Debiased One-shot NAS via Density-aware Sampling

Mehraveh Javan Roshtkhari Matthew Toews Marco Pedersoli
École de technologie supérieure
Montreal, Canada

Abstract

One-shot Neural Architecture Search (NAS) is based on training a supernet, a single model from which many different architectures with shared weights are sampled and updated during training. Training the shared weights of the supernet is much more computationally efficient than training each architecture independently. However, during the supernet training, architectures with similar gradients (dense regions in the gradient space) would cooperate with each other, increasing their training, while architectures in sparse (low-density) regions would not receive as much training benefit from others. This does not allow all architectures to be trained with the same amount of effective updates, resulting in a training bias that favors architectures in denser regions. As a consequence, the correlation between the supernet estimations and the actual performance of models independently trained is reduced. This negatively affects the supernet's ability to select good architectures once trained. In this paper, we propose two computationally feasible ways for different computational budgets and search spaces to approximate the architecture densities and implement a density-aware debiasing mechanism for supernet training. We propose a fine-grained density calculation for numerable smaller search spaces and an online density approximation based on density prototypes from a clustering algorithm for larger spaces. We validate our method on CIFAR10, CIFAR100, and ImageNet datasets using various search strategies and show consistently improved results compared to several single-path one-shot supernet training methods.

1. Introduction

Neural architecture search (NAS) that aims to automate the design of neural networks has shown great success for computer vision tasks such as image classification [6, 31, 42]. Part of this success is due to the introduction of efficient training methods such as one-shot NAS, in which a single compositional neural network (supernet), jointly trains different architectures (paths or subnets) from a pre-defined search space [21].

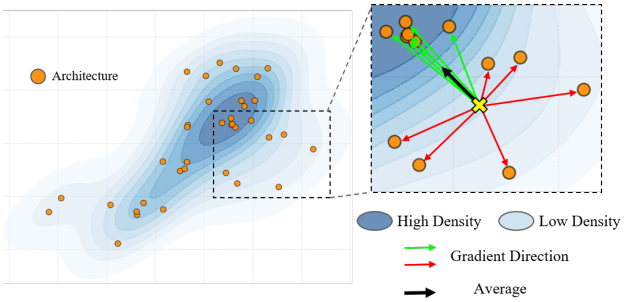

Figure 1. **Training bias in uniform sampling supernet training.** 2D projections of architecture gradients from the Pooling benchmark. When training with a uniform sampling of architectures, we found that architectures that share similar gradients (high-density gradient regions) tend to be globally over-trained, while the ones with different gradients (low-density gradient regions) will be under-trained. This produces a bias that favors architectures that lie in denser gradient regions.

Common one-shot NAS methods based on weight-sharing, use the supernet as a proxy to efficiently estimate architecture performances on a validation set. By training the supernet, a single set of shared weights learns to effectively parametrize a large set of diverse architectures, so that the inherited weights can adequately evaluate and rank those architectures. Therefore, the core assumption is that in single-path one-shot NAS, similar architectures can mutually benefit when one of them is sampled, requiring fewer training iterations to jointly train all architectures.

However, in one-shot NAS, the updates of certain architectures might be detrimental to others [33, 39]. This effect creates the well-known performance gap and poor rank correlation between supernet evaluations and independently trained architectures [20, 23]. Most crucially, this interference is not random: it depends on the similarity of the gradient updates. As demonstrated in Fig. 1, architectures with similar gradients will produce a positive interference, while architectures with different gradients will produce a negative interference. Therefore, architectures that have gradients similar to several neighbors (i.e. high-density regions) tend to help each other. In contrast, architectures with differ-

ent gradients from their neighbors (i.e. low-density regions) have a destructive effect on their gradient updates. Globally, this effect produces a clear bias in which denser regions will produce over-trained architectures, while lower density regions will have under-trained architectures. The main goal of this work is to tackle this bias.

We propose a computationally affordable approach to reduce the bias induced by weight sharing during supernet training. To achieve this, we present two algorithms to estimate the density of architectures in gradient space. The first, designed for smaller search spaces, directly computes gradient density following a brief warm-up period of uniform training. The second is designed for large search spaces, which approximates the density online using a learned set of prototypes. The estimated density is then used for importance sampling by prioritizing architectures with low density and down-sampling architectures in high-density regions. Finally, as computing gradients of large neural networks can be expensive, we show that similar results can be obtained by substituting the gradient with the output layer of a network. We empirically show that architecture distances on this much smaller space are highly correlated with gradient distances, but are cheaper to compute.

