Neural Architecture Search without Training

Joseph Mellor [1] **Jack Turner** [2] **Amos Storkey** [2] **Elliot J. Crowley** [3]

Abstract

The time and effort involved in hand-designing deep neural networks is immense. This has prompted the development of Neural Architecture Search (NAS) techniques to automate this design. However, NAS algorithms tend to be slow and expensive; they need to train vast numbers of candidate networks to inform the search process. This could be alleviated if we could partially predict a network's trained accuracy from its initial state. In this work, we examine the overlap of activations between datapoints in *untrained* networks and motivate how this can give a measure which is usefully indicative of a network's *trained* performance. We incorporate this measure into a simple algorithm that allows us to search for powerful networks without any training in a matter of seconds on a single GPU, and verify its effectiveness on NAS-Bench-101, NAS-Bench-201, NATS-Bench, and Network Design Spaces. Our approach can be readily combined with more expensive search methods; we examine a simple adaptation of regularised evolutionary search. Code for reproducing our experiments is available at `https://github.com/BayesWatch/nas-without-training`.

1. Introduction

The success of deep learning in computer vision is in no small part due to the insight and engineering efforts of human experts, allowing for the creation of powerful architectures for widespread adoption (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; He et al., 2016; Szegedy et al., 2016; Huang et al., 2017). However, this manual design is costly, and becomes increasingly more difficult as networks get larger and more complicated. Because of these challenges, the neural network community has seen a shift from designing architectures to designing algorithms that *search* for candidate architectures (Elsken et al., 2019; Wistuba et al., 2019). These Neural Architecture Search (NAS) algorithms are capable of automating the discovery of effective architectures (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018; Tan et al., 2019; Liu et al., 2019; Real et al., 2019).

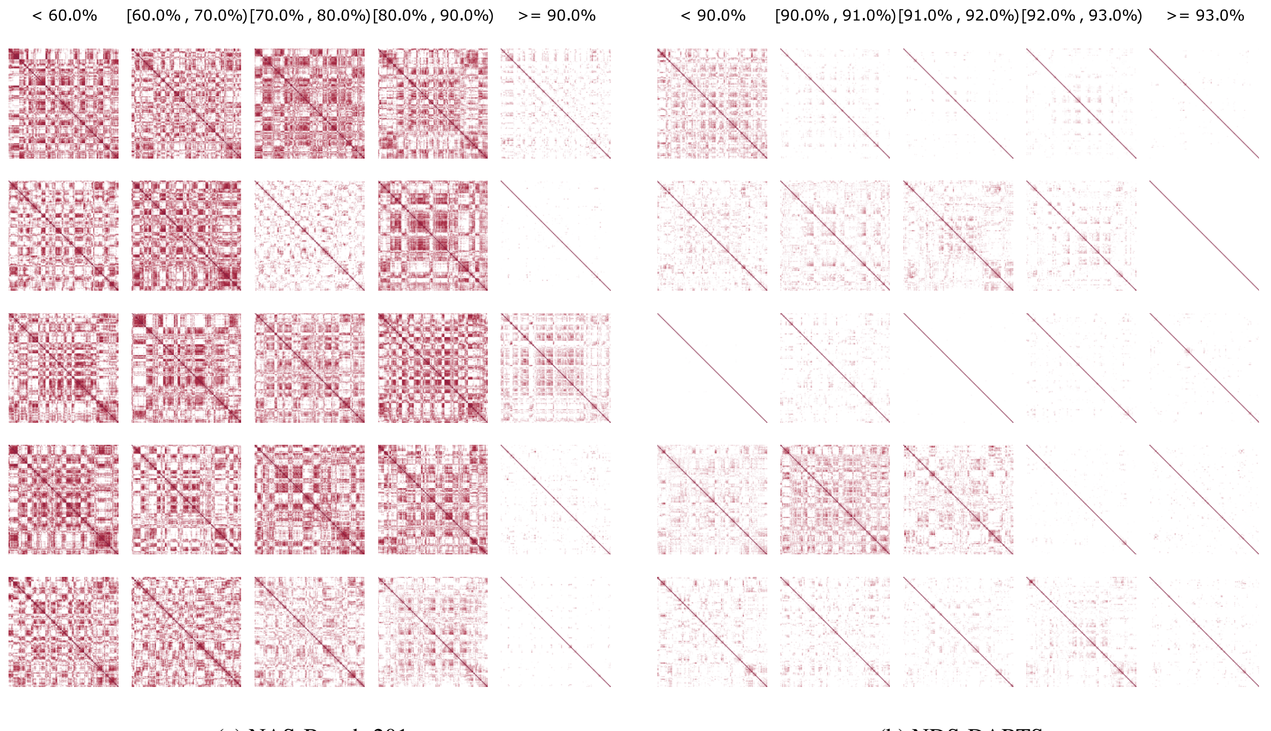
NAS algorithms are broadly based on the seminal work of Zoph & Le (2017). A controller network generates an architecture proposal, which is then trained to provide a signal to the controller through REINFORCE (Williams, 1992), which then produces a new proposal, and so on. Training a network for every controller update is extremely expensive; utilising 800 GPUs for 28 days in Zoph & Le (2017). Subsequent work has sought to ameliorate this by (i) learning stackable cells instead of whole networks (Zoph et al., 2018) and (ii) incorporating *weight sharing*; allowing candidate networks to share weights to allow for joint training (Pham et al., 2018). These contributions have accelerated the speed of NAS algorithms e.g. to half a day on a single GPU in Pham et al. (2018).

For some practitioners, NAS is still too slow; being able to perform NAS quickly (i.e. in seconds) would be immensely useful in the hardware-aware setting where a separate search is typically required for each device and task (Wu et al., 2019; Tan et al., 2019). This could be achieved if NAS could be performed *without any network training*. In this paper we show that this is possible. We explore NAS-Bench-101 (Ying et al., 2019), NAS-Bench-201 (Dong & Yang, 2020), NATS-Bench (Dong et al., 2021), and Network Design Spaces (NDS, Radosavovic et al., 2019), and examine the overlap of activations between datapoints in a mini-batch for an untrained network (Section 3). The linear maps of the network are uniquely identified by a binary code corresponding to the activation pattern of the rectified linear units. The Hamming distance between these binary codes can be used to define a kernel matrix (which we denote by $\mathbf{K}_H$) which is distinctive for networks that perform well; this is immediately apparent from visualisation alone across two distinct search spaces (Figure 1). We devise a score based on $\mathbf{K}_H$ and perform an ablation study to demonstrate its robustness to inputs and network initialisation.

We incorporate our score into a simple search algorithm

[1]Usher Institute, University of Edinburgh [2]School of Informatics, University of Edinburgh [3]School of Engineering, University of Edinburgh. Correspondence to: Joseph Mellor <joe.mellor@ed.ac.uk>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).



