# Neural Architecture Transfer

Zhichao Lu, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti

**Abstract**—Neural architecture search (NAS) has emerged as a promising avenue for automatically designing task-specific neural networks. Most existing NAS approaches require one complete search for each deployment specification of hardware or objective. This is a computationally impractical endeavor given the potentially large number of application scenarios. In this paper, we propose *Neural Architecture Transfer* (NAT) to overcome this limitation. NAT is designed to efficiently generate task-specific custom models that are competitive even under multiple conflicting objectives. To realize this goal we learn task-specific supernets from which specialized subnets can be sampled without any additional training. The key to our approach is an integrated online transfer learning and many-objective evolutionary search procedure. A pre-trained supernet is iteratively adapted while simultaneously searching for task-specific subnets. We demonstrate the efficacy of NAT on 11 benchmark image classification tasks ranging from large-scale multi-class to small-scale fine-grained datasets. In all cases, including ImageNet, NATNets improve upon the state-of-the-art under mobile settings ($\leq$ 600M Multiply-Adds). Surprisingly, small-scale fine-grained datasets benefit the most from NAT. At the same time, the architecture search and transfer is orders of magnitude more efficient than existing NAS methods. Overall, experimental evaluation indicates that, across diverse image classification tasks and computational objectives, NAT is an appreciably more effective alternative to fine-tuning based transfer learning. Code is available at https://github.com/human-analysis/neural-architecture-transfer.

**Index Terms**—Convolutional Neural Networks, Neural Architecture Search, AutoML, Transfer Learning, Evolutionary Algorithms.

## 1 Introduction

Image classification is a fundamental task in computer vision, where given a dataset and, possibly, multiple objectives to optimize, one seeks to learn a model to classify images. Solutions to address this problem fall into two categories: (a) Sufficient Data: A custom convolutional neural network architecture is designed and its parameters are trained from scratch using variants of stochastic gradient descent, and (b) Insufficient Data: An existing architecture designed on a large scale dataset, such as ImageNet [1], along with its pre-trained weights (e.g., VGG [2], ResNet [3]), is fine-tuned for the task at hand. These two approaches have emerged as the mainstays of present day computer vision.

Success of the aforementioned approaches is primarily attributed to architectural advances in convolutional neural networks. Initial efforts at designing neural architectures relied on human ingenuity. Steady advances by skilled practitioners has resulted in designs, such as AlexNet [4], VGG [2], GoogLeNet [5], ResNet [3], DenseNet [6] and many more, which have led to performance gains on the ImageNet Large Scale Visual Recognition Challenge [1]. In most other cases, a recent large scale study [7] has shown that, across many tasks, transfer learning by fine-tuning ImageNet pre-trained networks outperforms networks that are trained from scratch on the same data.

Moving beyond manually designed network architectures, Neural Architecture Search (NAS) [8] seeks to automate this process and find not only good architectures, but also their associated weights for a given image classification task. This goal has led to notable improvements in convolutional neural network architectures on standard image classification benchmarks, such as ImageNet, CIFAR-10 [9], CIFAR-100 [9] etc., in terms of predictive performance, computational complexity and model size. However, apart from transfer learning by fine-tuning the *weights*, current NAS approaches have failed to deliver new models for both *weights* and *topology* on custom non-standard datasets. The key barrier to realizing the full potential of NAS is the large data and computational requirements for employing existing NAS algorithms on new tasks.

In this paper, we introduce *Neural Architecture Transfer* (NAT) to breach this barrier. Given an image classification task, NAT obtains custom neural networks (both *topology* and *weights*), optimized for possibly many conflicting objectives, and does so without the steep computational burden of running NAS for each new task from scratch. A single run of NAT efficiently obtains multiple custom neural networks spanning the entire trade-off front of objectives.

Our solution builds upon the concept of a supernet [10] which comprises of many subnets. All subnets are trained simultaneously through weight sharing, and can be sampled very efficiently. This procedure decouples the network training and search phases of NAS. A many-objective[1] search can then be employed on top of the supernet to find all network architectures that provide the best trade-off among the objectives. However, training such supernets for each task from scratch is very computationally and data intensive. The key idea of NAT is to leverage an existing supernet and efficiently transfer it into a task-specific supernet, whilst simultaneously searching for architectures that offer the best trade-off between the objectives of interest. Therefore, unlike standard supernet based NAS, we combine supernet transfer learning with the search process. At the conclusion of this process, NAT returns (i) subnets that span the entire objective trade-off front, and (ii) a task-specific supernet. The latter can now be utilized for all future deployment-specific NAS, i.e., new and different hardware or objectives, without any additional training.

The core of NAT's efficiency lies in only adapting the subnets

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

[1] Problems having more than three objectives are called many-objective problems [11].

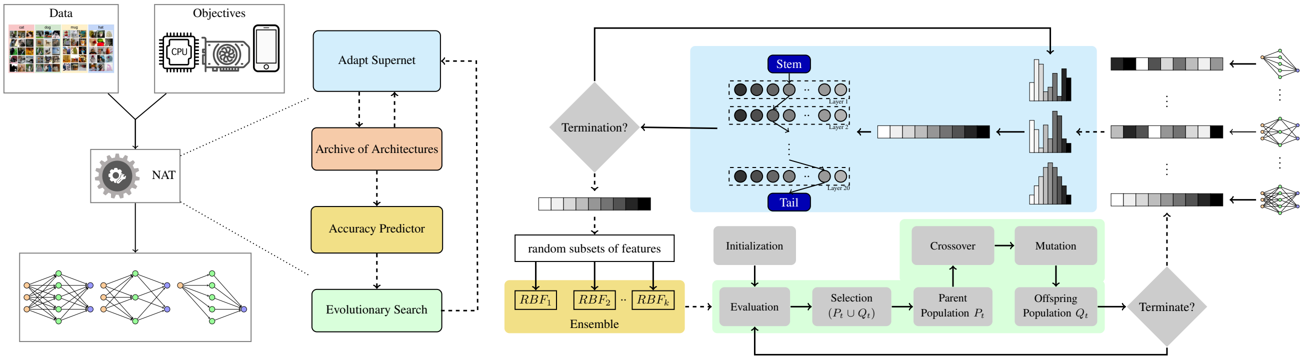


**Fig. 1: Overview:** Given a dataset and objectives to optimize, NAT designs custom architectures spanning the objective trade-off front. NAT comprises of two mains components, supernet adaptation and evolutionary search, that are iteratively executed. NAT also uses an online accuracy predictor model to improve its computational efficiency.

of the supernet that will lie on the efficient trade-off front of the new dataset, instead of all possible subnets. But, the structure of the corresponding subnets is unknown before adaptation. We resolve this "chicken-and-egg problem" by adopting an online procedure that alternates between the two primary stages of NAT: (a) *supernet adaptation* of subnets that are at the current trade-off front, and (b) *evolutionary search* for subnets that span the many-objective trade-off front. A pictorial overview of the entire NATNet method is shown in Fig.1.

In the *adaptation* stage, we first construct a layer-wise empirical distribution from the best subnets returned by evolutionary search. Then, subnets sampled from this distribution are fine-tuned. In the *search* stage, to improve the efficiency of the search, we adopt a surrogate model to quickly predict the objectives of any sampled subnet without a full-blown and costly evaluation. Furthermore, the predictor model itself is also learned online from previously evaluated subnets. We alternate between these two stages until our computational budget[2] is exhausted. The key contributions of this paper are:

**–** We introduce *Neural Architecture Transfer* as a NAS powered alternative to fine-tuning based transfer learning. NAT is powered by a simple, yet highly effective online supernet fine-tuning and online accuracy predicting surrogate model.

**–** We demonstrate the scalability and practicality of NAT on multiple datasets corresponding to different scenarios; large-scale multi-class (ImageNet, CINIC-10 [12]), medium-scale multi-class (CIFAR-10, CIFAR-100), small-scale multi-class (STL-10 [13]), large-scale fine-grained (Food-101 [14]), medium-scale fine-grained (Stanford Cars [15], FGVC Aircraft [16]) and small-scale fine-grained (DTD [17], Oxford-IIIT Pets [18], Oxford Flowers102 [19]) datasets.

**–** Under mobile settings ($\leq$ 600M MAdds), NATNets leads to state-of-the-art performance across all these tasks. For instance, on ImageNet, NATNet, achieves a Top-1 accuracy of 80.5% at 600M MAdds.

**–** Finally we demonstrate the scalability and utility of NAT across many objectives. Optimizing accuracy, model size and one of MAdds, CPU or GPU latency, NATNets dominates MobileNet-v3 [20] across all objectives. We also consider a 12 objective problem of finding a common architecture across eleven datasets while minimizing MAdds. The best trade-off NATNet dominates all models across these datasets under mobile settings.

## 2 Related Work

Recent years have witnessed growing interesting in neural architecture search. The promise of being able to automatically search for task-dependent network architectures is particularly appealing as deep neural networks are widely deployed in diverse applications and computational environments. Early methods [30], [31] made efforts to simultaneously evolve the topology of neural networks along with weights and hyperparameters. These methods perform competitively with hand-crafted networks on simple control tasks with shallow fully connected networks. Recent efforts [32] primarily focus on designing deep convolutional neural network architectures.

The development of NAS largely happened in two phases. Starting from NASNet [8], the focus of the first wave of methods was primarily on improving the predictive accuracy of CNNs including Block-QNN [33], Hierarchical NAS [34], and AmoebaNet [35], etc. These methods relied on Reinforcement Learning (RL) or Evolutionary Algorithm (EA) to search for an optimal modular structure that is repeatedly stacked together to form a network architecture. The search was typically carried out on relatively small-scale datasets (e.g. CIFAR-10/100 [9]), following which the best architectures were transferred to ImageNet for validation. A steady stream of improvements over state-of-the-art on numerous datasets was reported. The focus of the second wave of NAS methods was on improving the search efficiency.

A few methods have also been proposed to adapt NAS to other scenarios. These include meta-learning based approaches [36], [37] with application to few-shot learning tasks. XferNAS [38] and EAT-NAS [39] illustrate how architectures can be transferred between similar datasets or from smaller to larger datasets. Some approaches [40], [41] proposed RL-based NAS methods that search on multiple tasks during training and transfer the learned search strategy, as opposed to searched networks, to new tasks at inference. Next, we provide short overviews on methods that are closely related to the technical approach in this paper. Table 1 provides a comparative overview of NAT to existing NAS approaches.

**Performance Prediction:** Evaluating the performance of an architecture requires a computationally intensive process of iteratively optimizing model weights. To alleviate this computational burden regression models have been learned to predict an architecture's performance without actually training it. Baker *et al.* [42] use a

[2] We manually set the computational budget to a maximum of 1 day on a 8-GPU (NVIDIA 2080Ti) server. This is equivalent to the computational resources available to a small lab.

**TABLE 1:** Comparison of NAT and existing NAS methods.

| Methods | Search Method | Performance Prediction | Weight Sharing | Multiple Objective | Dataset Searched |
|---|---|---|---|---|---|
| NASNet [8] | RL | | | | C10 |
| PNAS [21] | SBMO | ✓ | | | C10 |
| DARTS [22] | gradient | | ✓ | | C10 |
| LEMONADE [23] | EA | | ✓ | ✓ | C10, C100, ImageNet64 |
| ProxylessNAS [24] | RL | | ✓ | | ImageNet |
| MnasNet [25] | RL | | | | ImageNet |
| EfficientNet [26] | RL+scaling | | | | ImageNet |
| ChamNet [27] | EA | ✓ | | | ImageNet |
| MobileNetV3 [20] | RL+expert | | | | ImageNet |
| OnceForAll [28] | EA | ✓ | ✓ | | ImageNet |
| FBNetV2 [29] | gradient | | | | ImageNet |
| **NAT (this paper)** | EA+transfer | ✓ | ✓ | ✓ | ImageNet, C10, C100, CINIC-10, STL-10, Flowers102, Pets, DTD, Cars, Aircraft, Food-101 |

radial basis function to estimate the final accuracy of architectures from its accuracy in the first 25% of training iterations. PNAS [21] uses a multilayer perceptron (MLP) and a recurrent neural network to estimate the expected improvement in accuracy if the current modular structure (which is later stacked together to form a network) is expanded with a new branch. Conceptually, both of these methods seek to learn a prediction model that extrapolate (rather than interpolate), resulting in poor correlation in prediction. OnceForAll [28] also uses a MLP to predict accuracy from architecture encoding. However, the model is trained offline for the entire search space, thereby requiring a large number of samples for learning (16K samples -> 2 GPU-days for just constructing the surrogate model). Instead of using uniformly sampled architectures to train the prediction model to approximate the entire landscape, ChamNet [27] trains many architectures through full SGD and selects only 300 samples of high accuracy with diverse efficiency (Multiply-adds, Latency, Energy) to train a prediction model offline. In contrast, NAT learns a prediction model in an online fashion only on the samples at the current trade-off front as we explore the search space. Such an approach only needs to interpolate over a much smaller space of architectures constituting the current trade-off front. Consequently, this procedure significantly improves both the accuracy and the sample complexity of constructing the prediction model.

**Weight Sharing:** Approaches in this category involve training a *supernet* that contains all searchable architectures as its subnets. They can be broadly classified into two categories depending on whether the supernet training is coupled with architecture search or decoupled into a two-stage process. Approaches of the former kind [22], [24], [43] are computationally efficient but return sub-optimal models. Numerous studies [44], [45], [46] allude to weak correlation between performance at the search and final evaluation stages. Methods of the latter kind [10], [28], [47] use performance of subnets (obtained by sampling the trained supernet) as a metric to select architectures during search. However, training a supernet beforehand for each new task is computationally prohibitive. In this work, we take an integrated approach where we train a supernet on large-scale datasets (e.g. ImageNet) once and couple it with our architecture search to quickly adapt it to a new task.

