontier.

Bayesian Network (BN) based Offspring Creation: Here we quantify the benefits of the exploitation stage i.e., off-spring creation by sampling from BN. We uniformly sampled 120 network architectures each from our encoding and from the BN constructed on the population archive generated by NSGA-Net at the end of exploration. The architectures sampled from the BN dominate (see Fig.8c) all network architectures created through uniform sampling.

4.7 Discussion

We analyze the intermediate solutions of our search and the trade-off frontiers and make some observations. Upon visualizing networks, like the one in Figure 2, we observe that as network complexity decreases along the front, the search process gravitates towards reducing the complexity by minimizing the amount of processing at higher image resolutions i.e., remove nodes from the phases that are closest to the input to the network. As such, NSGA-Net outputs a set of network architectures that are optimized for wide range of complexity constraints. On the other hand, approaches that search over a single repeated computational block can only control the complexity of the network by manually tuning the number of repeated blocks used. Therefore, NSGA-Net provides a more fine-grained control over the two objectives as opposed to the control afforded by arbitrary repetition of blocks. Moreover, some objectives, for instance susceptibility to adversarial attacks, may not be easily controllable by simple repetition of blocks. A subset of networks discovered on the trade-off frontier for CIFAR-10 is provided in Appendix.

5 CONCLUSIONS

This paper presented NSGA-Net, a multi-objective evolutionary approach for neural architecture search. NSGA-Net affords a number of practical benefits: (1) the design of neural network architectures that can effectively optimize and trade-off multiple competing objectives, (2) advantages afforded by population-based methods being more effective than optimizing weighted linear combination of objectives, (3) more efficient exploration and exploitation of the search space through a customized crossover scheme and leveraging the entire search history through BOA, and finally (4) output a set of solutions spanning a trade-off front in a single run. Experimentally, by optimizing both prediction performance and computational complexity NSGA-Net finds networks that are significantly better than handcrafted networks on both objectives and is compares favorably to other state-of-the-art single objective NAS methods for classification on CIFAR-10.

6 APPENDIX

6.1 Duplicate Checking and Removal

Due to the directed acyclic nature of our encoding, redundancy exists in the search space defined by our coding, meaning that there exist
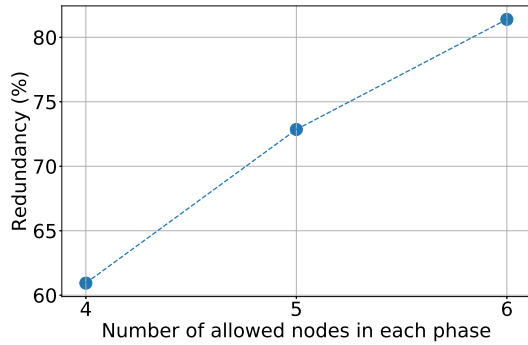
Figure 9: Increase in redundancy as node count increases.

Figure 10: Examples of different encoding bit strings that decode to the same network computation block.

multiple encoding strings that decode to the same network architecture. Empirically, we have witnessed the redundancy becomes more and more severe as the allowed number of nodes in each phase's computational block increase, as shown in Figure 9.

Since the training of a deep network is a computationally taxing task, it is essential to avoid the re-computation of the same architecture. In this section, we will provide with an overview of an algorithm we developed to quickly and approximately do a *duplicate-check* on genomes. The algorithm takes two genomes to be compared as an input, and outputs a *flag* to indicate if the supplied genomes decode to same architecture.

In general, comparing two graphs is NP-hard, however, given that we are working with Directed Acyclic Graphs with every node being the same in terms of operations, we were able to design an efficient network architecture duplicate checking method to identify most of the duplicates if not all. The method is built on top of simply intuition that under such circumstances, the duplicate network architectures should be identified by swapping the node numbers. Examples are provided in Figure 10. Our duplicates checking method first derive the connectivity matrix from the bit-string, which will have positive 1 indicating there is an input to that particular node and negative 1 indicating an output from that particular node. Then a series row-and-column swapping operation takes place, which essentially try to shuffle the node number to check if two connectivity matrix can be exactly matched. Empirically, we have found this method performs very efficiently in identifying duplicates. An example of different operation encoding bit-strings decode to the same network phase is provided in Figure 10.