The main contributions of this work are the following:

- For the first time, we show that for one-shot NAS, architectures with similar gradients (denser regions) are over-trained and architectures with different gradients (low-density regions) are under-trained. This affects the capability of the sampling method (such as uniform sampling) to approximate the real performance of the NAS architectures.
- We devise a density-aware sampling framework to mitigate this bias, and introduce two algorithms for efficiently estimating the gradient density. We further improve the framework by showing that gradient density can be adequately replaced with functional density based on model outputs, reducing the computational overhead with little loss in performance.
- We validate our model using various search strategies on different datasets and search spaces and show consistent performance improvement compared to other methods that aim to improve fairness or supernet consistency.

In the following sections, we present several strategies and design choices for efficient implementation of density-aware debiasing for supernet training (Sec. 3). We then compare our approach with the related work using experiments and analysis on different search spaces and models (Sec. 4).

2. Related Work

Weight sharing in NAS Early NAS approaches based on reinforcement learning [42] or evolutionary algorithms (EA) [22] encountered substantial computational cost to evaluate candidate architectures. To reduce the cost, Pham et al. [21] proposed one-shot NAS based on weight sharing, which has since become one of the most popular NAS techniques.

However, the ranking correlation obtained from the supernet with independent training (ground truth) can be poor [35, 36] due to co-adaptation and interference of optimizing shared weights for different architectures. This interference increases as more weights are shared [27]. Therefore, one strategy is to reduce weight-sharing among architectures, such as by shrinking the search space [2, 9, 32], or using a specialized supernet for different parts of the search space [8, 23, 40]. In general, methods using weight sharing can be divided into two main approaches:

- One stage methods such as DARTS [16] that perform supernet training and architecture search simultaneously.
- Two stage methods such as SPOS [6], FairNas [3], and PA&DA [17] that perform NAS in two stages: 1) supernet training followed by 2) architecture search (*e.g.* random, EA, Monte-Carlo tree search methods).

One stage methods allow the training to focus from the beginning on certain parts of the search space, effectively reducing the weight sharing at the cost of introducing bias. Two stage methods ensure more fairness in the training stage and can readily support searching multiple architectures under different constraints [1]. However, as we discuss in the following sections, they are not entirely bias-free.

In general, the aim of the first stage is to train a reliable and unbiased supernet to apply a search strategy in the second stage to obtain the final architecture. Therefore, the effectiveness of this method relies on both the quality of the supernet and the efficiency of the search algorithm. Many works focus on improving the search stage [30, 37, 41], our work focuses on improving the supernet training for less bias and better exploration of the search space.

Improving single-path supernet training Single-path sampling is the most commonly used sampling strategy for supernet training. It adapts a discrete search space and only samples one subnet from the supernet at each iteration, reducing the memory cost to a single subnet as well as helping in the decoupling of shared weights [6]. The standard and widely used sampling distribution is uniform, since it gives an equal chance of training to all architectures. It can be used either as the first stage supernet training [1, 3, 6, 30, 41] in two-stage methods, or as a warm-up for further supernet training [24, 25, 28]. While uniform sampling is simple and provides a strong baseline, it is not bias-free.

FairNas [3] shows that uniformly sampling architectures causes unfair training opportunities throughout training among different operations in the search space. This imbalanced training results in inaccurate architecture estimations and poor rank correlation. They propose to enforce strict fairness to ensure that every operation is updated the same

number of times at any point during training. ShiftNas [38] addresses bias in terms of resource allocation to subnets, where most uniformly sampled architectures have intermediate computational costs. They propose to adjust sampling probability based on subnet complexity to accommodate more complex subnets with more resources, while Jeon et al. [11] uses a dynamic complexity-aware learning rate scheduler.

Ning et al. [20] analyzed the gradient dynamics in supernet training, showing both positive and negative effects of gradient interference. Several other works aim to improve training using supernet gradients. Hu et al. [8] and Ly-Manson et al. [18] reduce gradient conflict directly by grouping operations using gradient matching and assigning a separate set of weights to each group. Other works modify the sampling: PA&DA [17] propose to increasingly sample subnets/data that minimize gradient variance to increase ranking consistency, Ma et al. [19] propose to project the gradient of a newly sampled architecture to be orthogonal to previously sampled architectures, GreedyNAS [10, 34] samples high-performing architectures more frequently. However, these methods introduce more bias in sampling toward certain architectures. Our method improves the supernet from a different direction than these methods that directly focus on stability and increased consistency.

Our work focuses on improving the supernet training by reducing the bias from the perspective of density. FairNas is concerned with fairness in training opportunity, and other works consider fairness based on complexity [11, 38]. We are concerned with the fairness in *effective* training of each architecture. Finally, while not the main objective, our debiasing method can be potentially seen as a form of increasing novelty and better exploration of search space by prioritizing outliers (low-density areas) during training.

3. Method

The supernet training bias is a direct consequence of weight-sharing, independent of the training approach and search space design. While our density-aware debiasing method may work in combination with other training algorithms, for the sake of simplicity and generality, we show its principle by analyzing standard supernet training with uniform sampling.