**Multi-Objective NAS:** Methods that consider multiple objectives for designing hardware specific models have also been developed. The objectives are optimized either through (i) scalarization, or (ii) Pareto-based solutions. The former include, ProxylessNAS [24], MnasNet [25], FBNet [48], and MobileNetV3 [20] which use a scalarized objective that encourages high accuracy and penalizes compute inefficiency at the same time, e.g., maximize $Acc * (Latency/Target)^{-0.07}$. These methods require a pre-defined preference weighting of the importance of different objectives before the search, which in itself requires a number of trials. Methods in the latter category include [23], [49], [50], [51], [52] and aim to approximate the entire Pareto-efficient frontier simultaneously. These approaches rely on heuristics (e.g., EA) to efficiently navigate the search space allowing practitioners to visualize the trade-off between the objectives and to choose a suitable network *a posteriori* to the search. NAT falls into the latter category and uses an accuracy prediction model and weight sharing for efficient architecture transfer to new tasks.

## 3 Proposed Approach

*Neural Architecture Transfer* consists of three main components; an accuracy predictor, an evolutionary search routine, and a supernet. NAT starts with an archive $\mathcal{A}$ of architectures (subnets) created by uniform sampling from our search space. We evaluate the performance $f_i$ of each subnet ($\boldsymbol{a}_i$) using weights inherited from the supernet. The accuracy predictor is then constructed from $(\boldsymbol{a}_i, f_i)$ pairs which (jointly with any additional objectives provided by the user) drives the subsequent many-objective evolutionary search towards optimal architectures. Promising architectures at the conclusion of the evolutionary process are added to the archive $\mathcal{A}$. The (partial) weights of the supernet corresponding to the top-ranked subnets in the archive are fine-tuned. NAT repeats this process for a pre-specified number of iterations. At the conclusion, we output both the archive and the task-specific supernet. Networks that offer the best trade-off between the objectives can be obtained from the archive. Detailed descriptions of each component of NAT are provided in the following subsections. Figure 1 and Algorithm 1 provide an overview of our entire approach.

### 3.1 Problem Formulation

The problem of neural architecture search for a target dataset $\mathcal{D} = \{\mathcal{D}_{trn}, \mathcal{D}_{vld}, \mathcal{D}_{tst}\}$ with many objectives can be formulated as the following bilevel optimization problem [53],

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{F}(\boldsymbol{a}) = \left(f_1(\boldsymbol{a}; \boldsymbol{w}^*(\boldsymbol{a})), \ldots, f_m(\boldsymbol{a}; \boldsymbol{w}^*(\boldsymbol{a}))\right)^T, \\ \text{subject to} \quad & \boldsymbol{w}^*(\boldsymbol{a}) \in \arg\min \ \mathcal{L}(\boldsymbol{w}; \boldsymbol{a}), \\ & \boldsymbol{a} \in \boldsymbol{\Omega}_a, \quad \boldsymbol{w} \in \boldsymbol{\Omega}_w, \end{aligned} \tag{1}$$

**Algorithm 1:** Neural Architecture Transfer

```
Input : training data $\mathcal{D}_{trn}$, validation data $\mathcal{D}_{vld}$, additional
        objectives $\tilde{f}$, supernet $\mathcal{S}_w$, archive size $N$, # of iterations $T$,
        # of epochs $E$, # of generations $G$.
$t \leftarrow 0$ // initialize an iteration counter
$\mathcal{A} \leftarrow$ randomly initialize an archive of archs with a size of $N$.
while $t < T$ do
    // compute accuracy by inheriting weights and inference.
    $f \leftarrow \mathcal{S}_w(\mathcal{A}, \mathcal{D}_{vld})$
    // construct the accuracy predictor by Algo 2.
    $\mathcal{S}_f \leftarrow$ Accuracy Predictor($\mathcal{A}, f$)
    // find promising archs by evolutionary search in Algo. 3.
    $P_t \leftarrow$ Evolutionary Search($\mathcal{S}_f, \tilde{f}, \mathcal{A}, G$)
    // keep the top-$N$ ranked archs in archive following Algo 4.
    $\mathcal{A} \leftarrow$ Selection($\mathcal{A} \cup P_t, N$)
    // fine tune supernet to promising archs by Algo. 5.
    $\mathcal{S}_w \leftarrow$ Adapt($\mathcal{S}_w, \mathcal{A}, \mathcal{D}_{trn}, E$)
    $t \leftarrow t + 1$
end
// optional in case of no preferences from users.
$\mathcal{A}^* \leftarrow$ choose a subset of archs from $\mathcal{A}$ based on trade-offs by method
    presented in Section A.1.
Return $\mathcal{S}_w, \mathcal{A}, \mathcal{A}^*$.
```

where the upper-level variable $\boldsymbol{a}$ defines a candidate architecture, and the lower-level variable $\boldsymbol{w}(\boldsymbol{a})$ denotes its associated weights. $\mathcal{L}(\boldsymbol{w};\boldsymbol{a})$ is the cross-entropy loss on the training data $\mathcal{D}_{trn}$ for an architecture $\boldsymbol{a}$. $\boldsymbol{F} : \boldsymbol{\Omega} \rightarrow \mathbb{R}^m$ constitutes $m$ (user-) desired, possibly competing, objectives—e.g., predictive performance on validation data $\mathcal{D}_{vld}$, number of parameters (#Params), multiply-adds (#MAdds), latency / power consumption / memory footprint on specific hardware etc.

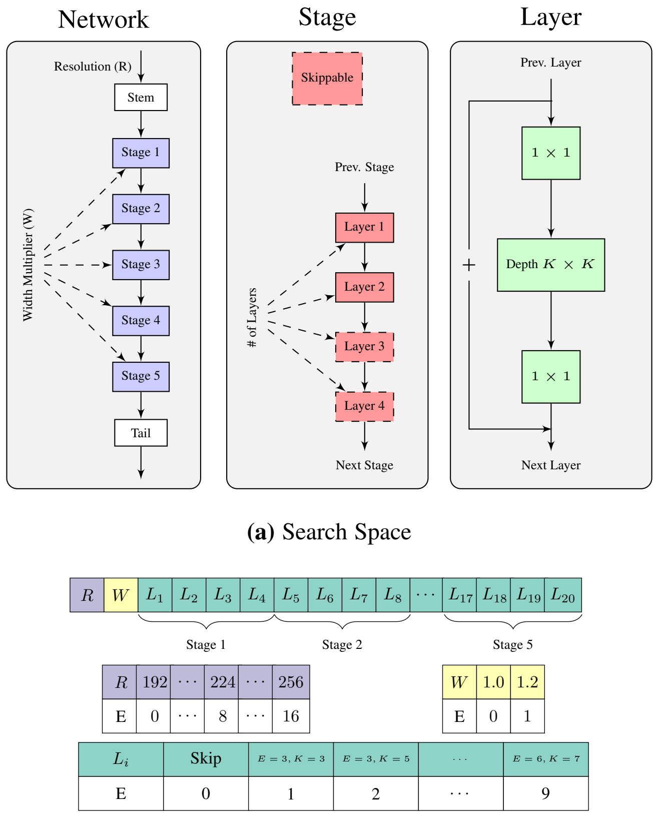
The bi-level optimization is typically solved in an iterative fashion, with an inner optimization loop over the weights of the network for a given architecture, and an outer optimization loop over the network architectures themselves. The computational challenge of solving this problem stems from both the upper and lower level optimization. Learning optimal weights of a network in the lower level necessitates costly iterations of stochastic gradient descent over multiple epochs. Similarly, searching the optimal architecture on the upper level is prohibitive due to the discrete nature of the architecture description, size of search space and our desire to optimize many, possibly conflicting, objectives.

## 3.2 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 feasible. We adopt a modular design for overall structure of the *network*, consisting of a stem, multiple stages and a tail (see Fig. 2a). The *stem* and *tail* are common to all networks and not searched. Each *stage* in turn comprises multiple layers, and each *layer* itself is an inverted residual bottleneck structure [54].

**-Network:** We search for the input image resolution and the width multiplier (a factor that scales the # of output channels of each layer uniformly [55]). Following previous work [25], [26], [28], we segment the CNN architecture into five sequentially connected stages. The stages gradually reduce the feature map size and increase the number of channels (Fig. 2a *Left*).

**-Stage:** We search over the number of layers, where only the first layer uses stride 2 if the feature map size decreases, and we allow each block to have minimum of two and maximum of four layers (Fig. 2a *Middle*).



**(a)** Search Space

**(b)** Encoding

**Fig. 2:** The architectures in our search space are variants of MobileNet-v2 family of models [20], [25], [26], [54]. (a) Each networks consists of five stages. Each stage has two to four layers. Each layer is an inverted residual bottleneck block. The search space includes, input image resolution (R), width multiplier (W), the number of layers in each stage, the # of output channels (expansion ratio E) of the first 1 × 1 convolution and the kernel size (K) of the depth-wise separable convolution in each layer. (b) Networks are represented as 22-integer strings, where the first two correspond to resolution and width multiplier, and the rest correspond to the layers. Each value indicates a choice, e.g. the third integer ($L_1$) takes a value of “1” corresponds to using expansion ratio of 3 and kernel size of 3 in layer 1 of stage 1.

**-Layer:** We search over the expansion ratio (between the # of output and input channels) of the first 1 × 1 convolution and the kernel size of the depth-wise separable convolution (Fig. 2a *Right*).

Overall, we search over four primary hyperparameters of CNNs i.e., the depth (# of layers), the width (# of channels), the kernel size, and the input resolution. The resulting volume of our search space is approximately $3.5 \times 10^{19}$ for each combination of image resolution and width multiplier.

To encode these architectural choices, we use an integer string of length 22, as shown in Fig. 2b. The first two values represent the input image resolution and width multiplier, respectively. The remaining 20 values denote the expansion ratio and kernel size settings for each of the 20 layers. The available options for expansion ratio and kernel size are [3, 4, 6] and [3, 5, 7], respectively. It is worth noting that we sort the layer settings in ascending #MAdds order, which is beneficial to the mutation operator used in our evolutionary search algorithm.

## 3.3 Accuracy Predictor

The main computational bottleneck of NAS arises from the nested nature of the bi-level optimization problem. The inner optimization requires the weights of the subnets to be thoroughly learned

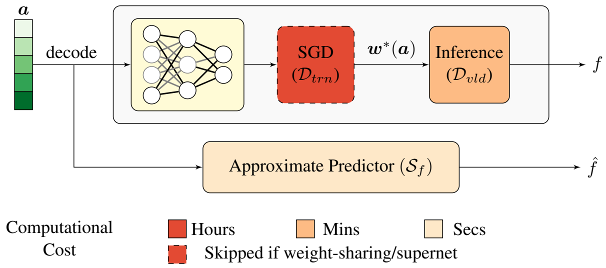

$a$
decode
SGD ($\mathcal{D}_{trn}$)
$w^*(a)$
Inference ($\mathcal{D}_{vld}$)
$f$
Approximate Predictor ($\mathcal{S}_f$)
$\hat{f}$
Computational Cost
Hours
Mins
Secs
Skipped if weight-sharing/supernet

**Fig. 3: Top Path:** A typical process of evaluating an architecture in NAS algorithms. **Bottom Path:** Accuracy predictor aims to bypass the time-consuming components for evaluating a network's performance by directly regressing its accuracy $f$ from $\boldsymbol{a}$ (architecture in the encoded space).

prior to evaluating its performance. Methods like weight-sharing [28], [43], [47] allow sampled subnets to inherit weights among themselves or from a supernet, avoiding the time-consuming process (typically requiring hours) of learning weights through SGD. However, standalone weight-sharing still requires inference on validation data (typically requiring minutes) to assess performance. Therefore, simply having to evaluate the subnets can still render the overall process computationally prohibitive for methods [8], [25], [35] that sample thousands of architectures during search.

To mitigate the computational burden of fully evaluating the subnets, we adopt a surrogate accuracy predictor that regresses the performance of a sampled subnet without performing training or inference. By learning a functional relation between the integer-strings (subnets in the encoded space) and the corresponding performance, this approach decouples the evaluation of an architecture from data-processing (including both SGD and inference). Consequently, the evaluation time reduces from hours/minutes to seconds. We illustrate this concept in Fig. 3. The effectiveness of this idea, however, is critically dependent on the quality of the surrogate model. Below we identify three desired properties of such a model:

1) Reliable prediction: high rank-order correlation[3] between predicted and true performance.
2) Consistent prediction: the quality of the prediction should be consistent across different datasets.
3) Sample efficiency: minimizing the number of training examples necessary to construct an accurate predictor model, since each training sample requires costly training and evaluation of a subnet.

Current approaches [21], [27], [28] that use surrogate based accuracy predictors, however, do not satisfy property (1) and (3) simultaneously. For instance, PNAS [21] uses 1,160 subnets to build the surrogate but only achieves a rank-order correlation of 0.476. Similarly, Once-For-All [28] uses 16,000 subnets to build the surrogate. The poor sample complexity and rank-order correlation of these approaches, is due to the offline learning of the surrogate model. Instead of focusing on models that are at the trade-off front of the objectives, these surrogate models are built for the entire search space. Consequently, these methods require a significantly larger and more complex surrogate model.