6.2 Architecture Complexity Estimation

We argue that the choice of inference time or number of parameters as proxies for computational complexity are sub-optimal and ineffective in practice. In fact, we initially considered both of these objectives. We concluded from extensive experimentation that inference time cannot be estimated reliably due differences and inconsistencies in computing environment, GPU manufacturer, and GPU temperature etc. Similarly, the number of parameters only relates one aspect of computational complexity. Instead, we chose to use the number of floating-point operations (FLOPs) for our second objective. The following table compares the number of active nodes, the number of
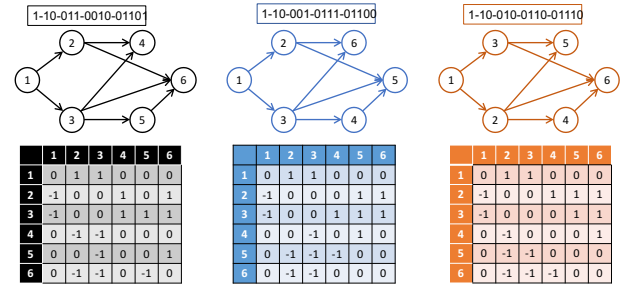
connections, the total number of parameters and the FLOPs over a few sampled architecture building blocks. See Table 4 for examples of these calculations.

REFERENCES

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *ICLR*.

[2] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient Architecture Search by Network Transformation. In *AAAI*.

[3] L.-C. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. 2018. Searching for Efficient Multi-Scale Architectures for Dense Image Prediction. *arXiv preprint arXiv:1809.04184* (Sep 2018). arXiv:cs.CV/1809.04184

[4] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille. 2014. Detect What You Can: Detecting and Representing Objects Using Holistic Models and Body Parts. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1979–1986. https://doi.org/10.1109/CVPR.2014.254

[5] Y. Chen, Q. Zhang, C. Huang, L. Mu, G. Meng, and X. Wang. 2018. Reinforced Evolutionary Neural Architecture Search. *ArXiv e-prints* (Aug 2018). arXiv:1808.00193

[6] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[7] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International Conference on Parallel Problem Solving From Nature*. Springer, 849–858.

[8] Terrance DeVries and Graham W Taylor. 2017. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552* (2017).

[9] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. PPP-Net: Platform-aware Progressive Search for Pareto-optimal Neural Architectures. In *ICLR*.

[10] T. Elsken, J. Hendrik Metzen, and F. Hutter. 2018. Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. *arXiv preprint arXiv:1804.09081* (April 2018). arXiv:stat.ML/1804.09081

[11] T. Elsken, J.H. Metzen, and F. Hutter. 2018. Simple and efficient architecture search for Convolutional Neural Networks. In *ICLR*.

[12] M. Fleischer. 2003. The measure of Pareto optima: Applications to multi-objective optimization. In *EMO*.

[13] David E. Goldberg and Kalyanmoy Deb. 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. Foundations of Genetic Algorithms, Vol. 1. Elsevier, 69 – 93. https://doi.org/10.1016/B978-0-08-050684-5.50008-2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