3.1. Uniform Sampling

Uniform sampling is the simplest approach for single-path one-shot (SPOS) NAS. It does not require keeping track of any additional parameters for operations/subnets and can scale well to very large search spaces. This makes it the default choice for the supernet training stage or as a warm-up to more complex algorithms. Therefore, improving this baseline can directly boost many different NAS methods. Following Guo et al. [6], in single-path supernet training, at each iteration, one architecture a is uniformly sampled from the search space $\mathcal{A}$. For a given minibatch of training data $\mathcal{X}_{tr} = \{x_0, x_1, ..., x_B\}$ and the corresponding labels $\mathcal{Y}_{tr} = \{y_0, y_1, ..., y_B\}$, the supernet f_a with weights w is trained to minimize the training loss:

$$\sum_{i=0}^{B} \mathcal{L}(f_a(w, x_i), y_i), \quad a \sim \mathcal{U}(0, |\mathcal{A}| - 1). \tag{1}$$

After the supernet training stage, to find the final architecture a^*, any search algorithm may be used to efficiently evaluate architectures by inheriting the weights w^* of the trained supernet:

$$a^* = \arg\max_{a \in \mathcal{A}} \; Acc(f_a(w^*, \mathcal{X}_{val})), \tag{2}$$

where Acc is the accuracy or any metric used to estimate the quality of the architecture a on the validation data $\mathcal{X}_{val}$.

3.2. Biased Sampling

Sampling from a uniform distribution implies equal attention to each subnet; however, due to gradient dynamics of shared weights, the effective optimization among subnets is not equal. Consider the gradient of the model weights g_a for the architecture a on a training minibatch:

$$g_a = \nabla_w \sum_{i=0}^{B} \mathcal{L}(f_a(w, x_i), y_i), \quad a \sim \mathcal{U}(0, |\mathcal{A}| - 1). \tag{3}$$

Assuming for simplicity that all weights w are fully shared across all subnets[1]. For two architectures i and j, their gradient alignment can be defined using cosine similarity:

$$s(g_i, g_j) = \frac{g_i \cdot g_j}{||g_i||_2 ||g_j||_2}. \tag{4}$$

By definition, if the gradients of two architectures align ($s(g_i, g_j) \approx 1$), a training step for i will have a positive transferring effect on j, effectively jointly optimizing for j, even though j is not directly sampled. Considering this communication among gradients, the only way that every architecture would receive equal effective updates on the weights would be with mutually orthogonal gradients ($s(g_i, g_j) = 0$). However, this implies independent training of subnets, which goes against the core premise of weight sharing, in which multiple architectures leverage their similar gradient updates to accelerate training and learn a set of generalized weights. In contrast, in our proposed approach, we still train with shared weights, but we estimate the effective gradient update and compensate for that to make the training fairer.

[1]For instance, when using convolutional filters and changing the dilations or strides. In our experiments section, we address how to handle when parameters are not fully shared.

3.3. Bias Estimation Based on Gradient Density

To reduce the bias in supernet training, we need to estimate whether a given architecture is generally over-trained or under-trained. To globally assess an architecture, we estimate the density of its gradients. We consider a dense region as one where there are many architectures with similar gradients, which induces overtrained architectures. In lower-density regions, few architectures have similar gradients and are therefore under-trained.

Density estimation A straightforward and intuitive way to measure the density of a given architecture a is then to use its mean distance to other architectures j in the search space:

$$\bar{d}_a = \frac{1}{|\mathcal{A}| - 1} \sum_{j \in \mathcal{A}, j \neq a} d(a, j), \tag{5}$$

where $d(a, j)$ is a suitable distance metric, such as cosine distance of gradients $d(a, j) = 1 - s(g_a, g_j)$ calculated on the same batch of data. Density α_a is then estimated as $\alpha_a = \frac{1}{\bar{d}_a}$.

3.4. Debiased Sampling for Numerable Architectures

In this section, we consider the case of a search space with a known and numerable set of architectures. To reduce the bias in supernet training, we need to account for the density distribution in the search space. The density-aware sampling rate of architecture a should be inversely proportional to its estimated density α_a. The debiased training is therefore:

$$\sum_i \mathcal{L}(f_a(w, x_i), y_i), \quad a \sim \mathcal{M}_\tau(\bar{d}), \tag{6}$$

with $\mathcal{M}_\tau$ the density-aware probability distribution and its sharpness controlled by the temperature parameter τ. For sufficiently small search spaces (depending on available computational resources), it is possible to estimate α_a for every architecture. Therefore, we define $\mathcal{M}$ directly using categorical distribution:

$$\mathcal{M}_\tau(a) = \frac{\exp(\bar{d}_a/\tau)}{\sum_{i \in A} \exp(\bar{d}_i/\tau)}. \tag{7}$$

Algorithm 1 outlines the debiased supernet training process.