(a) NAS-Bench-201 (b) NDS-DARTS

Figure 1. $\mathbf{K}_H$ for a mini-batch of 128 CIFAR-10 images for untrained architectures in (a) NAS-Bench-201 (Dong & Yang, 2020) and (b) NDS-DARTS (Radosavovic et al., 2019). $\mathbf{K}_H$ in these plots is normalised so that the diagonal entries are 1. The $\mathbf{K}_H$ are sorted into columns based on the final CIFAR-10 validation accuracy when trained. Darker regions have higher similarity. The profiles are distinctive; the $\mathbf{K}_H$ for good architectures in both search spaces have less similarity between different images. We can use $\mathbf{K}_H$ for an untrained network to predict its final performance without any training. Note that (b) covers a tighter accuracy range than (a), which may explain it being less distinctive.

that doesn't require training (Section 4). This allows us to perform architecture search quickly, for example, on CIFAR-10 (Krizhevsky, 2009) we are able to search for a network that achieves 92.81% accuracy in 30 seconds within the NAS-Bench-201 search space; several orders of magnitude faster than traditional NAS methods for a modest change in final accuracy. We also show how we can combine our approach with regularised evolutionary search (REA, Pham et al., 2018) as an example of how it can be readily integrated into existing NAS techniques.

We believe that this work is an important proof-of-concept for NAS without training. The large resource and time costs associated with NAS can be avoided; our search algorithm uses a single GPU and is extremely fast. The benefit is two-fold, as we also show that we can integrate our approach into existing NAS techniques for scenarios where obtaining as high an accuracy as possible is of the essence.

2. Background

Designing a neural architecture by hand is a challenging and time-consuming task. It is extremely difficult to intuit where to place connections, or which operations to use. This has prompted an abundance of research into neural architecture search (NAS); the automation of the network design process. In the pioneering work of Zoph & Le (2017), the authors use an RNN controller to generate descriptions of candidate networks. Candidate networks are trained, and used to update the controller using reinforcement learning to improve the quality of the candidates it generates. This algorithm is very expensive: searching for an architecture to classify CIFAR-10 required running 800 GPUs for 28 days. It is also inflexible; the final network obtained is fixed and cannot be scaled e.g. for use on mobile devices or for other datasets.

The subsequent work of Zoph et al. (2018) deals with these limitations. Inspired by the modular nature of successful hand-designed networks (Simonyan & Zisserman, 2015; He et al., 2016; Huang et al., 2017), they propose searching over neural building blocks, instead of over whole architectures. These building blocks, or *cells*, form part of a fixed overall network structure. Specifically, the authors search for a standard cell, and a reduced cell (incorporating pooling) for CIFAR-10 classification. These are then used as the building

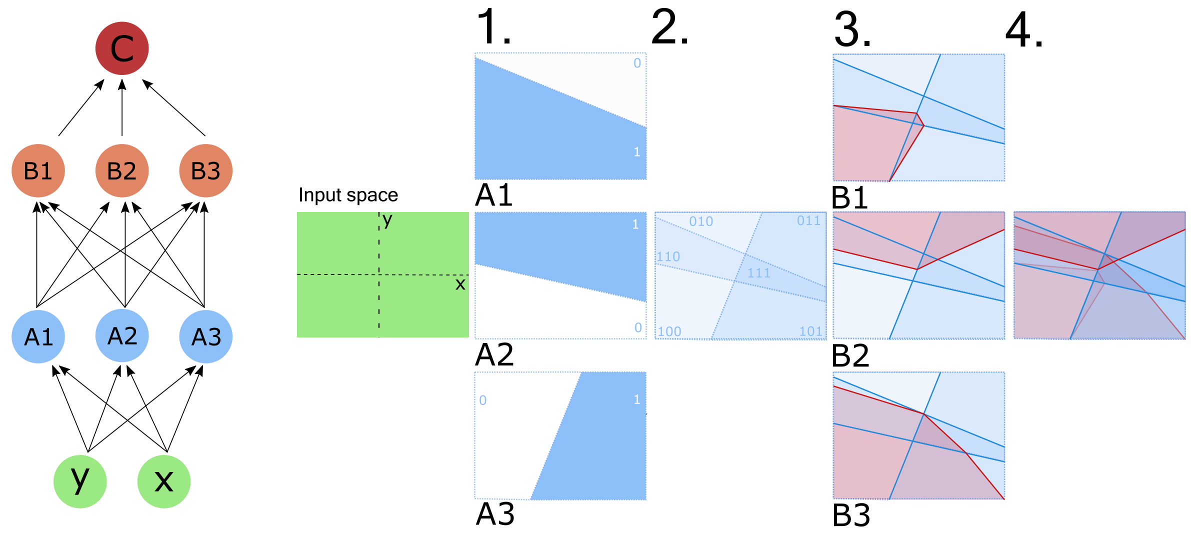


Figure 2. Visualising how binary activation codes of ReLU units correspond to linear regions. **1:** Each ReLU node Ai splits the input into an active (> 0) and inactive region We label the active region 1 and inactive 0. **2:** The active/inactive regions associated with each node Ai intersect. Areas of the input space with the same activation pattern are co-linear. Here we show the intersection of the A nodes and give the code for the linear regions. Bit i of the code corresponds to whether node Ai is active. **3:** The ReLU nodes B of the next layer divides the space further into active and inactive regions. **4:** Each linear region at a given node can be uniquely defined by the activation pattern of all the ReLU nodes that preceded it.

blocks of a larger network for ImageNet (Russakovsky et al., 2015) classification. While more flexible—the number of cells can be adjusted according to budget—and cheaper, owing to a smaller search space, this technique still utilised 500 GPUs across 4 days.

ENAS (Pham et al., 2018) reduces the computational cost of searching by allowing multiple candidate architectures to share weights. This facilitates the simultaneous training of candidates, reducing the search time on CIFAR-10 to half a day on a single GPU. Weight sharing has seen widespread adoption in a host of NAS algorithms (Liu et al., 2019; Luo et al., 2018; Cai et al., 2019; Xie et al., 2019; Brock et al., 2018). However, there is evidence that it inhibits the search for optimal architectures (Yu et al., 2020), exposing random search as an extremely effective NAS baseline (Yu et al., 2020; Li & Talwalkar, 2019). There also remains the problem that the search spaces are still so vast—there are 1.6×10^{29} possible architectures in Pham et al. (2018) for example—that it is impossible to identify the best networks and demonstrate that NAS algorithms find them.