[15] J. H. Holland. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: MIT Press.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[17] C.-H. Hsu, S.-H. Chang, D.-C. Juan, J.-Y. Pan, Y.-T. Chen, W. Wei, and S.-C. Chang. 2018. MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning. *arXiv preprint arXiv:1806.10332* (June 2018). arXiv:1806.10332
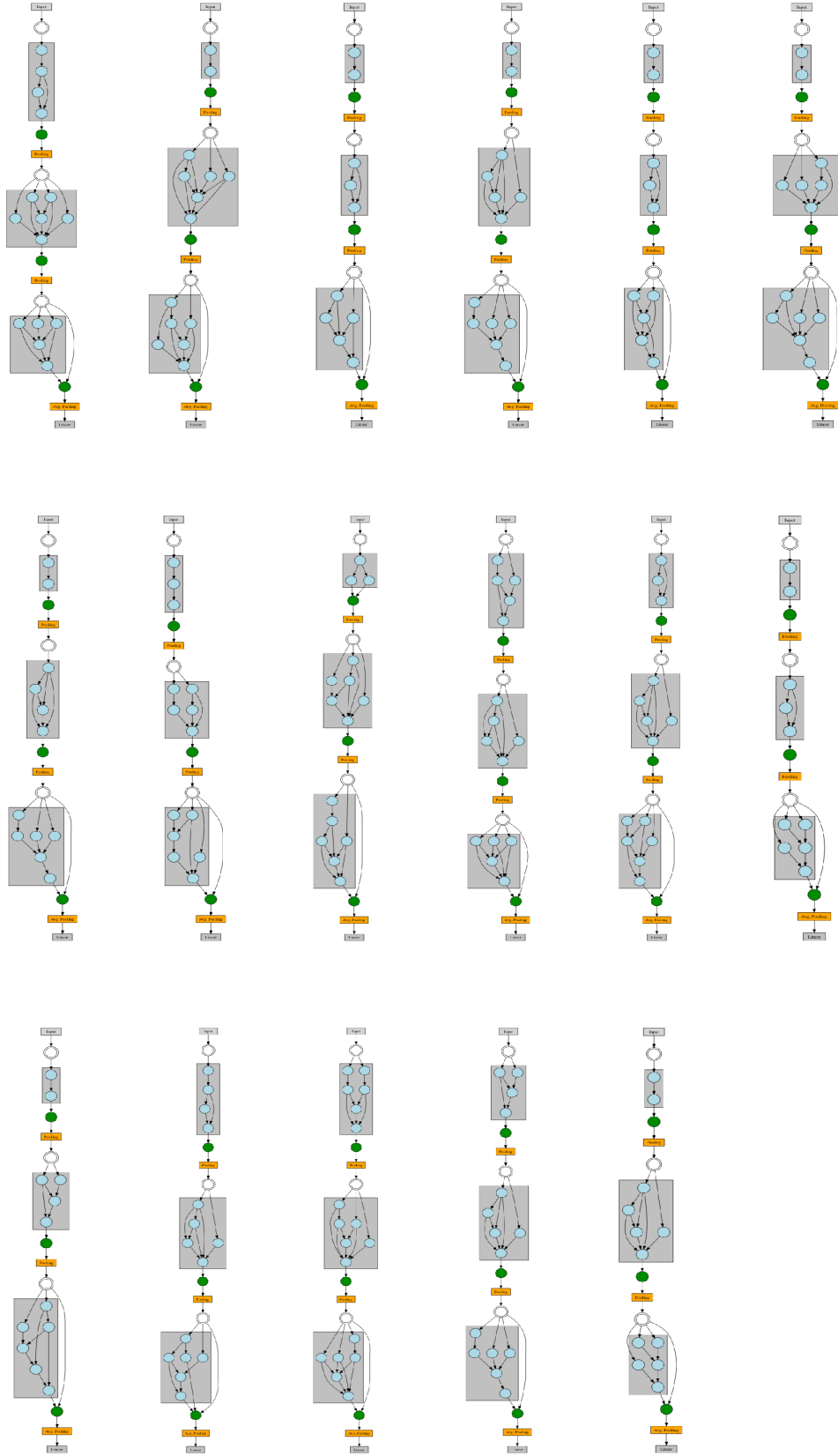
Figure 11: Set of networks architectures on the trade-off frontier discovered by NSGA-Net.

Table 4: Network examples comparing the number of active nodes, number of connections, number of parameters and number of multiply-adds.

Phase Architectures	# of Nodes	# of Conns	Params. (K)	FLOPs (M)
	3	4	113	101
	4	6	159	141
	4	7	163	145
	5	9	208	186
	5	10	216	193
	6	13	265	237

[18] C.-H. Hsu, S.-H. Chang, D.-C. Juan, J.-Y. Pan, Y.-T. Chen, W. Wei, and S.-C. Chang. 2018. Neural Architecture Optimization. *arXiv preprint arXiv:1808.07233* (Aug 2018). arXiv:1808.07233

[19] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *CVPR*.

[20] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[21] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. 2017. Local binary convolutional neural networks. In *CVPR*.

[22] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. *arXiv preprints arXiv:1802.07191* (Feb 2018). arXiv:1802.07191

[23] Y.H. Kim, B. Reddy, S. Yun, and C. Seo. 2017. NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *JMLR: Workshop and Conference Proceedings*, Vol. 1. 1–8.

[24] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n. d.]. CIFAR-10 (Canadian Institute for Advanced Research). ([n. d.]). http://www.cs.toronto.edu/~kriz/cifar.html

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*.

[26] J. Liang, E. Meyerson, and R. Miikkulainen. 2018. Evolutionary Architecture Search For Deep Multitask Networks. *ArXiv e-prints* (March 2018). arXiv:1803.03745

[27] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. 2017. Progressive Neural Architecture Search. *CoRR* abs/1712.00559 (2017). arXiv:1712.00559 http://arxiv.org/abs/1712.00559

[28] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical Representations for Efficient Architecture Search. In *ICLR*. https://openreview.net/forum?id=BJQRKzbA-

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. *arXiv preprint arXiv:1806.09055* (2018).

[30] I. Loshchilov and F. Hutter. 2016. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv preprint arXiv:1608.03983* (August 2016). arXiv:1608.03983

[31] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology (HLT '94)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 114–119. https://doi.org/10.3115/1075812.1075835

[32] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. 2017. Evolving Deep Neural Networks. *arXiv preprint arXiv:1703.00548* (March 2017). arXiv:1703.00548

[33] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[34] Gerulf KM Pedersen and Zhenyu Yang. 2006. Multi-objective PID-controller tuning for a magnetic levitation system using NSGA-II. In *GECCO*. ACM, 1737–1744.

[35] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. 1999. BOA: The Bayesian optimization algorithm. In *GECCO*. Morgan Kaufmann Publishers Inc., 525–532.