**Algorithm 2:** Accuracy Predictor (RBF Ensemble)

```
Input : training data $X$, training targets $Y$, ensemble size $K$
$k \leftarrow 0$ // initialize an counter
pool $\leftarrow \emptyset$ // initialize a pool to store all models.
while $k < K$ do
    $(\tilde{X}, \tilde{Y}) \leftarrow$ randomly create a subset of the training data.
    idx $\leftarrow$ randomly pick a subset of the features in training data.
    rbf $\leftarrow$ fit a RBF model from $\tilde{X}[:, idx]$ and $\tilde{Y}$.
    pool $\leftarrow$ pool $\cup$ (rbf, idx) // append the fitted model to the pool.
    $k \leftarrow k + 1$
end
Return a pool of $K$ RBF models.
```

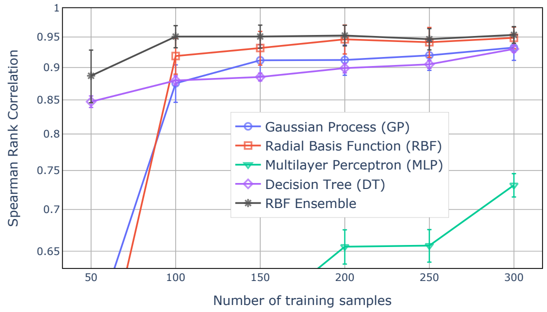


**Fig. 4:** Accuracy predictor performance as a function of training samples. For each model, we show the mean and standard deviation of the Spearman rank correlation on 11 datasets (Table 3). The size of RBF ensemble is 500.

We overcome the aforementioned limitation by restricting the surrogate model to the search space that constitutes the current objective trade-off. Such a solution significantly reduces the sample complexity of the surrogate and increases the reliability of its predictions. We adopt four low-complexity predictors, namely, Gaussian Process (GP) [27], Radial Basis Function (RBF) [42], Multilayer Perceptron (MLP) [21], and Decision Tree (DT) [56]. Empirically, we observe that RBFs are consistently better than the other three models if the # of training samples is more than 100. To further improve RBF's performance, especially under a high sample efficiency regime, we construct an ensemble of RBF models. As outlined in Algorithm 2, each RBF model is constructed with a subset of samples and features randomly selected from the training instances. The correlation between predicted accuracy and true accuracy from an ensemble of 500 RBF models outperforms all other models across all regimes. Figure 4 compares the performance of the different surrogate models we considered. Practically, we observed that the RBF ensemble can be learned under a minute.

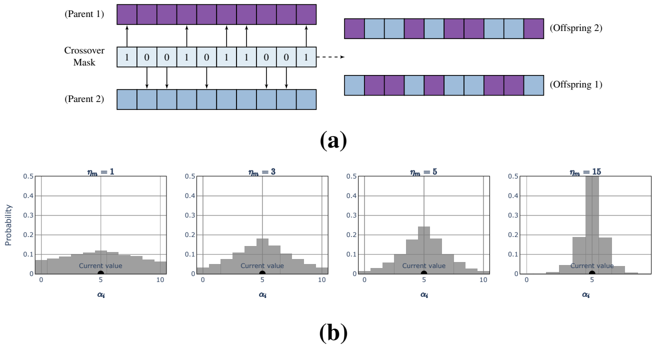
### 3.4 Many-Objective Evolutionary Search

Given the accuracy predictor we employ a customized evolutionary algorithm (EA) to search for optimal architectures that offer the best trade-off between many objectives. The EA is an iterative process in which initial architectures, selected from the archive of previously explored architectures, are gradually improved as a group, referred to as a *population*. In every generation (iteration), a group of *offsprings* (i.e., new architectures) are created by applying variations through crossover and mutation (described below) operations on the most promising architectures, also known as *parents*, found so far in the population. Every member of the population, i.e., both parents and offspring, competes for survival and reproduction (becoming a parent) in each generation. See Fig. 1 (bottom right shaded in green) for a pictorial overview, and Algorithm 3 for the pseudocode.

[3] Low mean square error is also desirable, but not necessary since the selection of architectures in the subsequent evolutionary search compares relative performance between architectures.

```
Algorithm 3: Evolutionary Search
Input : accuracy predictor $\mathcal{S}_f$, additional objectives $\tilde{f}$, archive of
        archs $\mathcal{A}$, max. number of generations $G$, population size $K$,
        crossover probability $p_c$, mutation probability $p_m$.
$g \leftarrow 0$ // initialize an generation counter
$f \leftarrow \mathcal{S}_f(\mathcal{A})$ // compute accuracy of all archs in archive.
$P \leftarrow$ Selection($\mathcal{A}, f, \tilde{f}(\mathcal{A}), K$) // initialize the parent population with
    top-$K$ ranked archs from $\mathcal{A}$.
while $g < G$ do
    // choose parents through tournament selection for mating.
    $P \leftarrow$ Binary Tournament Selection($P$)
    // create offspring population by crossover between parents.
    $Q \leftarrow$ Crossover($P, p_c$)
    // induce randomness to offspring population through mutation.
    $Q \leftarrow$ Mutation($Q, p_m$)
    $R \leftarrow P \cup Q$ // merge parent and offspring population.
    // survive the top-$K$ archs to next generation.
    $P \leftarrow$ Selection($R, \mathcal{S}_f(R), \tilde{f}(R), K$)
    $g \leftarrow g + 1$
end
Return parent population $P$.
```

```
Algorithm 4: Reference Point Based Selection
Input : A set of archs $R$, their objectives $F$, number of archs to
        select $N$, reference directions $Z$.
// put archs into different fronts (rank levels) based on domination.
$(F_1, F_2, \ldots) \leftarrow$ non_dominated_sort($F$)
$S \leftarrow \emptyset$, $i \leftarrow 1$
while $|S|+|F_i| < N$ do $S \leftarrow S \cup F_i$; $i \leftarrow i + 1$;
$F_L \leftarrow F_i$ // next front is the split front where we cannot accommodate
    all archs associated with it.
if $|S|+|F_L| = N$ then $S \leftarrow S \cup F_L$;
else
    $(\tilde{S}, \tilde{F_L}) \leftarrow$ Normalize($S, F_L$) // normalize the objectives based
        the ideal and nadir points derived from $R$.
    $d \leftarrow$ compute orthogonal dist to $Z_i$ for each $i$
    $\rho \leftarrow$ count #associated solutions for $Z_i$ based on $d$ for each $i$.
        // remaining archs from $F_L$ to fill up $S$
    $S \leftarrow S \cup$ Niching($\tilde{F_L}$, $N - |S|$, $\rho$, $d$)
end
Return $S$.
```



**(a)**

**(b)**

**Fig. 5:** (a) **Crossover Operator**: new offspring architectures are created by recombining integers from two parent architectures. The probability of choosing from either one of the parents is equal. (b) **Mutation Operator**: histograms showing the probabilities of mutated values with current value at 5 under different hyperparameter $\eta_m$ settings.

**Crossover** exchanges information between two or more population members to create two or more new members. Designing an effective crossover between non-standard solution representations can be difficult and has been largely ignored by existing EA-based NAS algorithms [34], [35], [57]. Here we adopt a customized, homogeneous crossover that uniformly picks integers from parent architectures to create offspring architectures. This crossover operator offers two properties: (1) it preserves common integers shared between parents; and (2) it is free of additional hyperparameters. Fig. 5a visualizes our implementation of the crossover operation. We generate two offspring architectures with each crossover, and an offspring population of the same size as the parent population is generated in each generation.

**Mutation** is a *local* operator that perturbs a solution to produce a new solution in its vicinity. In this work, we use a discretized version of the polynomial mutation (PM) operator [58] and apply it to every solution created by the crossover operator. For a given architecture $\boldsymbol{a}$, PM is carried out integer-wise with probability $p_m$, and the mutated $i^{th}$ integer, $a_i$, of the mutated offspring is:

$$a_i' = \begin{cases} a_i + ((2u)^{1/(1+\eta_m)} - 1)(a_i - a_i^{(L)}), & \text{for } u \leq 0.5, \\ a_i + (1 - \big(2(1-u)\big)^{1/(1+\eta_m)})(a_i^{(U)} - a_i), & \text{for } u > 0.5 \end{cases} \tag{2}$$

where $u$ is a uniform random number in the interval $[0, 1]$. $a_i^{(L)}$ and $a_i^{(U)}$ are the lower and upper bounds of $a_i$, respectively. Each mutated value in an offspring is rounded to the nearest integer. The PM operator inherits the *parent-centric* convention, in which the offspring are intentionally created around the parents. The centricity is controlled via an index hyperparameter $\eta_m$. In particular, high-values of $\eta_m$ tend to create mutated offspring around the parent, and low-values encourage mutated offspring to be further away from the parent architecture. See Fig. 5b for a visualization of the effect of $\eta_m$. It is the worth noting that the PM operator was originally proposed for continuous optimization where distances between variable values are naturally defined. In contrast, in context of our encoding, our variables are categorical in nature, indicating a particular layer hyperparameter. So we sort the searched subnets in ascending order of #MAdds, such that $\eta_m$ now controls the difference in #MAdds between the parent and the mutated offspring.

We apply PM to every member in the offspring population (created from crossover). We then merge the mutated offspring population with the parent population and select the top half using many-objective selection operator described in Algorithm 4. This procedure creates the parent population for the next generation. We repeat this overall process for a pre-specified number of generations and output the parent population at the conclusion of the evolution.

### 3.5 Many-Objective Selection

In addition to high predictive accuracy, real-world applications demand NAS algorithms to simultaneously balance a few other conflicting objectives that are specific to the deployment scenarios. For instance, mobile or embedded devices often have restrictions in terms of model size, multiply-adds, latency, power consumption, and memory footprint. With no prior assumption on the correlation among these objectives, a scalable (to the number of objectives) selection is required to drive the search towards the high dimensional Pareto front. In this work, we adopt the reference point guided selection originally proposed in NSGA-III [11], which has been shown to be effective in handling problems with as many as 15 objectives. In the remainder of this section, we provide an overview of NSGA-III procedure and refer readers to the original publication for more details.

*Domination* is a widely-used partial ordering concept for comparing two objective vectors. For a generic many-objective optimization problem: $\min_{\boldsymbol{a}} \{f_1(\boldsymbol{a}), \ldots, f_m(\boldsymbol{a})\}$, where $f_i(\cdot)$

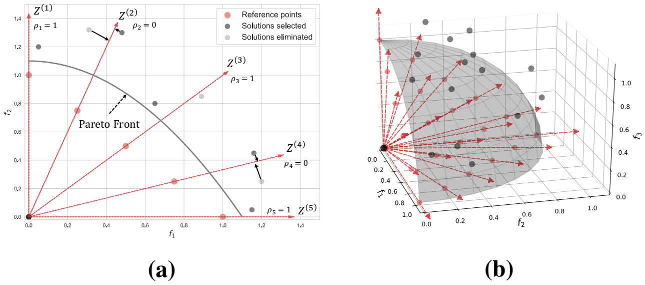


**Fig. 6:** (a) An example (assuming minimization of all objectives) of the selection process in Algo 4: We first create reference directions $Z$ by joining reference points with the ideal solution (origin). Then through *non_dominated_sort*, three non-dominated solutions are identified, associated with reference directions $Z^{(1)}$, $Z^{(3)}$ and $Z^{(5)}$. We then select the remaining solutions by the orthogonal distances to the reference directions with no associated solutions—i.e. $Z^{(2)}$ and $Z^{(4)}$. This selection is scalable to larger # of objectives. Tri-objective example is shown in (b).

are the objectives (say, loss functions) to be optimized and $\boldsymbol{a}$ is the representation of a neural network architecture. For two given solutions $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$, solution $\boldsymbol{a}_1$ is said to dominate $\boldsymbol{a}_2$ (i.e., $\boldsymbol{a}_1 \preceq \boldsymbol{a}_2$) if following conditions are satisfied:

1) $\boldsymbol{a}_1$ is no worse than $\boldsymbol{a}_2$ for all objectives $(f_i(\boldsymbol{a}_1) \leq f_i(\boldsymbol{a}_2), \forall i \in \{0, \dots, m\})$, and
2) $\boldsymbol{a}_1$ is strictly better than $\boldsymbol{a}_2$ in at least one objective $\exists i \in \{0, \dots, m\} \mid f_i(\boldsymbol{a}_1) < f_i(\boldsymbol{a}_2))$.

A solution $\boldsymbol{a}_i$ is said to be non-dominated if these conditions hold against all the other solutions $\boldsymbol{a}_j$ (with $j \neq i$) in the entire search space of $\boldsymbol{a}$.

With the above definition, we can sort solutions to different ranks of domination, where solutions in the same rank are non-dominated to each other, and there exists at least one solution in lower rank that dominates any solutions in the higher rank. Thus, a lower non-dominated ranked set is lexicographically better than a higher ranked set. This process is referred as *non_dominated_sort*, and it is the first step in the selection process. During the many-objective selection process, the lower ranked sets are chosen one at a time until no more sets can be included to maintain the population size. The final accepted set may have to be *split* to choose only a part. For this purpose, we choose the most diverse subset based on a diversity-maintaining mechanism. We first create a set of reference directions from a set of uniformly distributed (in $(m-1)$-dimensional space) reference points in the unit simplex by using Das-and-Dennis method [59]. Then we associate each solution to a reference direction based on orthogonal distance of the solution from the direction. Then, for every reference direction, we choose the closest associated solution in a systematic manner by adaptively computing a niche count $\rho$ so that every reference direction gets an equal opportunity to choose a representative closest solution in the selected population. The domination and diversity-preserving procedures are easily scalable to any number of objectives and importantly are free from any user-defined hyperparameter. See Algorithm 4 for the pseudocode and Fig. 6 for a graphical illustration.