An orthogonal direction for identifying good architectures is the estimation of accuracy prior to training (Deng et al., 2017; Istrate et al., 2019), although these differ from this work in that they rely on training a predictive model, rather than investigating more fundamental architectural properties. Since its inception others have explored our work and the ideas therein in interesting directions. Of most interest from our perspective are Abdelfattah et al. (2021) who integrate training-free heuristics into existing more-expensive search strategies to improve their performance as we do in this paper. Park et al. (2020) use the correspondence between wide neural networks and Gaussian processes to motivate using as a heuristic the validation accuracy of a Monte-Carlo approximated neural network Gaussian process conditioned on training data. Chen et al. (2021) propose two further heuristics—one based on the condition number of the neural tangent kernel (Jacot et al., 2018) at initialisation and the other based on the number of unique linear regions that partition training data at initialisation—with a proposed strategy to combine these heuristics into a stronger one.

2.1. NAS Benchmarks

A major barrier to evaluating the effectiveness of a NAS algorithm is that the search space (the set of all possible networks) is too large for exhaustive evaluation. This has led to the creation of several benchmarks (Ying et al., 2019; Zela et al., 2020; Dong & Yang, 2020; Dong et al., 2021) that consist of tractable NAS search spaces, and metadata for the training of networks within that search space. Concretely, this means that it is now possible to determine whether an algorithm is able to search for a good network. In this work we utilise NAS-Bench-101 (Ying et al., 2019), NAS-Bench-201 (Dong & Yang, 2020), and NATS-Bench (Dong et al., 2021) to evaluate the effectiveness of our approach. NAS-Bench-101 consists of 423,624 neural networks that have been trained exhaustively, with three different initialisations, on the CIFAR-10 dataset for 108 epochs. NAS-Bench-

201 consists of 15,625 networks trained multiple times on CIFAR-10, CIFAR-100, and ImageNet-16-120 (Chrabaszcz et al., 2017). NATS-Bench (Dong et al., 2021) comprises two search spaces: a topology search space (NATS-Bench TSS) which contains the same 15,625 networks as NAS-Bench 201; and a size search space (NATS-Bench SSS) which contains 32,768 networks where the number of channels for cells varies between these networks. These benchmarks are described in detail in Appendix A.

We also make use of the Network Design Spaces (NDS) dataset (Radosavovic et al., 2019). Where the NAS benchmarks aim to compare search *algorithms*, NDS aims to compare the search *spaces* themselves. All networks in NDS use the DARTS (Liu et al., 2019) skeleton. The networks are comprised of cells sampled from one of several NAS search spaces. Cells are sampled—and the resulting networks are trained—from each of AmoebaNet (Real et al., 2019); DARTS (Liu et al., 2019); ENAS (Pham et al., 2018), NASNet (Zoph & Le, 2017), and PNAS (Liu et al., 2018).

We denote each of these sets as NDS-AmoebaNet, NDS-DARTS, NDS-ENAS, NDS-NASNet, and NDS-PNAS respectively. Note that these sets contain networks of variable width and depth, whereas in e.g. the original DARTS search space these were fixed quantities.[1]

3. Scoring Networks at Initialisation

Our goal is to devise a means to score a network architecture at initialisation in a way that is indicative of its final trained accuracy. This can either replace the expensive inner-loop training step in NAS, or better direct exploration in existing NAS algorithms.

Given a neural network with rectified linear units, we can, at each unit in each layer, identify a binary indicator as to whether the unit is inactive (the value is negative and hence is multiplied by zero) or active (in which case its value is multiplied by one). Fixing these indicator variables, it is well known that the network is now locally defined by a linear operator (Hanin & Rolnick, 2019); this operator is obtained by multiplying the linear maps at each layer interspersed with the binary rectification units. Consider a mini-batch of data $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ mapped through a neural network as $f(\mathbf{x}_i)$. The indicator variables from the rectified linear units in f at $\mathbf{x}_i$ form a binary code $\mathbf{c}_i$ that defines the linear region.

The intuition to our approach is that the more similar the binary codes associated with two inputs are then the more challenging it is for the network to learn to separate these inputs. When two inputs have the same binary code, they lie within the same linear region of the network and so are particularly difficult to disentangle. Conversely, learning should prove easier when inputs are well separated. Figure 2 visualises binary codes corresponding to linear regions.

We use the Hamming distance $d_H(\mathbf{c}_i, \mathbf{c}_j)$ between two binary codes—induced by the untrained network at two inputs—as a measure of how dissimilar the two inputs are.

We can examine the correspondence between binary codes for the whole mini-batch by computing the kernel matrix

$$\mathbf{K}_H = \begin{pmatrix} N_A - d_H(\mathbf{c}_1, \mathbf{c}_1) & \cdots & N_A - d_H(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(\mathbf{c}_N, \mathbf{c}_1) & \cdots & N_A - d_H(\mathbf{c}_N, \mathbf{c}_N) \end{pmatrix} \tag{1}$$

where N_A is the number of rectified linear units.

We compute $\mathbf{K}_H$ for a random subset of NAS-Bench-201 (Dong & Yang, 2020) and NDS-DARTS (Radosavovic et al., 2019) networks **at initialisation** for a mini-batch of CIFAR-10 images. We plot normalised $\mathbf{K}_H$ for different trained accuracy bounds in Figure 1.

These normalised kernel plots are very distinct; high performing networks have fewer off-diagonal elements with high similarity. We can use this observation to predict the final performance of untrained networks, in place of the expensive training step in NAS. Specifically, we score networks using:

$$s = \log |\mathbf{K}_H| \tag{2}$$

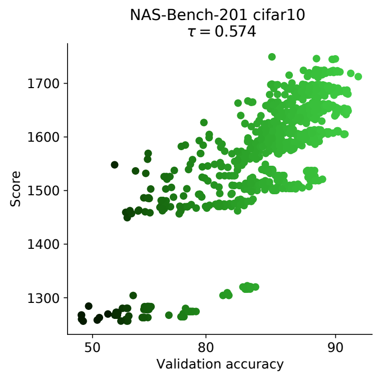
Given two kernels with the same trace, s is higher for the kernel closest to diagonal. A higher score at initialisation implies improved final accuracy after training.

For the search spaces across NAS-Bench-101 (Ying et al., 2019), NAS-Bench-201 (Dong & Yang, 2020), NATS-Bench SSS (Dong et al., 2021), and NDS (Radosavovic et al., 2019) we sample networks at random and plot our score s on the *untrained* networks against their validation accuracies *when trained*. The plots for NAS-Bench-101, NAS-Bench-201, and NDS are available in Figure 3. In most cases 1000 networks are sampled.[2] The plots for NATS-Bench SSS can be found in Figure 9 (Appendix B). We also provide comparison plots for the fixed width and depth spaces in NDS in Figure 10 (Appendix B). Kendall's Tau correlation coefficients τ are given at the top of each plot.