3.5. Debiased Sampling for Larger Search Spaces

For some search spaces, it is feasible to estimate the density of each architecture and store it in memory, leading to a fine-grained probability distribution $\mathcal{M}(\alpha)$. However, in very large search spaces, the entire search space cannot be covered. Furthermore, in some search spaces, as the supernet is trained, the density distribution of the search space may evolve. To account for these, we propose an online density approximation strategy coupled with supernet training.

Algorithm 1: Supernet training with density-aware sampling for small search spaces

input : $\mathcal{A}$: search space, $f_a(w)$: supernet with sampled subnet a, $iter_w$/$iter_t$: warm-up/total iterations, d: distance metric, τ: softmax temperature

```
# warm-up
while iter < iter_w do
    sample a ~ U(0, |A| − 1))
    train f_a(w)
end
# density calculation
for i ∈ A do
    d̄_a = 1/(|A|−1) Σ_{j∈A, j≠a} d(a, j)
end
# probability distribution estimation
M_τ(a) = exp(d̄_a/τ)/Σ_{i∈A} exp(d̄_i/τ)
# train with debiased sampling
while iter < iter_t do
    sample a ~ M_τ
    train f_a(w)
end
```

output : trained supernet $f(w^*)$

Density Prototypes To avoid comparing each architecture with every other one, we define a small, fixed set of representative prototypes $\mathcal{C} = \{c_1, c_2, ..., c_K\}$. These prototypes are designed to capture the overall structure of the search space. The set can be initialized by running a K-Means clustering algorithm on an adequately large, random sample of architecture representations (*e.g.* gradients) to obtain centroids and form $\mathcal{C}$. The density of each architecture is then approximated by measuring its relationship only to those prototypes, by replacing $\mathcal{A}$ with $\mathcal{C}$ in Eq. (5). This reduces the density computational cost from $\propto |\mathcal{A}|$ to $\propto K$.

Updating Prototypes The prototypes are continuously updated throughout training to adapt to more samples becoming available and changing representations as supernet weights are optimized. Given an architecture a, sampled during training, $\mathcal{C}$ is updated by a soft-assignment rule. The normalized assignment for prototype c_k update is determined by:

$$\beta_k = \frac{d^2(a, c_k)}{\sum_{j=0}^{K-1} d^2(a, c_j)}, \ \forall k \in [0, K-1]. \tag{8}$$

Each prototype c_k is then updated using an exponential moving average with λ controlling the weight of the new sample as:

$$c_k \leftarrow c_k + \lambda \cdot \beta_k \cdot (a - c_k), \ \forall c_k \in \mathcal{C}. \tag{9}$$

Algorithm 2: Debiased supernet training with density-aware sampling for large search spaces

input : $\mathcal{A}$: search space, $f_a(w)$: supernet with sampled subnet a, $iter_w$/$iter_t$: warm-up/total iterations, d: distance metric, τ: softmax temperature, x_v: validation batch, K: number of prototypes, N: initial sample size

```
# warm-up
while iter < iter_w do
    sample a ~ U(0, |A| − 1))
    train f_a(w)
end
# prototype initialization
sample {a_1, ..., a_N} ~ U(0, |A| − 1)
C ← K-Means clustering {a_1, ..., a_N} on x_v
# debiased training
while iter < iter_t do
    sample a ~ U(0, |A| − 1))
    d̄_a = 1/(K−1) Σ_{j∈C} d(a, j)
    α_a ← 1/d̄_a
    # supernet training
    train f(a) with sigmoid((α_mid − α_a)/τ)
    # prototype update
    β_k = d²(a, c_k) / Σ_{j=0}^{K−1} d²(a, c_j)
    c_k ← c_k + λ · β_k · (a − c_k)
end
```

output : trained supernet $f(w^*)$

Supernet Training To achieve a debiased training of the supernet, the contribution of an architecture should be scaled inversely to its density:

$$P_{train}(a) = \text{sigmoid}\left(\frac{\alpha_{\text{mid}} - \alpha_a}{\tau}\right). \tag{10}$$

$P_{train}(a)$ can be used to make a probabilistic decision to accept or reject a sample for training. Given enough iterations, this sampling method will approximate the density-aware sampling distribution. We show the supernet training pipeline in Algorithm 2.