### 3.6 Supnernet Adaptation

Instead of training every architectures sampled during search from scratch, NAS with weight sharing [22], [43] inherits weights from previously-trained networks or from a supernet. Directly inheriting the weights obviates the need to optimize the weights from scratch and speeds up the search from thousands of GPU days to only a few. In this work, we focus on the supernet approach [10], [28]. It involves first training a large network model (in which searchable architectures become subnets) prior to the search. Then the performance of the subnets, evaluated with the inherited weights, is used to guide the selection of architectures during search. The key to the success of this approach is that the performance of the subnets with the inherited weights be highly correlated with the performance of the same subnet when thoroughly trained from scratch. Satisfying this desideratum necessitates that the supernet weights be learned in such a way that *all* subnets are optimized *simultaneously*.

**Algorithm 5:** Adapt Supernet

```
Input : supernet $\mathcal{S}_w$, archive of archs $\mathcal{A}$, training data $\mathcal{D}_{trn}$,
        number of epochs $E$.
$e \leftarrow 0$ // initialize an epoch counter
$Dist \leftarrow$ construct the distribution from $\mathcal{A}$ following Eq. (3).
while $e < E$ do
    for each batch in $\mathcal{D}_{trn}$ do
        subnet $\leftarrow$ sample from Dist.
        $w \leftarrow$ set forward path of $\mathcal{S}_w$ according to subnet.
        $\mathcal{L} \leftarrow$ compute cross-entropy loss on data batch.
        $\nabla w \leftarrow$ compute the gradient by $\partial \mathcal{L} / \partial w$
        $\mathcal{S}_w \leftarrow$ one step of SGD.
    end
    $e \leftarrow e + 1$
end
Return supernet $\mathcal{S}_w$.
```

Existing methods [51], [60] attempt to achieve the above goal by imposing *fairness* in training the supernet, where the probabilities of training any particular subnet for each batch of data is uniform in expectation. However, we argue that simultaneously training all the subnets in the search space is practically not feasible and more importantly not necessary. Firstly, it is evident from existing NAS approaches [24], [48] that different objectives (#Params, #MAdds, latency on different hardware, etc.) require different architectures in order to be efficient. In other words, not all subnets are equally important for the task at hand. Secondly, only a tiny fraction[4] of the search space can practically be explored by a NAS algorithms.

Based on the aforementioned observations, we propose a simple yet effective supernet training routine that only focuses on training the subnets recommended by the evolutionary search algorithm in Section 3.5. Specifically, we seek to exploit the knowledge gained from the search process so far. Recall that our algorithm uses an archive to keep track of the promising architectures explored so far. For each value in our architecture encoding, we construct a categorical distribution from architectures in the archive, where the probability for $i^{th}$ integer taking on the $j$ value is computed as:

$$p(X_i = j) = \frac{\text{\# of architectures with option } j \text{ at } i^{th} \text{ integer}}{\text{total \# of architectures in the archive}} \tag{3}$$

In each training step (batch of data) we sample an integer-string from the above distribution[5]. We then activate the sub parts of the supernet corresponding to the architecture decoded from the integer-string. Only weights corresponding to the activated sub

[4] For example, AmoebaNet [35] samples a large number of 27K architectures which is still only about $10^{-13}\%$ of its search space.

[5] A visualization of such distributions is shown in Fig. 19.

**TABLE 2:** Hyperparameter Settings

| Category | Parameter | Setting |
|---|---|---|
| global | archive size | 300 |
| | number of iterations | 30 |
| accuracy predictor | Train size | 100 |
| | Ensemble size | 500 |
| evolutionary search | population size | 100 |
| | number of generations per iteration | 100 |
| | crossover probability | 0.9 |
| | mutation probability | 0.1 |
| | mutation index $\eta_m$ | 1.0 |
| supernet | number of epochs per iteration | 5 |

**TABLE 3:** Benchmark Datasets for Evaluation

| Dataset | Type | Train Size | Test Size | #Classes |
|---|---|---|---|---|
| ImageNet [1] | multi-class | 1,281,167 | 50,000 | 1,000 |
| CINIC-10 [12] | | 180,000 | 9,000 | 10 |
| CIFAR-10 [9] | | 50,000 | 10,000 | 10 |
| CIFAR-100 [9] | | 50,000 | 10,000 | 10 |
| STL-10 [13] | | 5,000 | 8,000 | 10 |
| Food-101 [14] | fine-grained | 75,750 | 25,250 | 101 |
| Stanford Cars [15] | | 8,144 | 8,041 | 196 |
| FGVC Aircraft [16] | | 6,667 | 3,333 | 100 |
| DTD [17] | | 3,760 | 1,880 | 47 |
| Oxford-IIIT Pets [18] | | 3,680 | 3,369 | 37 |
| Oxford Flowers102 [19] | | 2,040 | 6,149 | 102 |

parts in the supernet will be updated in each step. See Algorithm 5 for pseudocode.

## 4 Experimental Evaluation

In this section we present experimental results to evaluate the efficacy of *Neural Architecture Transfer* on multiple image classification tasks. In addition we also investigate the scalability of our approach to more than two objectives. For all the experiments in this section, we use the same set of hyperparameters (see Table 2) for the different components of NAT. These choices were guided by the ablation studies described in Section 5.

### 4.1 Datasets

We consider eleven image classification datasets for evaluation with sample size varying from 2,040 to 180,000 images (20 to 18,000 images per class; Table 3). These datasets span a wide variety of image classification tasks, including superordinate-level recognition (ImageNet [1], CIFAR-10 [9], CIFAR-100 [9], CINIC-10 [12], STL-10 [13]); fine-grained recognition (Food-101 [14], Stanford Cars [15], FGVC Aircraft [16], Oxford-IIIT Pets [18], Oxford Flowers102 [19]); and texture classification (DTD [17]). We use the ImageNet dataset for training the supernet, and use the other ten datasets for architecture transfer.

### 4.2 Supernet Preparation

Our supernet is constructed by setting the architecture encoding at the maximum value, i.e. four layers in each stage and every layer uses expand ratio of six and kernel size of seven. Adapting subnets of a supernet with randomly initialized weights leads to training instability and large variance in its performance. Therefore, we warm-up the supernet weights on ImageNet following the *progressive shrinking* algorithm [28], where the supernet is first trained at full-scale, with subnets corresponding to different options (expand ratio, kernel size, # of layers) being gradually activated during the training process. This procedure, which takes about 6 days on a sever with eight V100 GPUs, is optimized with only the cross-entropy loss i.e., a single objective. We note that supernet preparation[6] expense is a one-time cost that amortizes over any subsequent transfer to different datasets and objective combinations we show in the following subsections.

### 4.3 ImageNet Classification

Before we evaluate our approach for architecture transfer to other datasets, we first validate its effectiveness on the ImageNet-1K dataset. This experiment evaluates the effectiveness of NAT in adapting and searching for architectures that span trade-off between two objectives. For this experiment, we consider accuracy and #MAdds as the two objective of interest. We run NATNet for 30 iterations, and from the final archive of architectures we select four models ranging from 200M MAdds to 600M MAdds (for high-end mobile devices). Following [28] we further fine-tune[7] each model for 75 epochs. Our fine-tune training largely follows [25]: RMSProp optimizer with decay 0.9 and momentum 0.9; batch normalization momentum 0.99; weight decay 1e-5. We use a batch size of 512 and an initial learning rate of 0.012 that gradually reduces to zero following the cosine annealing schedule. Our regularization settings are similar as in [26]: we use augmentation policy [61], drop connect ratio 0.2, and dropout ratio 0.2.

Table 4 shows the performance of NAT models obtained through bi-objective optimization of maximizing accuracy and minimizing #MAdds. Our models, referred to as NAT-M$\{1,2,3,4\}$, are in ascending order of #MAdds (Fig. 7). Figure 8 shows the full #MAdds-accuracy trade-off curve comparison between NAT and existing NAS methods.

Results indicate that NATNets completely dominate (i.e. better in both #MAdds and accuracy) all existing designs, both manual and from other NAS algorithms, under mobile settings ($\leq$ 600M MAdds). Compared to manually. designed networks, NAT is noticeably more efficient. NAT-M1 is **2.3%** and **1.5% more accurate** than MobileNet-v3 [20] and FBNetV2-F4 [29] respectively, while being equivalent in efficiency (i.e. #MAdds, CPU and GPU latency). Furthermore, NATNets are consistently **6% more accurate** than MobileNet-v2 [54] scaled by width multiplier from 200M to 600M #MAdds. Our largest model, NAT-M4, achieves a new state-of-the-art ImageNet top-1 accuracy of 80.5% under mobile settings ($\leq$ 600M #MAdds). Interestingly, even though this experiment did not explicitly optimize for CPU or GPU latency, NATNets are faster than those ( MobileNet-V3 [20], MNasNet [25]) that explicitly do optimize for latency.

### 4.4 Architecture Transfer

Existing NAS approaches are rarely applied to tasks beyond standard datasets (i.e. CIFAR-10 [9] and ImageNet [1]), where the classification task is at superordinate-level and the # of training images are sufficiently large. Instead, they adopt a conventional transfer learning setup [7], in which the architectures found by searching on standard benchmark datasets are transferred as is, with weights fine-tuned to new datasets. We argue that such a process is conceptually contradictory to the goal of NAS. The architectures transferred from standard datasets are sub-optimal either with respect to accuracy, efficiency or both. On the other

[6] Many supernets, pre-trained on ImageNet, are already available if one does not want to train them from scratch. See https://github.com/mit-han-lab/once-for-all, https://github.com/meijieru/AtomNAS and https://github.com/JiahuiYu/slimmable_networks.

[7] Section 5.5 studies the impact of ablating the fine-tuning step.

**TABLE 4: ImageNet-1K Classification [1]:** NATNets comparison with manual and automated design of efficient convolutional neural networks. Models are grouped into sections for better visualization. Our results are underlined and the best result in each section is in bold. CPU latency (batchsize=1) is measured on Intel i7-8700K and GPU latency (batchsize=64) is measured on 1080Ti. "WS" stands for weight sharing. All methods are under single crop and single model condition, without any additional data.

| Model | Method | #Multi-Adds | CPU Lat (ms) | GPU Lat (ms) | Top-1 Acc (%) | Top-5 Acc (%) |
|---|---|---|---|---|---|---|
| **NAT-M1** | WS+EA | 225M | 9.1 | 30 | **77.5** | **93.5** |
| MobileNet-v2 [54] | manual | 300M | **8.3** | **23** | 72.0 | 91.0 |
| ProxylessNAS [24] | RL/gradient | 465M | 8.5 | 27 | 75.1 | 92.5 |
| MnasNet-A1 [25] | RL | 312M | 9.3 | 31 | 75.2 | 92.5 |
| MobileNet-v3 [20] | RL/NetAdapt | **219M** | 10.6 | 33 | 75.2 | - |
| MUXNet-m [52] | EA | **218M** | 14.7 | 42 | 75.3 | 92.5 |
| FBNetV2-F4 [29] | gradient | 238M | 15.6 | 44 | 76.0 | - |
| EfficientNet-B0 [26] | RL/scaling | 390M | 14.4 | 46 | 76.3 | 93.2 |
| **NAT-M2** | WS+EA | **312M** | **11.4** | **37** | **78.6** | **94.3** |
| MUXNet-l [52] | EA | 318M | 19.2 | 74 | 76.6 | 93.2 |
| FBNetV2-L1 [29] | gradient | 325M | - | - | 77.2 | - |
| AtomNAS-C+ [62] | WS+shrinkage | 363M | - | - | 77.6 | 93.5 |
| AutoNL-L [63] | gradient | 353M | - | - | 77.7 | 93.7 |
| DNA-c [64] | gradient | 466M | 14.5 | 67 | 77.8 | 93.7 |
| ResNet-152 [3] | mannual | 11.3B | 66.7 | 176 | 77.8 | 93.8 |
| FBNetV2-L2 [29] | gradient | 422M | - | - | 78.1 | - |
| **NAT-M3** | WS+EA | **490M** | **16.1** | **62** | **79.9** | **94.9** |
| EfficientNet-B1 [26] | RL/scaling | 700M | 19.5 | 67 | 78.8 | 94.4 |
| MixNet-L [65] | RL | 565M | 29.4 | 105 | 78.9 | 94.2 |
| BigNASModel-L [66] | WS | 586M | - | - | 79.5 | - |
| OnceForAll [28] | WS+EA | 595M | 16.5 | 72 | **80.0** | **94.9** |
| **NAT-M4** | WS+EA | **0.6B** | **17.3** | **78** | **80.5** | **95.2** |
| Inception-v4 [5] | manual | 13B | 84.6 | 206 | 80.0 | 95.0 |
| Inception-ResNet-v2 [5] | manual | 13B | 99.1 | 289 | 80.1 | 95.1 |

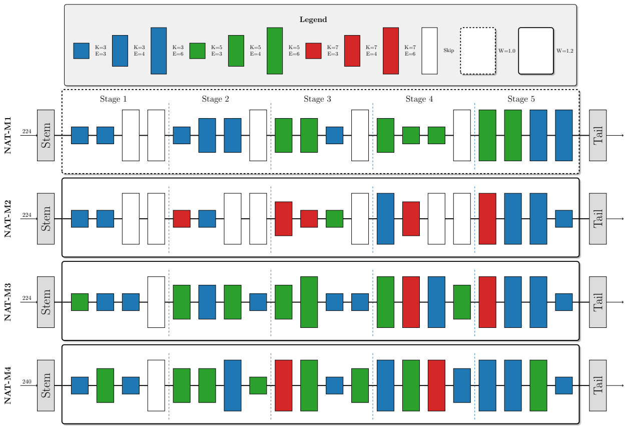


**Fig. 7:** ImageNet Architectures from Trade-Off Front.

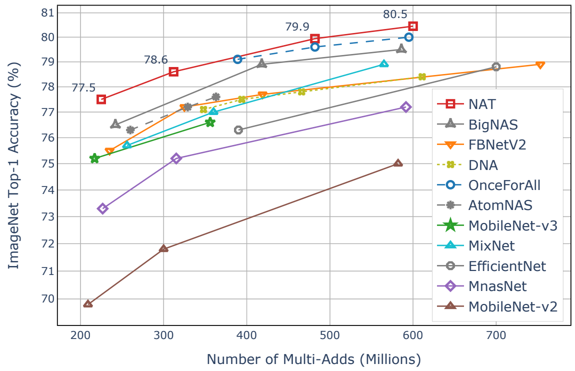


**Fig. 8: MAdds vs. ImageNet Accuracy**. NATNets outperform other models in both objectives. In particular, NAT-M4 achieves a new state-of-the-art top-1 accuracy of 80.5% under mobile setting (≤ 600M MAdds). NAT-M1 improves MobileNet-v3 top-1 accuracy by 2.3% with similar #MAdds.

hand, by transferring both architecture and weights NAT can indeed design bespoke models for each task.