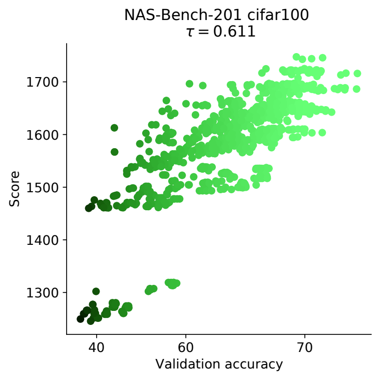
We find in all cases there is a positive correlation between

[1]NDS also contains a set of networks with fixed width and depth for the DARTS, ENAS, and PNAS cell search spaces. We provide experiments on these sets in Appendix B.

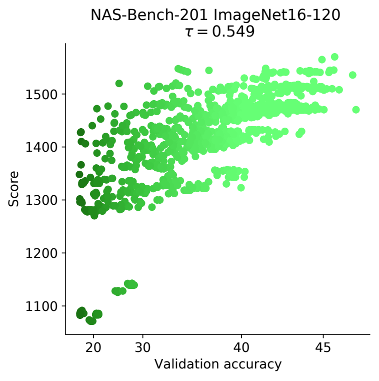
[2]Due to GPU memory limitations, there are 900, 749, and 973 networks shown for NDS-AmoebaNet, NDS-ENAS, and NDS-PNAS respectively.



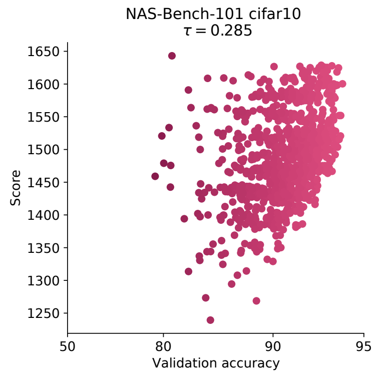
(a)



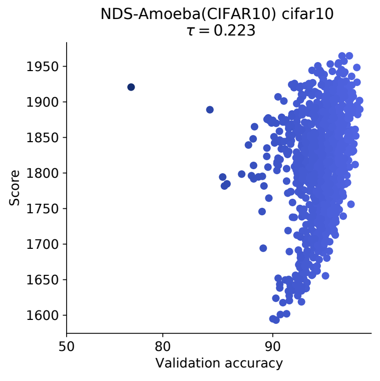
(b)



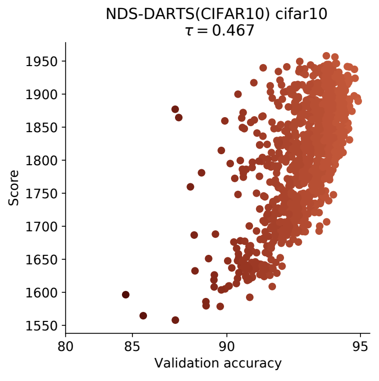
(c)



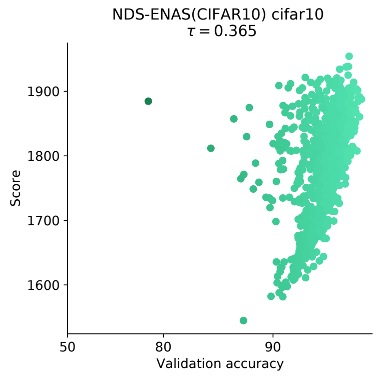
(d)



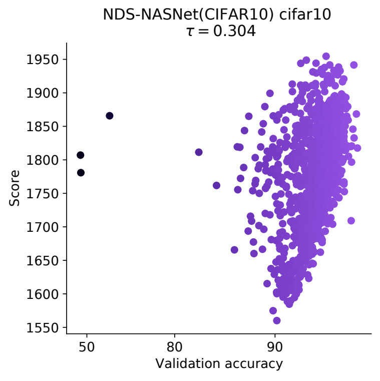
(e)



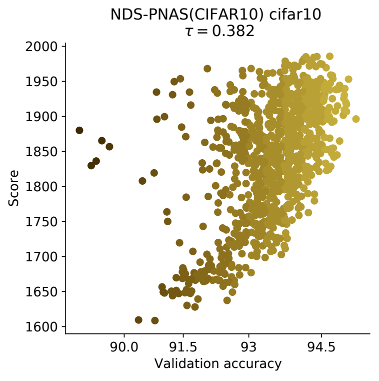
(f)



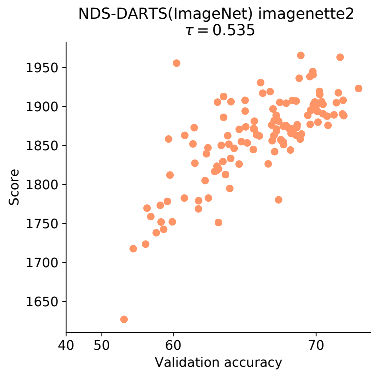
(g)



(h)



(i)



(j)

Figure 3. (a)-(i): Plots of our score for randomly sampled **untrained** architectures in NAS-Bench-201, NAS-Bench-101, NDS-Amoeba, NDS-DARTS, NDS-ENAS, NDS-NASNet, NDS-PNAS against validation accuracy when trained. The inputs when computing the score and the validation accuracy for each plot are from CIFAR-10 except for (b) and (c) which use CIFAR-100 and ImageNet16-120 respectively. (j): We include a plot from NDS-DARTS on ImageNet (121 networks provided) to illustrate that the score extends to more challenging datasets. We use a mini-batch from ImageNette2 which is a strict subset of ImageNet with only 10 classes. In all cases there is a noticeable correlation between the score for an untrained network and the final accuracy when trained.

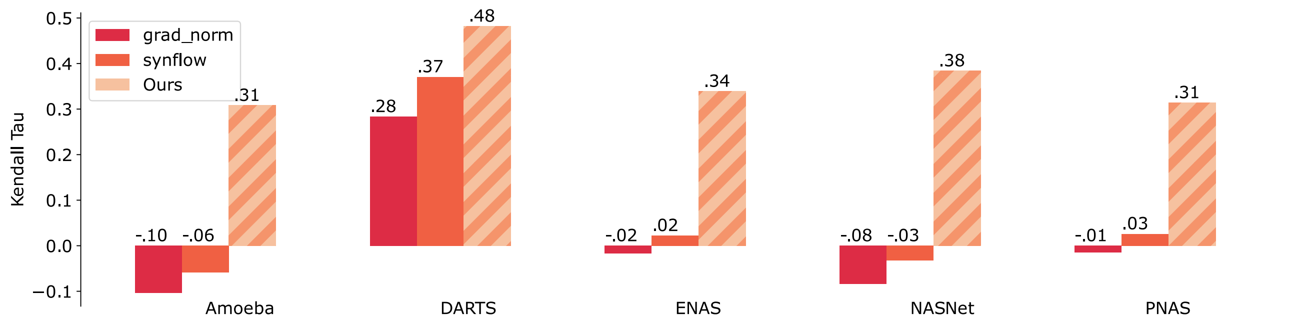


Figure 4. Kendall's Tau correlation across each of the NDS CIFAR-10 search spaces. We compare our method to two alternative measures: `grad_norm` and `synflow`. The results for `grad_norm` refer to the absolute Euclidean-norm of the gradients over one random mini-batch of data. `synflow` is the gradient-based score defined by Tanaka et al. (2020), summed over each parameter in the network.