3.6. Density Estimation with Functional Similarity

As discussed, a suitable distance metric $d(\cdot, \cdot)$ for Eq. (5) is the cosine distance of gradients, which provides the most direct measure of bias in supernet training. However, they can only be reliably calculated on shared weights and require costly backpropagation. Therefore, we seek a cheaper alternative representation for density estimations. We use output features as the summary of architectures and calculate functional distance [12] as:

$$d(a_i, a_j) = 1 - \frac{f(a_i) \cdot f(a_j)}{||f(a_i)||||f(a_j)||} \tag{11}$$

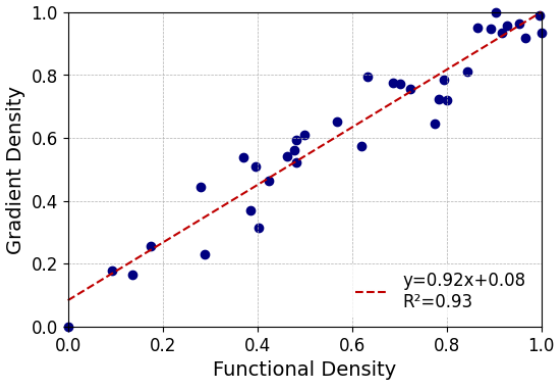

Figure 2. **Gradient density vs. functional density for Pooling benchmark.** Density estimated from gradients show strong correlation with functional density from outputs.

In Fig. 2, we show the calculated gradient and functional density from Eq. (5) for the Pooling search space. The correlation implies that functional density can be used as an alternative to gradient density. On NAS-Bench-Macro, another benchmark, our estimation shows a strong correlation (~ 0.9) for these values. We use functional density as the default for our experiments and explore other representations for density estimation in the experiments section.

4. Experiments

For numerable search spaces, we evaluate our method (Algorithm 1) on two benchmarks for CIFAR10 [14]: Pooling benchmark [23], and NAS-Bench-Macro [28]. For larger search space (Algorithm 2), we first use our method for NAS-Bench-201 [5] and then expand to MobileNet-V2 [26] search space on ImageNet [4]. For each search space, we compare our method with various combinations of sampling methods during supernet training (stage 1), and different search methods (stage 2) after training. For more information on search spaces, search methods, and experimental details, see the appendix.

4.1. Experiments for Numerable Search Spaces

4.1.1. Search Space

Pooling benchmark is a Resnet-based [7] macro search space where only downsampling (pooling) is optimized. Weights are fully shared among architectures, making one-shot NAS very challenging on this search space [23], and suitable for investigating the weight sharing mechanism.

NAS-Bench-Macro is another benchmark that optimizes macro structure by searching among 3 candidate operations for 8 layers, resulting in a total size of $3^8 = 6,561$.

Table 1. CIFAR10 results on (top) Pooling search space [23] (bottom) NAS-Bench-Macro [28]. We report the average performance of top-k architectures, the final architecture accuracy found by BSE, and rank correlation with ground truth (Kendall's Tau) for different supernet sampling methods.

Training Method	Top-1	Top-5	Top-10	BSE	KT
Pooling Benchmark					
Default	90.52	-	-	-	-
Uniform	90.51	90.46	90.54	90.42	0.16
Strict fairness	91.05	90.63	90.31	91.70	0.21
Debiased (ours)	91.83	90.92	90.59	92.01	0.34
Best	92.01	-	-	-	-
NAS-Bench-Macro					
Uniform	92.03	91.72	91.52	92.09	0.72
Strict fairness	92.47	91.45	91.22	92.51	0.72
Debiased (ours)	92.56	92.02	91.64	93.13	0.74
Best	93.13	-	-	-	-

4.1.2. Supernet Training

We compare debiased supernet training (our method) with uniform training and training with strict fairness (FairNas). In Tab. 1, we show the results of top-k architectures found after supernet training. Due to the search space size, the cost of density estimation for our method is negligible, and the cost of all compared models is similar. We report rank correlation with ground truth using Kendall's Tau. In all cases, our method shows a clear improvement. To obtain the final architectures, we apply Boltzmann Softmax Exploration (BSE) [29] to the trained supernet as the search method. Using BSE, only our method is able to output the best architecture in the search space as the final result for both benchmarks.

In Fig. 3, we compare the accuracy of architectures when trained independently (ground truth) with the accuracy obtained for supernet training with uniform and debiased sampling. Uniform sampling shows bias in the evaluation of high-density architectures, which is not consistent with the ground truth. Debiased sampling mitigates this bias, improves overall accuracy in low-density regions, and crucially increases the ranking of top-performing architectures.

4.1.3. Ablations

Alternative representations for density estimation We proposed to use functional (output) density as an alternative to gradient density. In Tab. 2, we compare different architecture representations and distance metric $d(\cdot, \cdot)$ for density estimation in Eq. (5).

We directly use the gradients for the Pooling benchmark. For NAS-Bench-Macro, in which weights are not fully shared, we consider gradients as orthogonal $d(i, j) = 1$ when layers i and j do not share weights. From Tab. 2, we observe that gradient density and functional density perform similarly.