We evaluated NAT on ten image classification datasets (see Table 3) that present different challenges in terms of diversity in classes (superordinate vs. fine-grained) and size of training set (large vs small). For each dataset, we run NAT with two objectives: maximize top-1 accuracy on validation data (20% randomly separated from the training set) and minimize #MAdds. We start from the supernet trained on ImageNet (which is created once before all experiments; see Section 4.2) and transfer it to the new task. During this procedure the last linear layer is reset depending on the number of categories in the new task. NAT is now applied for a total of 30 iterations. In each iteration the supernet is adapted for 5 epochs using SGD with a momentum of 0.9. The learning rate is initialized at 0.01 and annealed to zero in 150 epochs (30 iterations with five epochs in each). All hyperparameters are set at default values from Table 2. For each dataset, the overall NAT process takes slightly under a day on a server with eight 2080Ti GPUs.

Figure 9 shows the accuracy and #MAdds trade-off for each dataset across a wide range of models, including NATNets and conventional transfer learning (fine-tuning weights) of existing NAS and hand designed models. Across all datasets NATNets consistently achieve better accuracy while being an order of magnitude more efficient (#MAdds) than existing models. Under mobile settings (≤ 600M), NATNets achieve the state-of-the-art on these datasets, and a new state-of-the-art accuracy on both STL-

8. Actual values of each NATNet model are provided in Table 7.

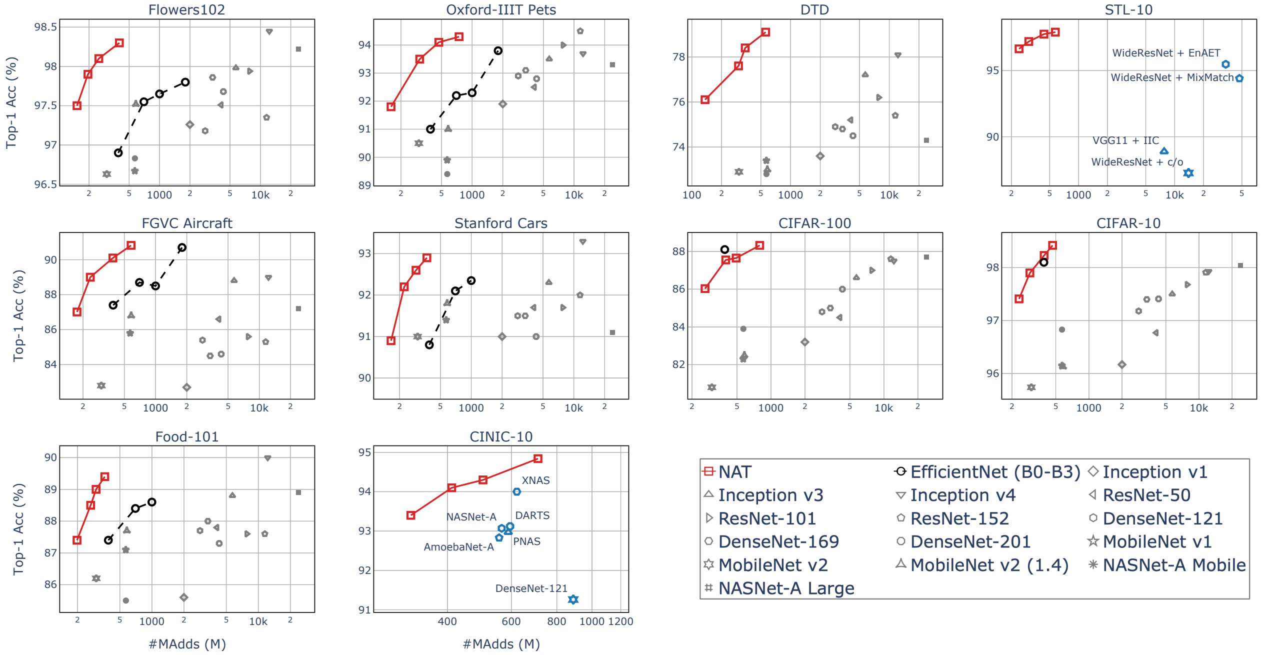


**Fig. 9: MAdds vs. Accuracy** trade-off curves[8]comparing NAT existing architectures on a diverse set of datasets. The datasets are arranged in ascending order of training set size. Methods shown in the legend pre-train on ImageNet and fine-tune the weights on the target dataset. Methods in blue train from scratch or use external training data.

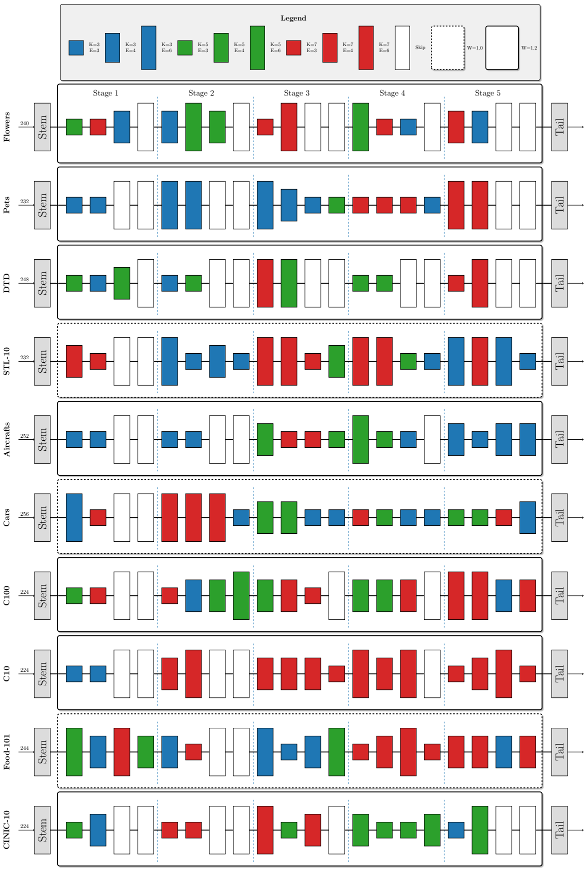


**Fig. 10:** Transfer architectures (350M MAdds) on ten datasets.

10 [13] and CINIC-10[9] [12] datasets. Surprisingly, on small scale datasets e.g. Oxford Flowers102 [19], Oxford-IIIT Pets [18], DTD [17] and STL-10 [13], we observe that NATNets are significantly more effective than conventional fine-tuning. Even on fine-grained datasets such as Stanford Cars and FGVC, where conventional fine-tuning did not improve upon training from scratch, NATNets improve accuracy while also being significantly more efficient.

Figure 10 shows a visualization of architectures with 350M MAdds for each dataset. The lack of similarity in the networks suggest that NAT was able to generate networks customized for each task. Additional visualization of architectures searched on all datasets is provided in Fig. 19.

## 4.5 Scalability to Objectives

Practical applications of NAS can rarely be considered from the point of view of a single objective, and most often, they must be evaluated from many different, possibly competing, objectives. We demonstrate the scalability of NAT to more than two objectives and evaluate its effectiveness.

### *4.5.1 Three Objectives*

We use NAT to simultaneously optimize for three objectives—namely, model accuracy on ImageNet, model size (#params), and model computational efficiency. We consider three different metrics to quantify computational efficiency—#MAdds, CPU latency, and GPU latency. In total, we run three instances of three-objective search—i.e. maximize accuracy, minimize #params, and minimize one of #MAdds, CPU latency or GPU latency. We follow the settings from the ImageNet experiment in Section 4.3, except the fine-tuning step.

After obtaining the non-dominated (trade-off) solutions, we first visualize the objectives in Fig. 11. We observe that Pareto surfaces emerge at higher model complexity regime (i.e. high #params, #MAdds, etc.), shown in the 3D scatter plot in the top row, suggesting that trade-offs exist between model size (#params) and model efficiency (#MAdds and latency). In other words,

[9] According to [67] for STL-10, and [68] for CINIC-10.

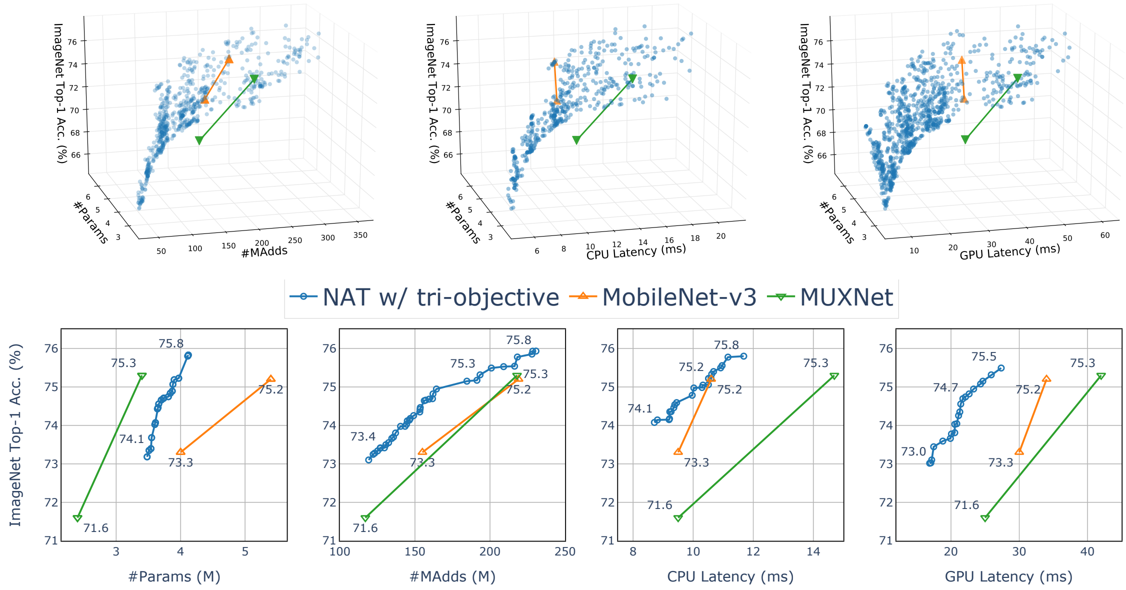


**Fig. 11: Top row:** NATNets obtained from tri-objective search to maximize ImageNet top-1 accuracy, minimize model size (#Params), and minimize {#MAdds, CPU latency, GPU latency} from left to right. **Bottom row:** 2D projections from above 3D scatter, showing top-1 accuracy vs. each of the four efficiency related measurements. To better visualize (the comparison with MobileNet-v3 [20] and MUXNet [52]), partial solutions from the non-dominated frontiers are shown. All top-1 accuracy shown are without fine-tuning.

#params and {#MAdds, CPU, GPU latency} are not completely correlated—e.g. a model with a fewer #params is not necessarily more efficient in #MAdds or latency than another model with more #params. This is one of the advantages of using a many-objective optimization algorithm compared to optimizing a single scalarized objective (such, as a weighted-sum of objectives).

Figure 11 visualizes, in 2D, the top-1 accuracy as a trade-off with each one of the four considered efficiency metrics in the bottom row. The 2D projection is obtained by ignoring the third objective. For better visualization we only show the architectures that are close to the performance trade-off of MobilNet-v3 [20]. NATNets obtained directly from the three-objective search i.e., before any fine-tuning of their weights, consistently outperform MobileNet-v3 on ImageNet along all the objectives (top-1 accuracy, #params, #MAdds, CPU and GPU latency). Additionally, we compare to MUXNets [52] which are also obtained from a three-objective NAS optimizing {top-1 accuracy, #params, and #MAdds}. However, MUXNets adopt a search space that is specifically tailored for reducing model size. Therefore, in comparison to MUXNets, we observe that NATNets perform favourably on all the remaining three efficiency metrics, except for #params.