the validation accuracy and the score. This is particularly strong for NAS-Bench-201 and NDS-DARTS. We show the Kendall's Tau correlation coefficient between s and final accuracy on CIFAR-10 for NDS in Figure 4. For comparison, we include the best-performing architecture scoring functions from Abdelfattah et al. (2021) — `grad_norm` and `synflow` — as baselines. The first is the sum of the gradient norms for every weight in the network, and the second is the summed Synaptic Flow score derived in Tanaka et al. (2020). our score (Equation 2) correlates with accuracy across all of the search spaces, where the other two scores fluctuate substantially. These results point to our score being effective on a wide array of neural network design spaces.

3.1. Ablation Study

How important are the images used to compute the score? Since our approach relies on randomly sampling a single mini-batch of data, it is reasonable to question whether different mini-batches result in different scores. To determine whether our method is dependent on mini-batches, we randomly select 10 architectures from different CIFAR-100 accuracy percentiles in NAS-Bench-201 and compute the score separately for 20 random CIFAR-100 mini-batches. The resulting box-and-whisker plot is given in Figure 5(top-left): the ranking of the scores is reasonably robust to the specific choice of images. In Figure 5(top-right) we compute our score using normally distributed random inputs; this has little impact on the general trend. This suggests our score captures a property of the network architecture, rather than something data-specific.

Does the score change for different initialisations? Figure 5(bottom-left) shows how the score for our 10 NAS-Bench-201 architectures differs over 20 initialisations. While there is some noise, the better performing networks remain distinctive, and can be isolated.

Does the size of the mini-batch matter? As $\mathbf{K}_H$ scales

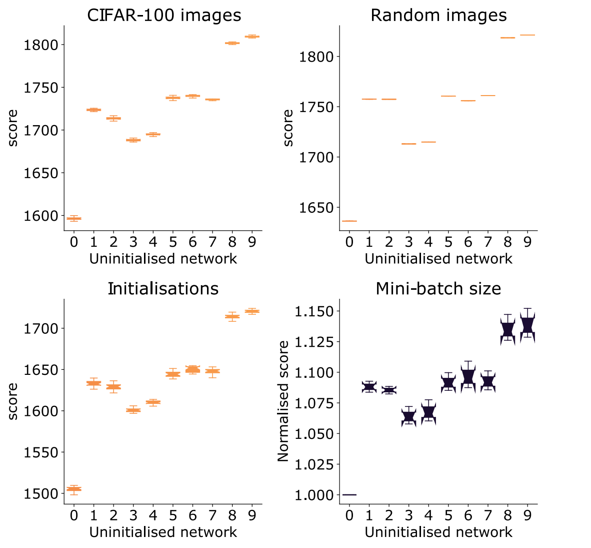


Figure 5. Ablation experiments showing the effect on our score using different CIFAR-100 mini-batches (top-left), random normally-distributed input images (top-right), weight initialisations (bottom-left), and mini-batch sizes (bottom-right) for 10 randomly selected NAS-Bench-201 architectures (one in each 5% percentile range from 50-55, ..., 95-100). For each network, 20 samples were taken for each ablation. The mini-batch size was 128 for all experiments apart from the bottom-right. The bottom-right experiment used mini-batch sizes of 32, 64, 128, and 256; as the score depends on the mini-batch size we normalised the score by the minimum score of the sampled networks from the same mini-batch size.

with mini-batch size we compare across mini-batch sizes by dividing a given score by the minimum score using the same mini-batch size from the set of sampled networks. Figure 5(bottom-right) presents this normalised score for different mini-batch sizes. The best performing networks remain distinct.

How does the score evolve as networks are trained? Although the motivation of our work is to score uninitialised networks, it is worth observing how the score evolves as a network is trained. We consider 10 NAS-Bench-201 networks with $> 90\%$ validation accuracy when evaluated on CIFAR-10. This makes the performance difference between the 10 networks much smaller than for the search space as a whole. We train each network via stochastic gradient descent with a cross entropy loss for 100 epochs and evaluate the score at each training epoch. Figure 6 shows the evolution of the score. The left subplot shows a zoomed-in view of the score trajectories across the first two epochs and the right subplot shows the score trajectories across all 100 epochs. We observe that the score increases in all cases immediately after some training has occurred, but very quickly stabilises to a near constant value. The increase in the score value after initialisation was similar amongst the networks, and the relative ranking remained similar throughout.

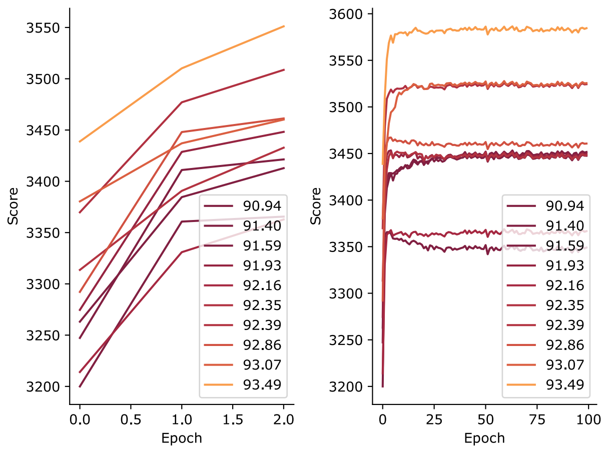


Figure 6. Plots of our score (Equation 2) during training for 10 networks from NAS-Bench-201 using the CIFAR-10 dataset. The legend provides the final accuracy of the network as given by the NAS-Bench-201 API. For all 10 networks the score increases sharply in the first few epochs and then flattens. The ranking of the scores between networks remains relatively stable throughout training.

In Section 4 we demonstrate how our score (Equation 2) can be used in a NAS algorithm for extremely fast search.

4. Neural Architecture Search without Training — NASWOT

In Section 3 we derived a score for cheaply ranking networks at initialisation based on their expected performance (Equation 2). Here as a proof of concept, we integrate this score into a simple search algorithm and evaluate its ability to alleviate the need for training in NAS. Code for reproducing our experiments is available at `https://github.com/BayesWatch/nas-without-training`.

Table 1. Mean ± std. accuracy from NAS-Bench-101. NASWOT is our training-free algorithm (across 500 runs). REA uses evolutionary search to select an architecture (50 runs), Random selects one architecture (500 runs). AREA (assisted-REA) uses our score (Equation 2) to select the starting population for REA (50 runs). Search times for REA and AREA were calculated using the NAS-Bench-101 API.