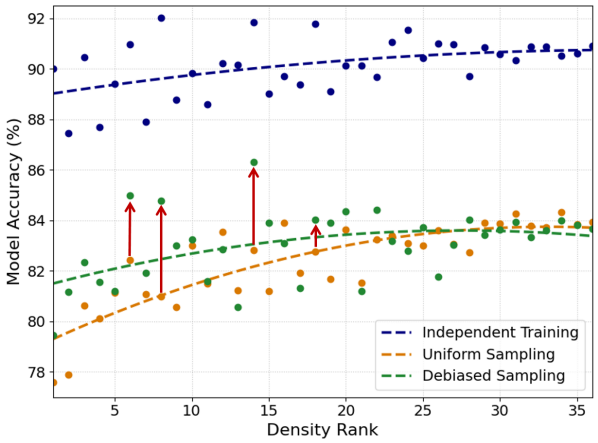

Figure 3. **NAS accuracy on Pooling benchmark** for models trained independently (our gold standard, in blue), a uniformly trained supernet (orange), and our debiased training (green). Models are sorted from low to high density. While independently trained models have low correlation with the density, the model trained with uniform sampling shows higher correlation, where high-density samples are evaluated as performing better, as they have received more effective training iterations. In contrast, our approach removes that density bias and shows a more similar correlation to the independent training, which we aim to approximate. Debiased training shows improvement, especially in assessing top-ranking architectures (red arrows).

We also explore density estimation from a representational similarity/distance measure. Representational similarity is complementary to functional similarity and compares the pattern of activations in hidden layers of the feature maps for a given input [12]. Since comparing large feature maps for all layers is costly, we only apply Centered Kernel Alignment (CKA) [13] to a selected number of layers to estimate density. Functional distances achieve great performance with a lower cost than both. We also compare with the cheapest representations, the encoding using Hamming distances. For more details about these representations/distances, see the appendix.

Density during training In Fig. 4, we show density variation during supernet training. At initialization, shared weights are initialized randomly from the same distribution, resulting in indistinguishable architectures. After only a few epochs of training, density estimations mostly stabilize, meaning that a short warm-up for density estimation is feasible.

Debiasing temperature Temperature (τ) in Eq. (7) controls the strength of debiasing with $\tau \to \infty$ corresponding to uniform sampling. In Fig. 5, we compare various tempera-

Table 2. CIFAR10 results on (top) Pooling search space [23] (bottom) NAS-Bench-Macro [28]. We compare using average gradients of layers, feature maps, and encodings to estimate density. We compare the average accuracy of top-k architectures and the final architecture accuracy found by Boltzmann search.

Debiasing Method	Top-1	Top-5	BSE
Pooling Benchmark			
Gradient	91.54	91.37 ± 0.21	91.77
Representational	90.96	90.81 ± 0.31	91.04
Encoding	90.52	90.61 ± 0.27	91.01
Functional (ours)	91.83	91.59 ± 0.24	92.01
NAS-Bench-Macro			
Gradient	92.68	92.58 ± 0.11	93.05
Representational	92.39	92.32 ± 0.09	92.90
Encoding	91.95	90.59 ± 0.05	92.02
Functional (ours)	92.56	92.21 ± 0.16	93.13

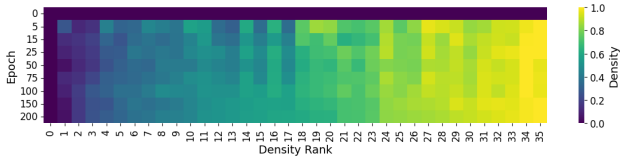

Figure 4. **Density variations during training.** Density estimations during supernet training stabilize after a few training epochs.

tures with uniform sampling. Very low τ enforces debiasing strongly and under-trains high-density architectures significantly, which introduces instability and lowers the overall performance of the supernet.

4.2. Experiments on Large Search Spaces

To validate our method for large search spaces (Algorithm 2), we performed experiments on NAS-Bench-201 and the MobileNet search space on ImageNet.

NAS-Bench-201 experiments NAS-Bench-201 [5] is a tabular cell-based benchmark based on a DARTS-like [16] search space. It searches for 4 nodes with 5 possible operations in a cell, totaling $15,625$ architectures. In Tab. 3 we report top-1 accuracy and Kendall's Tau for CIFAR10 and CIFAR100 datasets. Our method shows improvement in terms of both accuracy and rank correlation.

Number of Prototypes To initialize prototypes, we need to apply K-Means clustering to a population of N samples. We note that uniformly sampling architectures draws more samples from dense regions of the search space. Sparse regions are then less likely to be assigned their own prototypes by the subsequent clustering. While the prototype set size