### 4.5.2 *Twelve Objectives*

To further validate the scalability of NAT to a large number of objectives, we consider the top-1 accuracy on each of the 11 datasets shown in Table 3 along with #MAdds, as separate objectives, resulting in a 12-objective optimization problem. Not only is such a large-scale optimization plausible with NAT, it also reveals important information, which a low-dimensional optimization may not. During search, the accuracy on each dataset is computed by inheriting weights from the dataset-specific supernets generated from previous experiments (Section 4.4). Since the supernets are already adapted to each dataset, we exclude the supernet adaptation step in NAT for this experiment.

Figure 12 (Left) shows the 12 objective values for all 45 non-dominated architectures obtained by NAT in a parallel coordinate plot (PCP), where each vertical bar is an objective and each line connecting all 12 vertical bars is an architecture. We now apply the trade-off decision analysis presented in Section A.1 and observe that the highest trade-off solution is more than $(\mu + 3\sigma)$ trade-off away from the rest of 44 solutions. This solution is highlighted in dark blue in Fig. 12 (Left). Its intermediate performance in all objectives indicate that this best trade-off solution makes a good compromise on all 12 objectives among all 45 obtained solutions. In Fig. 12 (Right), we compare this solution with different baseline models that are fine-tuned to each dataset separately. Notably, our NATNet achieves better accuracy on all datasets with similar or less #MAdds than EfficientNet-B0 [26], Mobilenet-v2 [54], NASNet-A [8], and ResNet-50 [3], making our highest trade-off solution a preferred one.

The above analysis alludes to a computational mechanism for choosing a single preferred trade-off solution from the Pareto solutions obtained by a many-objective optimization algorithm. If such an overwhelmingly high trade-off solution exists in the Pareto front, it becomes one of the best choices and can outperform solutions found by a single-objective optimization algorithm. Without resorting to a many-objective optimization to find multiple trade-off solutions, identification of such a high trade-off solution is very challenging.

# 5 Ablation Study

## 5.1 Accuracy Predictor Performance

In this subsection we assess the effectiveness of different accuracy predictor models. We first uniformly sampled 350 architectures from our search space and trained them using SGD for 150 epochs on ImageNet. Each one of them is fine-tuned for 50 epochs on the

10. Actual values are provided in Table 8.

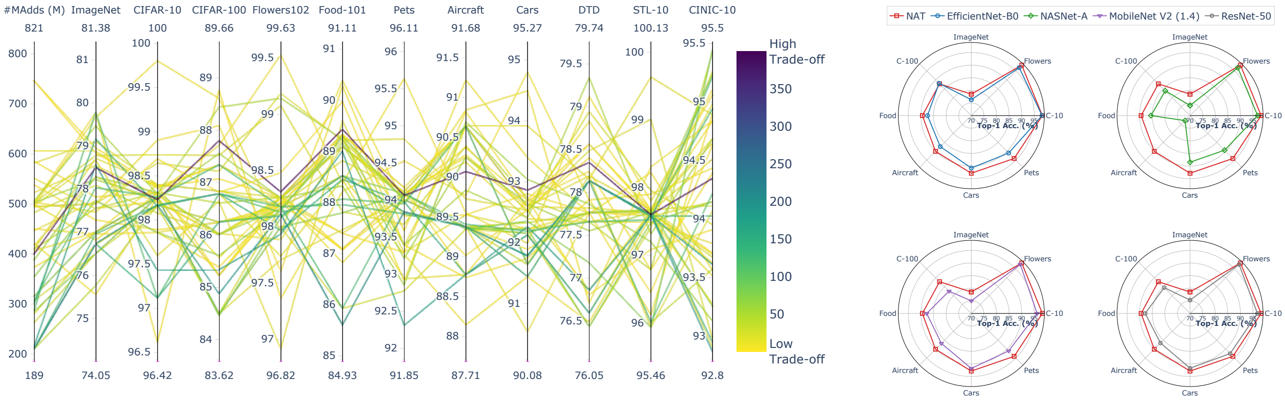


**Fig. 12: Left:** Parallel Coordinate Plot (PCP) where each vertical bar is an objective and each line is a non-dominated architectures achieved by NAT from a 12-obj optimization of minimizing #MAdds and maximizing accuracy on the 11 datasets in Table 3. The model with the best trade-off (see Section A.1 for details) is highlighted in dark blue. **Right:** 1-on-1 comparison between the selected NATNet (top-ranked in trade-off) and representative peer models on top-1 accuracy on various datasets. Method with larger area is better.

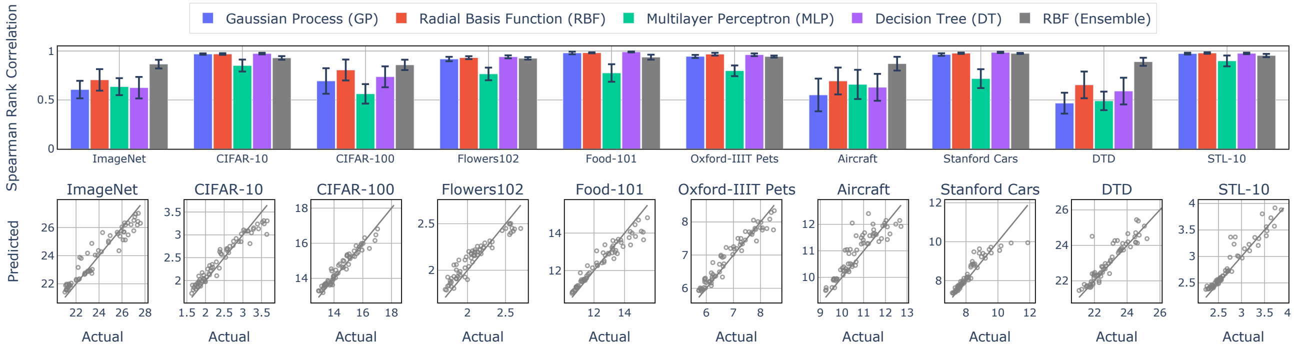


**Fig. 13: Top row:** Spearman rank correlation between predicted accuracy and true accuracy of different surrogate models across many datasets. Each accuracy predictor is constructed from 250 samples (trained architectures). Error bars[10]show mean and standard deviation over ten runs. **Bottom row:** Goodness of fit visualization of RBF ensemble, the best accuracy predictor.

other ten datasets (Table 3). From the 350 pairs of architectures and top-1 accuracy computed on each dataset, we reserved a subset (randomly chosen) of 50 pairs for testing, and the remaining 300 pairs are then available for training the predictor models.

Figure 4 compares the mean (over 11 datasets) Spearman rank correlation between the predicted and true accuracy for each accuracy predictor as the training sample size is varied from 50 to 300. Empirically, we observe that radial basis function (RBF) has Spearman rank correlation compared to the other three models. And, the proposed RBF ensemble model further improves performance over the standalone RBF model across all training sample size regimes. Figure 13 shows a visualization of the comparative performance of predictor models on different datasets. From the trade-off perspective of minimizing number of training examples (which reduces the overall computational cost) and maximizing Spearman rank correlation in prediction (which improves the accuracy in ranking architectures during search), we chose the RBF ensemble as our accuracy predictor model and a training size of 100 for all our experiments.

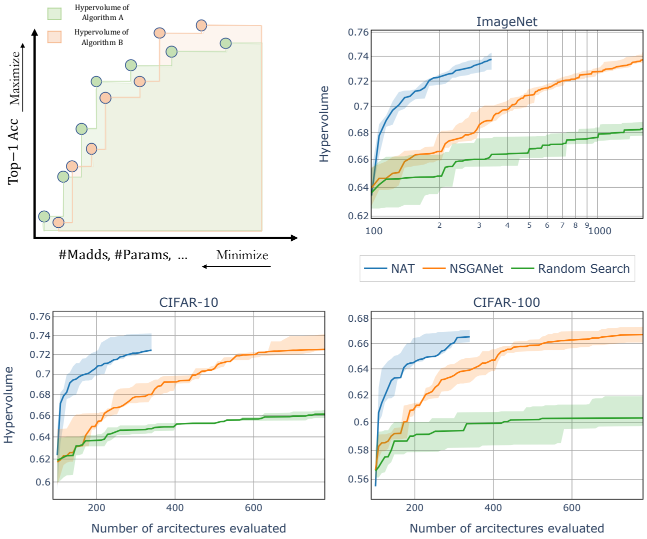


**Fig. 14: Top left:** A sketch visualizing the hypervolume metric [69]. In case of bi-objective, it measures the dominated area achieved by a multi-objective algorithm. A larger hypervolume indicates a better Pareto front achieved. **Rest:** Search efficiency comparison between NAT, NSGANet [49], and random search under a bi-objective setup. Mean hypervolume over five runs are plotted with shaded region showing the standard deviation.

## 5.2 Search Efficiency

The overall computation cost consumed by a NAS algorithm can be factored into three phases: (1) *Prior-search*: Cost incurred prior to architecture search, e.g. training supernet in case of one-shot approaches [10], [28] or constructing accuracy predictor [27], etc; (2) *During-search*: Cost associated with measuring the performance of candidate architectures sampled during search through inference. It also includes the cost of training the supernet in case it is coupled with the search, like in [22] and NAT; (3) *Post-search*: Cost associated with choosing a final architec-

**TABLE 5:** Comparing the relative search efficiency of NAT to other methods. "–" denotes for not applicable, "WS" stands for weight sharing and "SMBO" stands for sequential model-based optimization [70]. $^\dagger$ is taken from [29], $^\ddagger$ estimate based on the # of models evaluated during search (20K in [8], 1.2K in [21], 27K in [35]). * denotes re-ranking stage where top 100-250 models undergo extended training and evaluation for 300 epochs before selecting the final model.

| | Method | Type | Top-1 Acc. (%) | #Madds (M) | Estimated Search Cost (GPU hours) | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Prior-search | During-search | Post-search | Total |
| ImageNet | MnasNet [25] | gradient | 75.2 | 312 | - | - | - | 91k$^\dagger$ |
| | OnceForAll [28] | WS+EA | 76.9 | 230 | 1,200 | 40 | 75 | 1.3k |
| | **NAT (ours)** | WS+EA | **77.5** | **225** | 1,200 | 150 | 75 | 1.4k |
| CIFAR-10 | NASNet [8] | RL | 97.4 | 569 | - | 10,000$^\ddagger$ | 6000* | 16k |
| | PNASNet [21] | SMBO | 96.6 | 588 | - | 600$^\ddagger$ | 36 | 0.6k |
| | DARTS [22] | WS+gradient | 97.3 | 595 | - | 96 | 36 | 0.1k |
| | AmoeabaNet [35] | EA | 97.5 | 555 | - | 13,500$^\ddagger$ | 2400* | 16k |
| | **NAT (ours)** | transfer+EA | **98.4** | **468** | - | 24 | - | 0.1k |

ture, and/or fine-tuning / re-training the final architectures from scratch. For comparison, we select representative NAS algorithms, including those based on reinforcement learning (RL), gradients, evolutionary algorithm (EA), and weight sharing (WS). Table 5 shows results for ImageNet and CIFAR-10. The former is the dataset on which the supernet is trained and the latter is a proxy for transfer learning to a non-standard dataset. NAT consistently achieves better performance, both in terms of top-1 accuracy and model efficiency (e.g. #MAdds), compared to the baselines while computational cost is similar or lower. The primary computational cost of NAT is the *prior-search* training of supernet for 1200 hours. We emphasize, again, that it is a one-time cost that is amortized across all subsequent deployment scenarios (e.g. 10 additional datasets in Section 4.4).

Comparing the search phase contribution to the success of different NAS algorithms is challenging and ambiguous due to substantial disparities in search spaces and training procedures. So, we conduct the following controlled experiment where we replace only the evolutionary search component in the NAT pipeline with (1) a *random search* that uniformly samples (with possible repetition) from the search space, and (2) *NSGANet* [49], another multi-objective EA-based NAS algorithm. This experiment is under a bi-objective setup: maximize top-1 accuracy and minimize #MAdds. We run each method five times on three datasets to capture the variance in performance due to inherent stochasticity in the optimization initialization. We use hypervolume [69], a widely-used metric for comparing algorithms under multiple objectives, as the evaluation metric. Figure 14 shows the mean and standard deviation of the hypervolume achieved by each method. The evolutionary search component in NAT is 3× - 5× more sample efficient than the baselines for the same hypervolume.

### 5.3 Analysis of Crossover

Crossover is a standard operator in evolutionary algorithms, but has largely been avoided by existing EA-based NAS methods [34], [35], [57]. But as we demonstrate here, a carefully designed crossover operation can significantly improve search efficiency. We run the evolutionary search of NAT with and without the crossover operator on four datasets; ImageNet [1], CIFAR-10 [9], Oxford Flowers102 [19], and Stanford Cars [15]. The hyperparameters that we compare are:

1) *w/ crx*: crossover probability of 0.9; mutation probability of 0.1; mutation index $\eta_m$ of 3.
2) *w/o crx*: crossover probability of 0.0; mutation probability of 0.2; mutation index $\eta_m$ of 3.