Method	Search (s)	CIFAR-10
Random	N/A	90.38±5.51
NASWOT (N=100)	23	91.77±0.05
REA	12000	93.87±0.22
AREA (Ours)	12000	93.91±0.29

Algorithm 1 NASWOT
```
generator = RandomGenerator()
best_net, best_score = None, 0
for i=1:N do
    net = generator.generate()
    score = net.score()
    if score > best_score then
        best_net, best_score = net, score
chosen_net = best_network
```

Many NAS algorithms are based on that of Zoph & Le (2017); it uses a generator network which proposes architectures. The weights of the generator are learnt by training the networks it generates, either on a proxy task or on the dataset itself, and using their trained accuracies as signal through e.g. REINFORCE (Williams, 1992). This is repeated until the generator is trained; it then produces a final network which is the output of this algorithm. The vast majority of the cost is incurred by having to train candidate architectures for every single controller update. Note that there exist alternative schema utilising e.g. evolutionary algorithms (Real et al., 2019) or bilevel optimisation (Liu et al., 2019) but all involve training.

We instead propose a simple alternative—NASWOT—illustrated in Algorithm 1. Instead of having a neural network as a generator, we randomly propose a candidate from the search space and then rather than training it, we score it in its untrained state using Equation 2. We do this N times—i.e. we have a sample size of N architectures—and then output the highest scoring network.

NAS-Bench-101. We compare NASWOT to 12000 seconds of REA (Real et al., 2019) and random selection on NAS-Bench-101 (Ying et al., 2019) in Table 1. NASWOT can find a network with a final accuracy roughly midway between these methods in under a minute on a single GPU.

NAS-Bench-201. Dong & Yang (2020) benchmark a wide range of NAS algorithms, both with and without weight sharing, that we compare to NASWOT. The weight shar-

Table 2. **(a):** Mean ± std. accuracies on NAS-Bench-201. Baselines are taken directly from Dong & Yang (2020), averaged over 500 runs (3 for weight-sharing methods). Search times are recorded for a single 1080Ti GPU. Note that all searches are performed on CIFAR-10 before evaluating the final model on CIFAR-10, CIFAR-100, and ImageNet-16-120. The performance of our training-free approach (NASWOT) is given for different sample size N (also 500 runs), along with that of our Assisted REA (AREA) approach (50 runs). We also report the results for picking a network at random, and the best possible network from the sample. **(b):** Mean ± std. accuracies (over 500 runs) on NATS-Bench SSS (Dong et al., 2021) comparing non-weight sharing approaches to NASWOT. Unlike NAS-Bench-201, each search is performed on the same dataset that is then used to evaluate the proposed network.

Method	Search (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
				(a) NAS-Bench-201			
				Non-weight sharing			
REA	12000	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
RS	12000	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE	12000	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB	12000	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
				Weight sharing			
RSPS	7587	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS-V1	10890	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2	29902	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS	28926	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.90
SETN	31010	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90±4.07
ENAS	13315	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
				Training-free			
NASWOT (N=10)	3.05	89.14 ± 1.14	92.44 ± 1.13	68.50 ± 2.03	68.62 ± 2.04	41.09 ± 3.97	41.31 ± 4.11
NASWOT (N=100)	30.01	89.55 ± 0.89	92.81 ± 0.99	69.35 ± 1.70	69.48 ± 1.70	42.81 ± 3.05	43.10 ± 3.16
NASWOT (N=1000)	306.19	89.69 ± 0.73	92.96 ± 0.81	69.86 ± 1.21	69.98 ± 1.22	43.95 ± 2.05	44.44 ± 2.10
Random	N/A	83.20 ± 13.28	86.61 ± 13.46	60.70 ± 12.55	60.83 ± 12.58	33.34 ± 9.39	33.13 ± 9.66
Optimal (N=10)	N/A	89.92 ± 0.75	93.06 ± 0.59	69.61 ± 1.21	69.76 ± 1.25	43.11 ± 1.85	43.30 ± 1.87
Optimal (N=100)	N/A	91.05 ± 0.28	93.84 ± 0.23	71.45 ± 0.79	71.56 ± 0.78	45.37 ± 0.61	45.67 ± 0.64
AREA	12000	91.20 ± 0.27	-	71.95 ± 0.99	-	45.70 ± 1.05	-
				(b) NATS-Bench SSS			
				Non-weight sharing			
REA	12000	90.37±0.20	93.22±0.16	70.23±0.50	70.11±0.61	45.30±0.69	45.54±0.92
RS	12000	90.10±0.26	93.03±0.25	69.57±0.57	69.72±0.61	45.01±0.74	45.42±0.86
REINFORCE	12000	90.25±0.23	93.16±0.21	69.84±0.59	69.96±0.57	45.06±0.77	45.24±1.18
BOHB	12000	90.07±0.28	93.01±0.24	69.75±0.60	69.90±0.60	45.11±0.69	45.56±0.81
NASWOT (N=10)	3.02	88.95 ± 0.88	88.66 ± 0.90	64.55 ± 4.57	64.54 ± 4.70	40.22 ± 3.73	40.48 ± 3.73
NASWOT (N=100)	32.36	89.68 ± 0.51	89.38 ± 0.54	66.71 ± 3.05	66.68 ± 3.25	42.68 ± 2.58	43.11 ± 2.42
NASWOT (N=1000)	248.23	90.14 ± 0.30	93.10 ± 0.31	68.96 ± 1.54	69.10 ± 1.61	44.57 ± 1.48	45.08 ± 1.55

ing methods are random search with parameter sharing (RSPS, Li & Talwalkar, 2019), first-order DARTS (DARTS-V1, Liu et al., 2019), second order DARTS (DARTS-V2, Liu et al., 2019), GDAS (Dong & Yang, 2019b), SETN (Dong & Yang, 2019a), and ENAS (Pham et al., 2018). The non-weight sharing methods are random search with training (RS), REA (Real et al., 2019), REINFORCE (Williams, 1992), and BOHB (Falkner et al., 2018). For implementation details we refer the reader to Dong & Yang (2020). The hyperparameters in NAS-Bench-201 are fixed — these results may not be invariant to hyperparameter choices, which may explain the low performance of e.g. DARTS.

All searches are performed on CIFAR-10, and the output architecture is then trained and evaluated on each of CIFAR-10, CIFAR-100, and ImageNet-16-120 for different dataset splits. We report results in Table 2(a). Search times are reported for a single GeForce GTX 1080 Ti GPU. As per the NAS-Bench-201 setup, the non-weight sharing methods are given a time budget of 12000 seconds. For NASWOT and the non-weight sharing methods, accuracies are averaged over 500 runs. For weight-sharing methods, accuracies are reported over 3 runs. We report NASWOT for sample sizes of N=10, N=100, and N=1000. NASWOT is able to outperform all of the weight sharing methods while requiring a fraction of the search time.