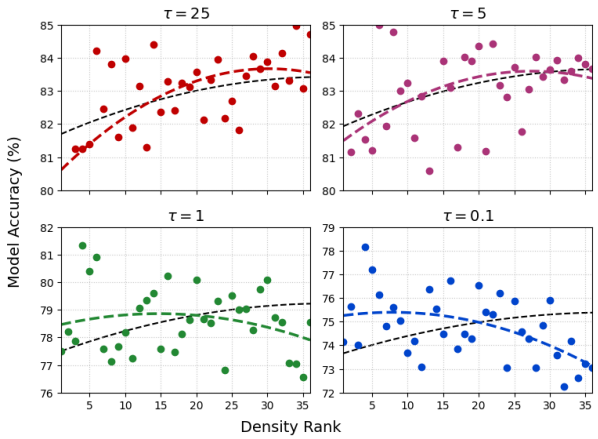

Figure 5. **Supernet evaluation with different debiasing temperatures.** Independent training (ground truth) slope is shown with a black line as a reference. At a high temperature $\tau = 25$, the supernet behaves similarly to uniform sampling. Decreasing the temperature to $\tau = 5$ improves the correlation with ground truth. When $\tau < 1$, density-aware debiasing is over-enforced, resulting in a strong bias towards lower density architectures.

K can be very large for distance calculation in Eq. (11), it is effectively limited by computational budget to calculate N. To measure how density estimations using prototypes model the fine-grained accuracy, we report the correlation of density estimation using K prototypes with fine-grained estimation in Tab. 4.

MobileNet Search Space The search space consists of 21 layers of mobile inverted bottleneck convolutions (MBConv) with kernel sizes $(3, 5, 7)$ and expansion ratio of $(3, 6)$ and choice of skip connection, resulting in a size of 12^{21}. To accelerate training on ImageNet, we utilize mixed-precision and FFCV [15] library.

For comparison with supernet sampling methods during training (stage 1), we consider the following sampling methods: **1)** default uniform sampling (original SPOS), **2)** FairNas (sampling with fairness), **3)** PA&DA (sampling probability adjustments based on gradient norm of path and data).

To find the final architecture (stage 2), we consider the following search methods:

- **Random search:** Not using supernet. Randomly select ten architectures as a baseline and report the best.
- **One-shot:** Use random search on trained supernet and select the best architecture based on supernet evaluations from 50 candidates.
- **EA with shifting:** Use evolutionary algorithm with supernet shifting [37] for architecture search on the trained supernet.

In Tab. 5, we show that under the same setting, training the supernet with density-aware debiasing increases the over-

Table 3. CIFAR10, CIFAR100 and ImageNet16-120 results for NAS-Bench-201 [5]. We compare Top-1 and Kendall's Tau for various sampling methods.

Training Method	CIFAR10		CIFAR100		ImageNet16-120		GPU (hours)
	Top-1 Acc.	Kendall's τ	Top-1 Acc.	Kendall's τ	Top-1 Acc.	Kendall's τ	
Uniform	84.02 ± 2.31	0.42	52.31 ± 3.49	0.39	28.14 ± 6.15	0.49	2.59
FairNas	87.33 ± 0.89	0.53	54.07 ± 2.45	0.49	37.03 ± 1.96	0.51	2.64
Debiased (ours)	91.63 ± 1.21	0.59	66.48 ± 2.08	0.58	42.71 ± 0.79	0.63	2.70
Best	94.37	-	73.51	-	47.31	-	-

Table 4. Top-1 accuracy and rank correlation for NAS-Bench-201 for various number of prototypes (K). We report the Pearson correlation coefficient of estimated density from prototypes with fine-grained density estimation.

K	Top-1 Acc.	Pearson
5	54.12 ± 5.16	0.21
50	86.71 ± 4.36	0.42
100	91.63 ± 1.21	0.86
200	91.52 ± 1.24	0.88

Table 5. ImageNet results on MobileNet search space. We compare supernet training debiased sampling with 3 other sampling methods. We apply one-shot and EA with shifting on the trained supernet as a search algorithm to obtain the final results.

Training Method	Top-1	Top-5	FLOPs (M)	Params (M)
MobileNetV2	72.0	91.0	300	3.4
Random search	70.27	88.1	300	4.1
One-shot				
Uniform	73.08	91.8	312	3.2
FairNas	73.14	91.9	320	4.0
PA&DA	73.41	92.0	340	4.8
Debiased (ours)	74.75	92.2	282	4.2
EA with shifting				
Uniform	73.51	92.0	336	4.8
FairNas	73.65	92.0	324	3.4
PA&DA	73.84	92.1	388	4.9
Debiased (ours)	75.10	92.5	332	4.1

all performance of the found architectures with both search methods.

5. Conclusion

In this work, we have addressed the inherent training bias in one-shot NAS, where commonly used methods (*e.g.* uniform sampling) over-train architectures in dense regions of the representation space, while under-training those in sparse ones, compromising the supernet training process. We have introduced a flexible density-aware sampling framework that reduces this bias by sampling architectures inversely proportional to their estimated density. With practical algorithms for both numerable and large-scale search spaces, our method consistently improves supernet training across diverse benchmarks, leading to the discovery of superior final architectures. Furthermore, our approach can be easily combined with various search methods such as evolutionary algorithms. This work demonstrates the critical importance of correcting for sampling bias and provides a robust, general-purpose alternative to sampling methods such as uniform for NAS pipelines.