[36] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, Stockholmsmässan, Stockholm Sweden, 4095–4104. http://proceedings.mlr.press/v80/pham18a.html

[37] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *CoRR* abs/1802.03268 (2018). arXiv:1802.03268 http://arxiv.org/abs/1802.03268

[38] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

[39] E. Real, A. Aggarwal, Y. Huang, and Q. V Le. 2018. Regularized Evolution for Image Classifier Architecture Search. *arXiv preprint arXiv:1802.01548* (Feb 2018). arXiv:1802.01548

[40] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. 2017. Large-Scale Evolution of Image Classifiers. *arXiv preprint arXiv:1703.01041* (March 2017). arXiv:1703.01041

[41] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-scale Image Recognition. In *ICLR*.

[42] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks Through Augmenting Topologies. *Evol. Comput.* 10, 2 (June 2002), 99–127. https://doi.org/10.1162/106365602320169811

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015.

Table 5: Summary of relevant related work along with datasets each method has been applied to, objectives optimized, and the computational power used (if reported). Methods not explicitly named are presented as the author names. PTB refers to the Penn Treebank [31] dataset. The Dataset(s) column describes what datasets the method performed a search with, meaning other datasets may have been presented in a study, but not used to perform architecture search. A dash represents some information not being provided. We attempt to limit the focus here to published methods, though some unpublished methods may be listed for historical contingency.

	Method Name	Dataset(s)	Objective(s)	Compute Used
RL	Zoph and Lee [50]	CIFAR-10, PTB	Accuracy	800 Nvidia K80 GPUs 22,400 GPU Hours
	NASNet [51]	CIFAR-10	Accuracy	500 Nvidia P100 GPUs 2,000 GPU Hours
	BlockQNN [49]	CIFAR-10	Accuracy	32 Nvidia 1080Ti GPUS 3 Days
	MetaQNN [1]	SVHN, MNIST CIFAR-10	Accuracy	10 Nvidia GPUs 8-10 Days
	MONAS [17]	CIFAR-10	Accuracy & Power	Nvidia 1080Ti GPUs
	EAS [2]	SVHN, CIFAR-10	Accuracy	5 Nvidia 1080Ti GPUs 2 Days
	ENAS [37]	CIFAR-10, PTB	Accuracy	1 Nvidia 1080Ti GPUs < 16 Hours
EA	CoDeepNEAT [32]	CIFAR-10, PTB	Accuracy	1 Nvidia 980 GPU
	Real *et al.* [40]	CIFAR-10, CIFAR-100	Accuracy	-
	AmoebaNet [39]	CIFAR-10	Accuracy	450 Nvidia K40 GPUs ~7 Days
	GeNet [47]	CIFAR-10	Accuracy	10 GPUs 17 GPU Days
	NEMO [23]	MNIST, CIFAR-10 Drowsiness Dataset	Accuracy & Latency	60 Nvidia Tesla M40 GPUs
	Liu *et al.* [28]	CIFAR-10	Accuracy	200 Nvidia P100 GPUs
	LEMONADE [10]	CIFAR-10	Accuracy	Titan X GPUs 56 GPU Days
	PNAS [27]	CIFAR-10	Accuracy	-
	PPP-Net [9]	CIFAR-10	Accuracy & Params/FLOPS/Time	Nvidia Titan X Pascal
Other	NASBOT [22]	CIFAR-10 Various	Accuracy	2-4 Nvidia 980 GPUs
	DPC [3]	Cityscapes [4]	Accuracy	370 GPUs 1 Week
	NAO [18]	CIFAR-10	Accuracy	200 Nvidia V100 GPUs 1 Day
	DARTS [29]	CIFAR-10	Accuracy	1 Nvidia 1080Ti GPUs 1.5 - 4 Day

12

Going Deeper with Convolutions. In *CVPR*.

[44] Ma Guadalupe Castillo Tapia and Carlos A Coello Coello. 2007. Applications of multi-objective evolutionary algorithms in economics and finance: A survey. In *CEC*. IEEE, 532–539.

[45] Christopher John Cornish Hellaby Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation. King's College, Cambridge, UK. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[46] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. 2016. Network Morphism. In *ICML*.

[47] L. Xie and A. Yuille. 2017. Genetic CNN. In *ICCV*.

[48] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. In *BMVC*.

[49] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. 2017. Practical Network Blocks Design with Q-Learning. *CoRR* abs/1708.05552 (2017). arXiv:1708.05552 http://arxiv.org/abs/1708.05552

[50] B. Zoph and Q. V. Le. 2016. Neural Architecture Search with Reinforcement Learning. *arXiv preprint arXiv:1611.01578* (Nov 2016). arXiv:cs.LG/1611.01578

[51] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.