The non-weight sharing methods do outperform NASWOT, though they also incur a large search time cost. It is encouraging however, that in a matter of seconds, NASWOT is able to find networks with performance close to the best non-weight sharing algorithms, suggesting that network architectures themselves contain almost as much information about final performance at initialisation as after training.

Table 2(a) also shows the effect of sample size (N). We show the accuracy of networks chosen by our method for each N. We list optimal accuracy for each N, and random selection over the whole benchmark, both averaged over 500 runs. We observe that sample size increases NASWOT performance.

NATS-Bench. Dong et al. (2021) expanded on the original NAS-Bench-201 work to include a further search space: NATS-Bench SSS. The same non-weight sharing methods were evaluated as in NAS-Bench-201. Unlike NAS-Bench-201, search and evaluation are performed on the same dataset. Implementation details are available in Dong et al. (2021). We observe that sample size increases NASWOT performance significantly on NATS-Bench SSS, to the point where it is extremely similar to other methods.

A key practical benefit of NASWOT is its rapid execution time. This may be important when repeating NAS several times, for instance for several hardware devices or datasets. This affords us the ability in future to specialise neural architectures for a task and resource environment cheaply, demanding only a few seconds per setup. Figure 7 shows our method in contrast to other NAS methods for NAS-Bench-201, showing the trade-off between final network accuracy and search time.

4.1. Assisted Regularised EA — AREA

Our proposed score can be straightforwardly incorporated into existing NAS algorithms. To demonstrate this we implemented a variant of REA (Real et al., 2019), which we call Assisted-REA (AREA). REA starts with a randomly-selected population (10 in our experiments). AREA instead randomly-samples a larger population (in our experiments we double the randomly-selected population size to 20) and uses our score (Equation 2) to select the initial population (of size 10) for the REA algorithm. Pseudocode can be found in Algorithm 2 with results on NAS-Bench-101 and NAS-Bench-201 in Tables 1 and 2. AREA outerforms REA on NAS-Bench-201 (CIFAR-100, ImageNet-16-120) but is very similar to REA on NAS-Bench-101. We hope that future work will build on this algorithm further.

5. Conclusion

NAS has previously suffered from intractable search spaces and heavy search costs. Recent advances in producing tractable search spaces, through NAS benchmarks, have allowed us to investigate if such search costs can be avoided. In this work, we have shown that it is possible to navigate these spaces with a search algorithm—NASWOT—in a matter of seconds, relying on simple, intuitive observations made on initialised neural networks, that challenges more expensive black box methods involving training. Future applications of this approach to architecture search may allow us to use NAS to specialise architectures over multiple tasks and devices without the need for long training stages. We also demonstrate how our approach can be combined into an existing NAS algorithm. This work is not without its limitations; our scope is restricted to convolutional architectures for image classification. However, we hope that this will be a powerful first step towards removing training from NAS and making architecture search cheaper, and more readily available to practitioners.

Algorithm 2 Assisted Regularised EA — AREA

```
population = []
generator = RandomGenerator()
for i=1:M do
    net = generator.generate()
    scored_net = net.score()
    population.append(scored_net)
Keep the top N scored networks in the population
history = []
for net in population do
    trained_net = net.train()
    history.append(trained_net)
while time limit not exceeded do
    Sample sub-population, S, without replacement from population
    Select network in S with highest accuracy as parent
    Mutate parent network to produce child
    Train child network
    Remove oldest network from population
    population.append(child network)
    history.append(child network)
chosen_net = Network in history with highest accuracy
```

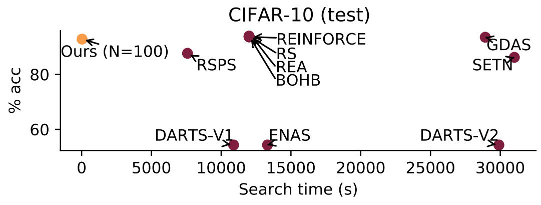


Figure 7. Plot showing the search time (as measured using a 1080Ti) against final accuracy of the proposed NAS-Bench-201 network for a number of search strategies.

Acknowledgements. This work was supported in part by the EPSRC Centre for Doctoral Training in Pervasive Parallelism and a Huawei DDMPLab Innovation Research Grant. The authors are grateful to Eleanor Platt, Massimiliano Patacchiola, Paul Micaelli, and the anonymous reviewers for their helpful comments. The authors would like to thank Xuanyi Dong for correspondence on NAS-Bench-201.

References

Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł., and Lane, N. D. Zero-cost proxies for lightweight NAS. In *International Conference on Learning Representations*, 2021.

Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Chen, W., Gong, X., and Wang, Z. Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Deng, B., Yan, J., and Lin, D. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

Dong, X. and Yang, Y. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the International Conference on Computer Vision*, 2019a.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019b.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Dong, X., Liu, L., Musial, K., and Gabrys, B. NATS-Bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. In *Advances in Neural Information Processing Systems*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Istrate, R., Scheidegger, F., Mariani, G., Nikolopoulos, D., Bekas, C., and Malossi, A. C. I. Tapas: Train-less accuracy predictor for architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In *Conference on Uncertainty in Artificial Intelligence*, 2019.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*, 2018.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, 2018.

Park, D. S., Lee, J., Peng, D., Cao, Y., and Sohl-Dickstein, J. Towards nngp-guided neural architecture search. *arXiv preprint arXiv:2011.06006*, 2020.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018.

Radosavovic, I., Johnson, J., Xie, S., Lo, W.-Y., and Dollár, P. On network design spaces for visual recognition. In *Proceedings of the International Conference on Computer Vision*, 2019.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *Int. Journal of Computer Visi—-on (IJCV)*, 115(3):211–252, 2015.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wistuba, M., Rawat, A., and Pedapati, T. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Xie, S., Zheng, H., Liu, C., and Lin, L. Snas: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, 2019.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020.