6. Acknowledgment

This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Fonds de recherche du Québec – Nature et technologies. We also thank the Digital Research Alliance of Canada (alliancecan.ca) for the use of their computing resources.

References

[1] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 2

[2] Minghao Chen, Jianlong Fu, and Haibin Ling. One-shot neural ensemble architecture search by diversity-guided search space shrinking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16530–16539, 2021. 2

[3] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on computer vision*, pages 12239–12248, 2021. 2

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[5] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 5, 7, 8

[6] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-

shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020. 1, 2, 3

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[8] Shoukang Hu, Ruochen Wang, Lanqing Hong, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot nas with gradient matching. *arXiv preprint arXiv:2203.15207*, 2022. 2, 3

[9] Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. In *European conference on computer vision*, pages 119–134. Springer, 2020. 2

[10] Tao Huang, Shan You, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Greedynasv2: Greedier search with a greedy path filter. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11902–11911, 2022. 3

[11] Jeimin Jeon, Youngmin Oh, Junghyup Lee, Donghyeon Baek, Dohyung Kim, Chanho Eom, and Bumsub Ham. Subnet-aware dynamic supernet training for neural architecture search. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 30137–30146, 2025. 3

[12] Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *ACM Computing Surveys*, 57(9):1–52, 2025. 5, 6

[13] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMlR, 2019. 6

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[15] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. Ffcv: Accelerating training by removing data bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12011–12020, 2023. 7

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2, 7

[17] Shun Lu, Yu Hu, Longxing Yang, Zihao Sun, Jilin Mei, Jianchao Tan, and Chengru Song. Pa&da: Jointly sampling path and data for consistent nas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11940–11949, 2023. 2, 3

[18] Timotée Ly-Manson, Mathieu Leonardon, Abdeldjalil Aissa El Bey, Ghouti Boukli Hacene, and Lukas Mauch. Analyzing few-shot neural architecture search in a metric-driven framework. In *International Conference on Automated Machine Learning AutoML24*, pages 1–33, 2024. 3

[19] Lianbo Ma, Yuee Zhou, Ye Ma, Guo Yu, Qing Li, Qiang He, and Yan Pei. Defying multi-model forgetting in one-shot neural architecture search using orthogonal gradient learning. *IEEE Transactions on Computers*, 2025. 3

[20] Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. Evaluating efficient performance estimators of neural architectures. *Advances in Neural Information Processing Systems*, 34:12265–12277, 2021. 1, 3

[21] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018. 1, 2

[22] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, pages 4780–4789, 2019. 2

[23] Mehraveh Javan Roshtkhari, Matthew Toews, and Marco Pedersoli. Balanced mixture of supernets for learning the cnn pooling architecture. In *International Conference on Automated Machine Learning*, pages 8–1. PMLR, 2023. 1, 2, 5, 6, 7

[24] Mehraveh Javan Roshtkhari, Matthew Toews, and Marco Pedersoli. Iterative monte carlo tree search for neural architecture search. In *International Conference on Automated Machine Learning*, pages 3–1. PMLR, 2025. 2

[25] Mehraveh Javan Roshtkhari, Matthew Toews, and Marco Pedersoli. Neural architecture search by learning a hierarchical search space. *arXiv preprint arXiv:2503.21061*, 2025. 2

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 5

[27] Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. *arXiv preprint arXiv:1909.09569*, 2019. 2

[28] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10968–10977, 2021. 2, 5, 6, 7

[29] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998. 6

[30] Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9): 5503–5515, 2021. 2

[31] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10734–10742, 2019. 1

[32] Xin Xia, Xuefeng Xiao, Xing Wang, and Min Zheng. Progressive automatic design of search space for one-shot neural architecture search. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2455–2464, 2022. 2

[33] Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. Analyzing and mitigating interference in neural architecture search. In *International Conference on Machine Learning*, pages 24646–24662. PMLR, 2022. 1

[34] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020. 3

[35] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019. 2

[36] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019. 2

[37] Beichen Zhang, Xiaoxing Wang, Xiaohan Qin, and Junchi Yan. Boosting order-preserving and transferability for neural architecture search: a joint architecture refined search and fine-tuning approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5662–5671, 2024. 2, 7

[38] Mingyang Zhang, Xinyi Yu, Haodong Zhao, and Linlin Ou. Shiftnas: Improving one-shot nas via probability shift. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5919–5928, 2023. 3

[39] Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. *arXiv preprint arXiv:2001.01431*, 2020. 1

[40] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR, 2021. 2

[41] Yiyang Zhao, Linnan Wang, Kevin Yang, Tianjun Zhang, Tian Guo, and Yuandong Tian. Multi-objective optimization by learning space partitions. *arXiv preprint arXiv:2110.03173*, 2021. 2

[42] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 2