**TABLE 6:** Effect of fine-tuning for 75 epochs on ImageNet.

| | NAT-M1 | NAT-M2 | NAT-M3 | NAT-M4 |
|---|---|---|---|---|
| w/o fine-tuning | 75.9 | 77.4 | 78.9 | 79.4 |
| w/ fine-tuning | 77.5 | 78.6 | 79.9 | 80.5 |
| Difference($\Delta$) | (+1.6) | (+1.2) | (+1.0) | (+1.1) |

We double the mutation probability when crossover is not used to compensate for the reduced exploration ability of the search. On each dataset, we run each setting to maximize the top-1 accuracy 11 times and report the median performance as a function of the number of architecture sampled in Fig 15a. On all four datasets, the crossover operator significantly improves the efficiency of the evolutionary search algorithm. To further validate, we sweep over the probability of crossover while maintaining the rest of the settings. The median performance (over 11 runs) deteriorates as the crossover probability is reduced from 0.9 to 0.2 (see Fig. 15b).

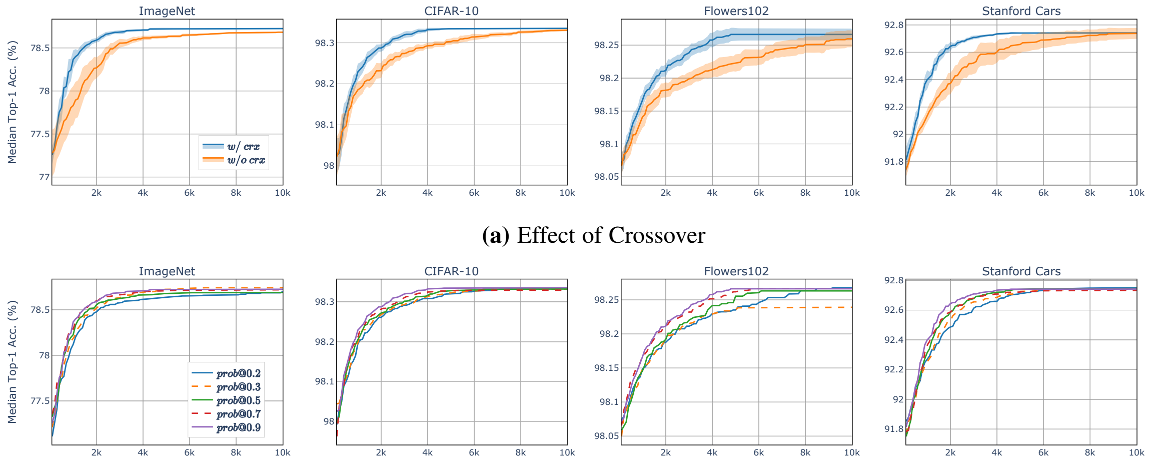
### 5.4 Analysis of Mutation Hyperparameters

The mutation operator used in NAT is controlled via two hyperparameters—namely, the mutation probability $p_m$ and mutation index $\eta_m$. To identify the optimal hyperparameter values, we conduct the following parameter sweep experiments. Setting the rest of the hyperparameters to their default values (see Table 2), we sweep the value of $p_m$ from 0.1 to 0.8, and $\eta_m$ from 1.0 to 20. And for each setting, we run NAT eleven times on four datasets (same as the crossover experiment) to maximize the top-1 accuracy. Figures 16a and 16b show the effect of mutation probability $p_m$ and mutation index $\eta_m$, respectively. We observe that increasing the mutation probability has an adverse effect on performance. Similarly, low values of $\eta_m$, which encourages the mutated offspring to be further away from parent architectures, improves the performance. Based on these observations, we set the mutation probability $p_m$ and mutation index $\eta_m$ parameters to 0.1 and 1.0, respectively, for all our experiments in Section 4.

### 5.5 Effectiveness of Supernet Adaptation

Recall that NAT adopts any supernet trained on a large-scale dataset, e.g. ImageNet and seeks to efficiently transfer to a task-specific supernet on a given dataset. Here we compare this procedure to a more conventional approach of adapting every subnet (candidate architectures in search) directly. Specifically, we consider the following,

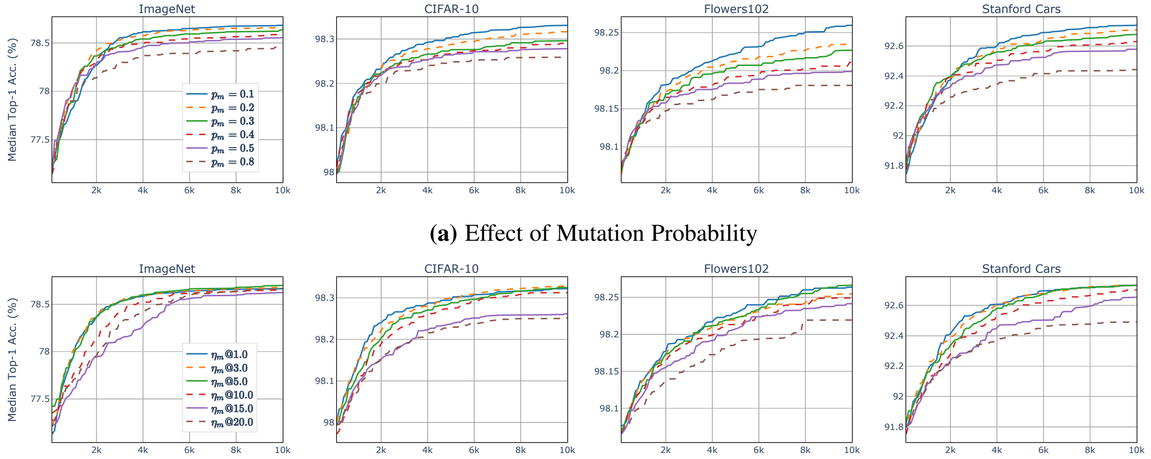
1) *Supernet Adaptation*: fine-tune supernet for 5 epochs in each iteration and use accuracy from inherited weights



**(a)** Effect of Crossover

**(b)** Effect of Crossover Probability

**Fig. 15:** Ablation study on the crossover operator: (a) the median performance from eleven runs of our evolutionary algorithm with and without the crossover operator. (b) the median performance deteriorates as the crossover probability reduces from 0.9 to 0.2.



**(a)** Effect of Mutation Probability

**(b)** Effect of Mutation Hyperparameter $\eta_m$

**Fig. 16:** Hyperparameter study on (a) mutation probability $p_m$ and (b) mutation index parameter $\eta_m$. For each study, we run NAT eleven times on four datasets to maximize top-1 accuracy and report the median performance.

(without further training) to select architectures during search.

2) *Subnet Adaptation*: fine-tune each subnet for 5 epochs from the inherited weights, then measure the accuracy.

We apply these two approaches to a bi-objective search of maximizing top-1 accuracy and minimizing #MAdds on four datasets, including CIFAR-10, CIFAR-100, Oxford Flowers102, and STL-10. Figure 17 compares the final Pareto fronts. Adapting the supernet yields significantly better performance than adapting individual subnets. Furthermore, we select a subset of searched subnets after *subnet adaptation* and fine-tune their weights for an additional 150 epochs. We refer to this as *additional fine-tuning* in Fig. 17. Empirically, we observe that further fine-tuning can match the performance of *supernet adaptation* on datasets with larger training samples per class (e.g. 4,000 in CIFAR-10). On datasets with fewer samples per class (e.g. 20 in Flowers 102), there is still a large performance gap between *supernet adaptation* and *additional fine-tuning*. Overall the results suggest that *supernet adaptation* is more effective on tasks with limited training samples. Finally, we report the effect of fine-tuning the trade-off architectures on ImageNet in Section 4.3 for an additional 75 epochs in Table 6.

## 6 Conclusion

This paper considered the problem of designing custom neural network architectures that trade-off multiple objectives for a given image classification task. We introduced *Neural Architecture Transfer* (NAT), a practical and effective approach for this purpose. We described our efforts to harness the concept of a supernet and an evolutionary search algorithm for designing task-specific neural networks trading-off accuracy and computational complexity. We also showed how to use an online regressor, as a surrogate model to predict the accuracy of subnets in the supernet. Experimental evaluation on eleven benchmark image classification datasets, ranging from large-scale multi-class to small-scale fine-grained tasks, showed that networks obtained by NAT outperform conventional fine-tuning based transfer learning while being orders of magnitude more efficient under mobile settings ($\leq$ 600M Multiply-Adds). NAT was especially effective for small-scale fine-grained tasks where fine-tuning pre-trained ImageNet models is

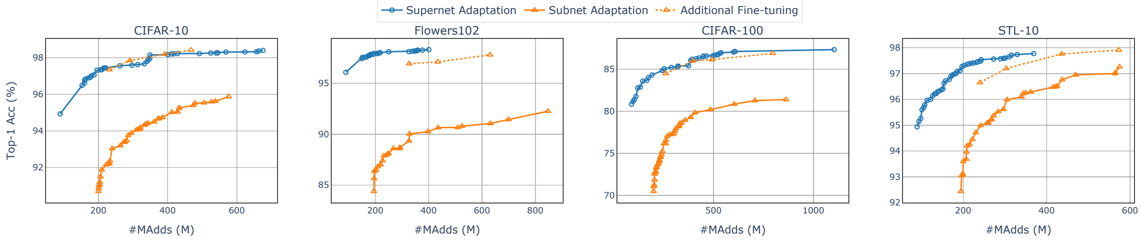


**Fig. 17:** Comparing the performance of *adapting supernet*, *adapting subnet* and *additional fine-tuning* under a bi-objective search setup on four datasets. Details are provided in Section 5.5.

ineffective. Finally, we also demonstrated the utility of NAT in optimizing up to twelve objectives with a subsequent trade-off analysis procedure for identifying a single preferred solution. Overall, NAT is the first large scale demonstration of many-objective neural architecture search for designing custom task-specific models on diverse image classification datasets.

## References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

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

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

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[5] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2017.

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

[7] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in *CVPR*, 2019.

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

[9] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[10] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *ICML*, 2018.

[11] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, Aug 2014.

[12] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.

[13] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[14] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *ECCV*, 2014.

[15] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[16] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," Tech. Rep., 2013.

[17] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *CVPR*, 2014.

[18] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *CVPR*, 2012.

[19] M. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, 2008.

[20] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *ICCV*, 2019.

[21] 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.

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

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

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

[25] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *CVPR*, 2019.

[26] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[27] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *CVPR*, 2019.

[28] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *ICLR*, 2020.

[29] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *CVPR*, 2020.

[30] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

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

[32] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[33] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu, "Practical block-wise neural network architecture generation," in *CVPR*, 2018, pp. 2423–2432.

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

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

[36] D. Lian, Y. Zheng, Y. Xu, Y. Lu, L. Lin, P. Zhao, J. Huang, and S. Gao, "Towards fast adaptation of neural architectures with meta learning," in *ICLR*, 2020.

[37] T. Elsken, B. Staffler, J. H. Metzen, and F. Hutter, "Meta-learning of neural architectures for few-shot learning," in *CVPR*, 2020.

[38] M. Wistuba, "Xfernas: Transfer neural architecture search," *arXiv preprint arXiv:1907.08307*, 2019.

[39] J. Fang, Y. Chen, X. Zhang, Q. Zhang, C. Huang, G. Meng, W. Liu, and X. Wang, "Eat-nas: Elastic architecture transfer for accelerating large-scale neural architecture search," *arXiv preprint arXiv:1901.05884*, 2019.

[40] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural automl," in *NeurIPS*, 2018.

[41] E. Kokiopoulou, A. Hauth, L. Sbaiz, A. Gesmundo, G. Bartok, and J. Berent, "Fast task-aware architecture inference," *arXiv preprint arXiv:1902.05781*, 2019.

[42] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

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

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

[45] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *CVPR*, 2019.

[46] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *ICLR*, 2020.

[47] A. Brock, T. Lim, J. Ritchie, and N. Weston, “SMASH: One-shot model architecture search through hypernetworks,” in *ICLR*, 2018.

[48] 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.

[49] 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.

[50] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, “Dpp-net: Device-aware progressive search for pareto-optimal neural architectures,” in *ECCV*, 2018.

[51] X. Chu, B. Zhang, R. Xu, and J. Li, “Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search,” *arXiv preprint arXiv:1907.01845*, 2019.

[52] Z. Lu, K. Deb, and V. N. Boddeti, “MUXConv: Information multiplexing in convolutional neural networks,” in *CVPR*, 2020.

[53] J. Bracken and J. T. McGill, “Mathematical programs with optimization problems in the constraints,” *Operations Research*, vol. 21, no. 1, pp. 37–44, 1973. [Online]. Available: http://www.jstor.org/stable/169087

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

[55] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[56] 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.

[57] 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.

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

[59] I. Das and J. E. Dennis, “Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems,” *SIAM J. on Optimization*, vol. 8, no. 3, p. 631657, Mar. 1998. [Online]. Available: https://doi.org/10.1137/S1052623496307510

[60] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, “Single path one-shot neural architecture search with uniform sampling,” *arXiv preprint arXiv:1904.00420*, 2019.

[61] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, “Randaugment: Practical automated data augmentation with a reduced search space,” *arXiv preprint arXiv:1909.13719*, 2019.

[62] J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang, “Atomnas: Fine-grained end-to-end neural architecture search,” in *ICLR*, 2020.

[63] Y. Li, X. Jin, J. Mei, X. Lian, L. Yang, C. Xie, Q. Yu, Y. Zhou, S. Bai, and A. Yuille, “Neural architecture search for lightweight non-local networks,” in *CVPR*, 2020.

[64] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, “Blockwisely supervised neural architecture search with knowledge distillation,” in *CVPR*, 2020.

[65] M. Tan and Q. V. Le, “Mixconv: Mixed depthwise convolutional kernels,” in *BMVC*, 2019.

[66] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le, “Bignas: Scaling up neural architecture search with big single-stage models,” *arXiv preprint arXiv:2003.11142*, 2020.

[67] X. Wang, D. Kihara, J. Luo, and G.-J. Qi, “Enaet: Self-trained ensemble autoencoding transformations for semi-supervised learning,” *arXiv preprint arXiv:1911.09265*, 2019.

[68] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, “Xnas: Neural architecture search with expert advice,” in *NeurIPS*, 2019.

[69] E. Zitzler and L. Thiele, “Multiobjective optimization using evolutionary algorithms — a comparative case study,” in *Parallel Problem Solving from Nature — PPSN V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 292–301.

[70] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523.

[71] K. Deb, *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: Wiley, 2001.

# Appendix A

In this appendix we include: (i) an “unsupervised” method for choosing an architecture based on trade-offs in Section A.1, (ii) additional comparison to existing convolutional neural networks on ImageNet [1] in Section A.2, and (iii) visualization of the obtained architectures in Section A.3.

## A.1 Choosing Best Trade-off Solution

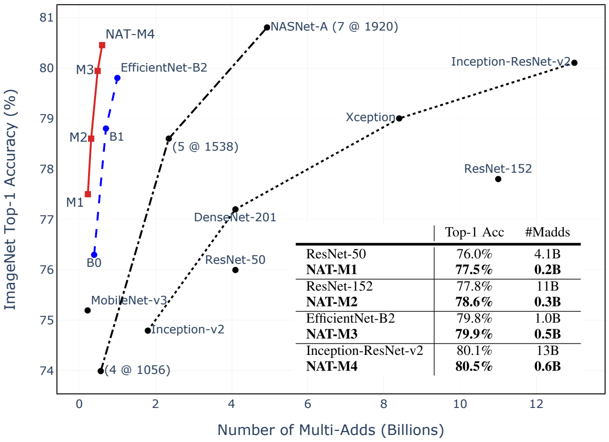
The proposed many-objective EA is expected to produce $N$ (population size) solutions trading-off all $m$ objectives. These solutions are guaranteed to have one property: a gain in one objective between $i$-th and $j$-th solutions comes only from a loss in at least one other objective between them. We calculate the trade-off of $i$-th solution as the average loss per unit average gain among $m$ nearest neighbors ($B(\boldsymbol{i})$) based on normalized Euclidean distance are used here), as follows [71]:

$$\text{Trade-off}(\boldsymbol{i}) = \max_{j=1}^{|B(\boldsymbol{i})|} \frac{\text{Avg.Loss}(\boldsymbol{i},\boldsymbol{j})}{\text{Avg.Gain}(\boldsymbol{i},\boldsymbol{j})} \tag{4}$$

where

$$\begin{aligned} \text{Avg.Loss}(\boldsymbol{i},\boldsymbol{j}) &= \frac{\sum_{k=1}^{m} \max\left(0, f_k(\boldsymbol{j}) - f_k(\boldsymbol{i})\right)}{\sum_{k=1}^{m} \{1 | f_k(\boldsymbol{j}) > f_k(\boldsymbol{j})\}} \\ \text{Avg.Gain}(\boldsymbol{i},\boldsymbol{j}) &= \frac{\sum_{k=1}^{M} \max\left(0, f_k(\boldsymbol{i}) - f_k(\boldsymbol{j})\right)}{\sum_{k=1}^{m} \{1 | f_k(\boldsymbol{i}) > f_k(\boldsymbol{j})\}} \end{aligned}$$

Thereafter, the solutions having the highest trade-off value indicates that it causes the largest average loss in some objectives to make a unit average gain in other objectives to choose any of its neighbors. If this highest trade-off value is much larger statistically than other solutions, then the highest trade-off solution is the preferred choice, in case of no preferences provided from users.

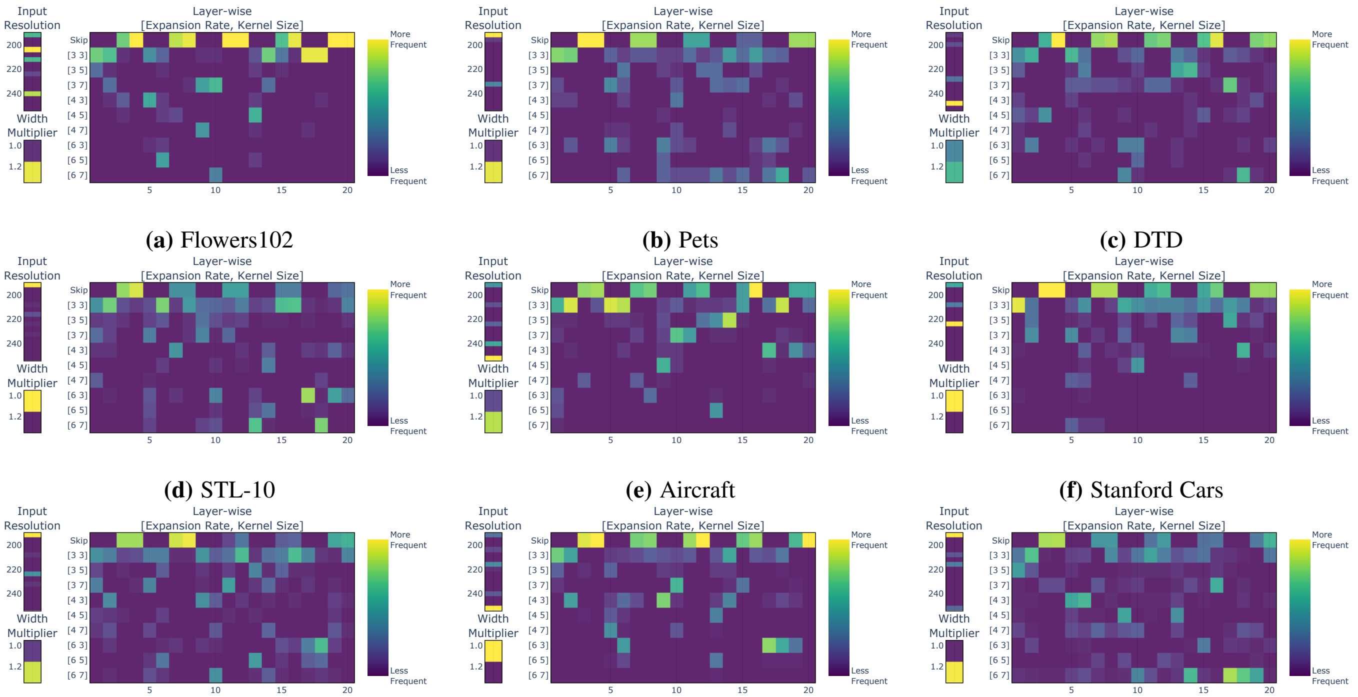


| | Top-1 Acc | #Madds |
|---|---|---|
| ResNet-50 | 76.0% | 4.1B |
| **NAT-M1** | **77.5%** | **0.2B** |
| ResNet-152 | 77.8% | 11B |
| **NAT-M2** | **78.6%** | **0.3B** |
| EfficientNet-B2 | 79.8% | 1.0B |
| **NAT-M3** | **79.9%** | **0.5B** |
| Inception-ResNet-v2 | 80.1% | 13B |
| **NAT-M4** | **80.5%** | **0.6B** |

**Fig. 18: MAdds vs. ImageNet Accuracy**. Our NATNets significantly outperforms other models from NAS algorithms and human experts. In particular, NAT-M4 achieves new state-of-the-art 80.5% top-1 accuracy under mobile setting (600M MAdds).

## A.2 Comparison to Existing ConvNets

Figure 18 visualizes the #MAdds-accuracy trade-off curve, where our NATNets achieve better top-1 accuracy with much fewer #MAdds than other CNN models. Notably, NAT-M1 is more accurate, but **20x more efficient** in #MAdds than ResNet-50 [3]; NAT-M4 is more accurate, but **21x more efficient** in #MAdds than Inception-ResNet-v2 [5].



**(a)** Flowers102 **(b)** Pets **(c)** DTD

**(d)** STL-10 **(e)** Aircraft **(f)** Stanford Cars

**(g)** CIFAR-10 **(h)** Food-101 **(i)** ImageNet

**Fig. 19:** Non-dominated architectures to {top-1 accuracy, #MAdds} obtained by NAT on different datasets.

**TABLE 7:** NAT model performance corresponding to Fig. 9.

| Flowers102 [19] | | | Oxford-IIIT Pets [18] | | | DTD [17] | | | STL10 [13] | | | FGVC Aircraft [16] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) |
| 3.3M | 152M | 97.5 | 4.0M | 160M | 91.8 | 2.2M | 136M | 76.1 | 4.4M | 240M | 96.7 | 3.2M | 175M | 87.0 |
| 3.4M | 195M | 97.9 | 5.5M | 306M | 93.5 | 4.0M | 297M | 77.6 | 5.1M | 303M | 97.2 | 3.4M | 235M | 89.0 |
| 3.7M | 250M | 98.1 | 5.7M | 471M | 94.1 | 4.1M | 347M | 78.4 | 7.5M | 436M | 97.8 | 5.1M | 388M | 90.1 |
| 4.2M | 400M | 98.3 | 8.5M | 744M | 94.3 | 6.3M | 560M | 79.1 | 7.5M | 573M | 97.9 | 5.3M | 581M | 90.8 |
| Stanford Cars [15] | | | CIFAR-100 [9] | | | CIFAR-10 [9] | | | Food-101 [14] | | | CINIC-10 [12] | | |
| #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) | #Params | #MAdds | Top-1 Acc (%) |
| 2.4M | 165M | 90.9 | 3.8M | 261M | 86.0 | 4.3M | 232M | 97.4 | 3.1M | 198M | 87.4 | 4.6M | 317M | 93.4 |
| 2.7M | 222M | 92.2 | 6.4M | 398M | 87.5 | 4.6M | 291M | 97.9 | 4.1M | 266M | 88.5 | 6.2M | 411M | 94.1 |
| 3.5M | 289M | 92.6 | 7.8M | 492M | 87.7 | 6.2M | 392M | 98.2 | 3.9M | 299M | 89.0 | 8.1M | 501M | 94.3 |
| 3.7M | 369M | 92.9 | 9.0M | 796M | 88.3 | 6.9M | 468M | 98.4 | 4.5M | 361M | 89.4 | 9.1M | 710M | 94.8 |

**TABLE 8:** Accuracy predictor model mean (standard deviation) performance corresponding to Fig. 13.

| Method | ImageNet [1] | CIFAR-10 [9] | CIFAR-100 [9] | Flowers102 [19] | Food-101 [14] | Oxford-IIIT Pets [18] | Aircraft [16] | Stanford Cars [15] | DTD [17] | STL-10 [13] |
|---|---|---|---|---|---|---|---|---|---|---|
| GP | 0.606 (0.09) | 0.969 (0.01) | 0.693 (0.13) | 0.918 (0.02) | 0.980 (0.01) | 0.945 (0.02) | 0.551 (0.17) | 0.964 (0.01) | 0.467 (0.11) | 0.973 (0.11) |
| RBF | 0.705 (0.11) | 0.969 (0.01) | 0.806 (0.08) | 0.932 (0.03) | 0.981 (0.01) | 0.967 (0.01) | 0.693 (0.08) | 0.977 (0.01) | 0.653 (0.06) | 0.979 (0.01) |
| MLP | 0.635 (0.09) | 0.851 (0.06) | 0.562 (0.10) | 0.766 (0.06) | 0.775 (0.09) | 0.798 (0.05) | 0.658 (0.15) | 0.717 (0.10) | 0.490 (0.09) | 0.899 (0.06) |
| DT | 0.625 (0.11) | 0.974 (0.01) | 0.736 (0.11) | 0.940 (0.02) | 0.990 (0.01) | 0.961 (0.01) | 0.629 (0.14) | 0.986 (0.01) | 0.590 (0.14) | 0.976 (0.01) |
| RBF Ensemble | 0.866 (0.04) | 0.959 (0.02) | 0.858 (0.05) | 0.931 (0.01) | 0.967 (0.03) | 0.943 (0.01) | 0.870 (0.07) | 0.975 (0.01) | 0.890 (0.04) | 0.964 (0.02) |

## A.3 Architecture Visualization

One of the main advantages of multi-objective optimization is that it generates a set of non-dominated solutions in a single run. These non-dominated solutions are special in the sense that one has to sacrifice on one objective to gain on another. Thereby, "mining" on these non-dominated solutions oftentimes yields important design principles for the task at hand, in this case, to efficiently construct an architecture specific to the objectives and dataset. To demonstrate this concept, we visualize the non-dominated architectures (to maximize top-1 accuracy and minimize #MAdds) resulting from NAT on a diverse set of datasets in Fig. 19. Each sub-figure is a heat map showing the distribution of the searched, input image resolutions, width multipliers, and layer settings.

It is clear from Fig. 19 that even under the same objectives, the optimal architectures for different datasets are different. For example, the most frequent input image resolution is 192 (the lowest value in our searched options) for Oxford-IIIT Pets [18] and STL-10 [13]. While on FGVC Aircraft [16] and Food-101 [14], the most frequent choice of resolution is 256, which is the highest value in our searched option. Similar observations can be made in case of width multiplier and layer settings. This example provides empirical evidence necessary for finding dataset-specific optimal architectures, as opposed to conventional transfer learning. And as demonstrated in the main paper, our proposed NAT presents an efficient and effective way to achieve this goal.