Zela, A., Siems, J., and Hutter, F. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

A. NAS Benchmarks

A.1. NAS-Bench-101

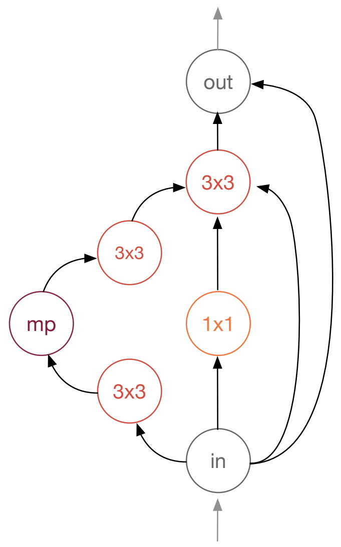
In NAS-Bench-101, the search space is restricted as follows: algorithms must search for an individual cell which will be repeatedly stacked into a pre-defined skeleton, shown in Figure 8c. Each cell can be represented as a directed acyclic graph (DAG) with up to 9 nodes and up to 7 edges. Each node represents an operation, and each edge represents a state. Operations can be chosen from: 3×3 convolution, 1×1 convolution, 3×3 max pool. An example of this is shown in Figure 8a. After de-duplication, this search space contains 423,624 possible neural networks. These have been trained exhaustively, with three different initialisations, on the CIFAR-10 dataset for 108 epochs.

A.2. NAS-Bench-201

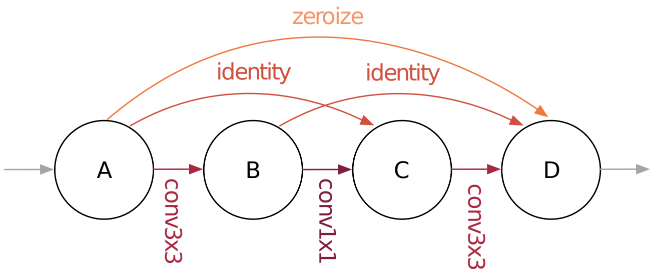
In NAS-Bench-201, networks also share a common skeleton (Figure 8c) that consists of stacks of its unique *cell* interleaved with fixed residual downsampling blocks. Each cell (Figure 8b) can be represented as a densely-connected DAG of 4 ordered nodes (A, B, C, D) where node A is the input and node D is the output. In this graph, there is an edge connecting each node to all subsequent nodes for a total of 6 edges, and each edge can perform one of 5 possible operations (Zeroise, Identity, 3×3 convolution, 1×1 convolution, 3×3 average pool). The search space consists of every possible cell. As there are 6 edges, on which there may be one of 5 operations, this means that there are $5^6 = 15,625$ possible cells. This makes for a total of 15,625 networks as each network uses just one of these cells repeatedly. The authors have manually split CIFAR-10, CIFAR-100, and ImageNet-16-120 (Chrabaszcz et al., 2017) into train/val/test, and provide full training results across all networks for (i) training on train, evaluation on val, and (ii) training on train/val, evaluation on test. The split sizes are 25k/25k/10k for CIFAR-10, 50k/5k/5k for CIFAR-100, and 151.7k/3k/3k for ImageNet-16-120.

A.3. NATS-Bench

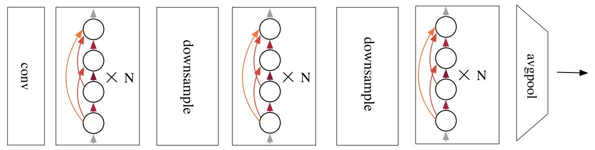
NATS-Bench (Dong et al., 2021) comprises two search spaces: a topology search space and a size search space. The networks in both spaces share a common skeleton which is the same as the skeleton used in NAS-Bench-201. The topology search space (NATS-Bench TSS) is the same as NAS-Bench-201 whereby networks vary by operation comprising the network cell. The size search space instead varies the channels of layers in 5 blocks of the skeleton architecture. Every network uses the same cell operations. The choice of operations corresponds to the best performing network in the topology search space with respect to the CIFAR-100 dataset. For each block the layer channel size is chosen from 8 possible sizes (8, 16, 24, 32, 40, 48, 56, 64). This leads to $8^5 = 32768$ networks in the size search space (NATS-Bench SSS).



(a) A NAS-Bench-101 cell.



(b) A NAS-Bench-201 and NATS-Bench TSS cell.



(c) The skeleton for NAS-Bench-101 (N=3), 201 (N=5), and NATS-Bench TSS (N=5).

Figure 8. (a): An example cell from NAS-Bench-101, represented as a directed acyclic graph. The cell has an input node, an output node, and 5 intermediate nodes, each representing an operation and connected by edges. Cells can have at most 9 nodes and at most 7 edges. NAS-Bench-101 contains 426k possible cells. By contrast, (b) shows a NAS-Bench-201 (NATS-Bench TSS) cell, which uses nodes as intermediate states and edges as operations. The cell consists of an input node (A), two intermediate nodes (B, C) and an output node (D). An edge e.g. A$\rightarrow$ B performs an operation on the state at A and adds it to the state at B. Note that there are 6 edges, and 5 possible operations allowed for each of these. This gives a total of 5^6 or 15,625 possible cells. (c): Each cell is the constituent building block in an otherwise-fixed network skeleton (where N=5). As such, NAS-Bench-201 contains 15,625 architectures.

B. Additional Plots

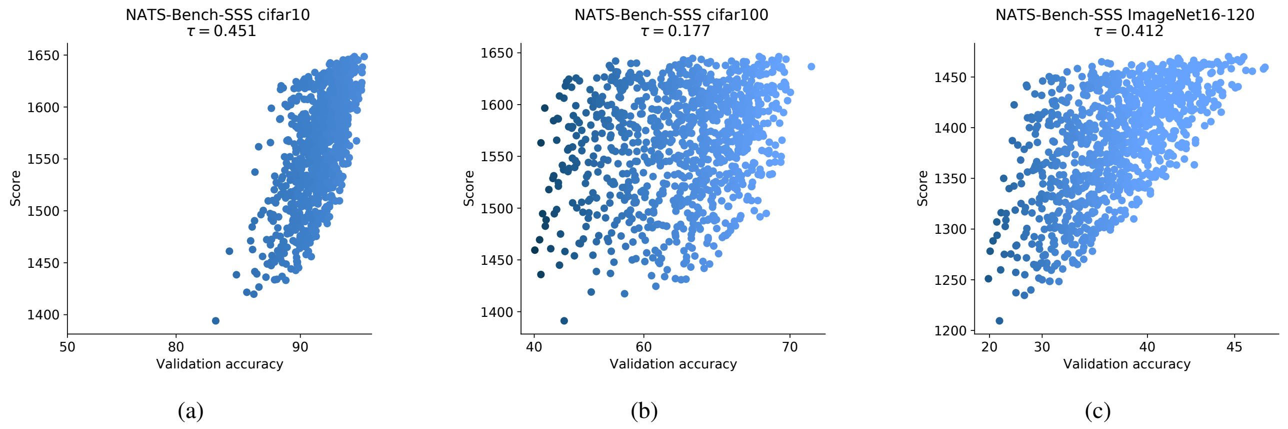


Figure 9. Further plots of our score (Equation 2) for 1000 randomly sampled **untrained** architectures in NATS-Bench SSS against validation accuracy when trained on (a) CIFAR-10, (b) CIFAR-100, and (c) ImageNet16-120.

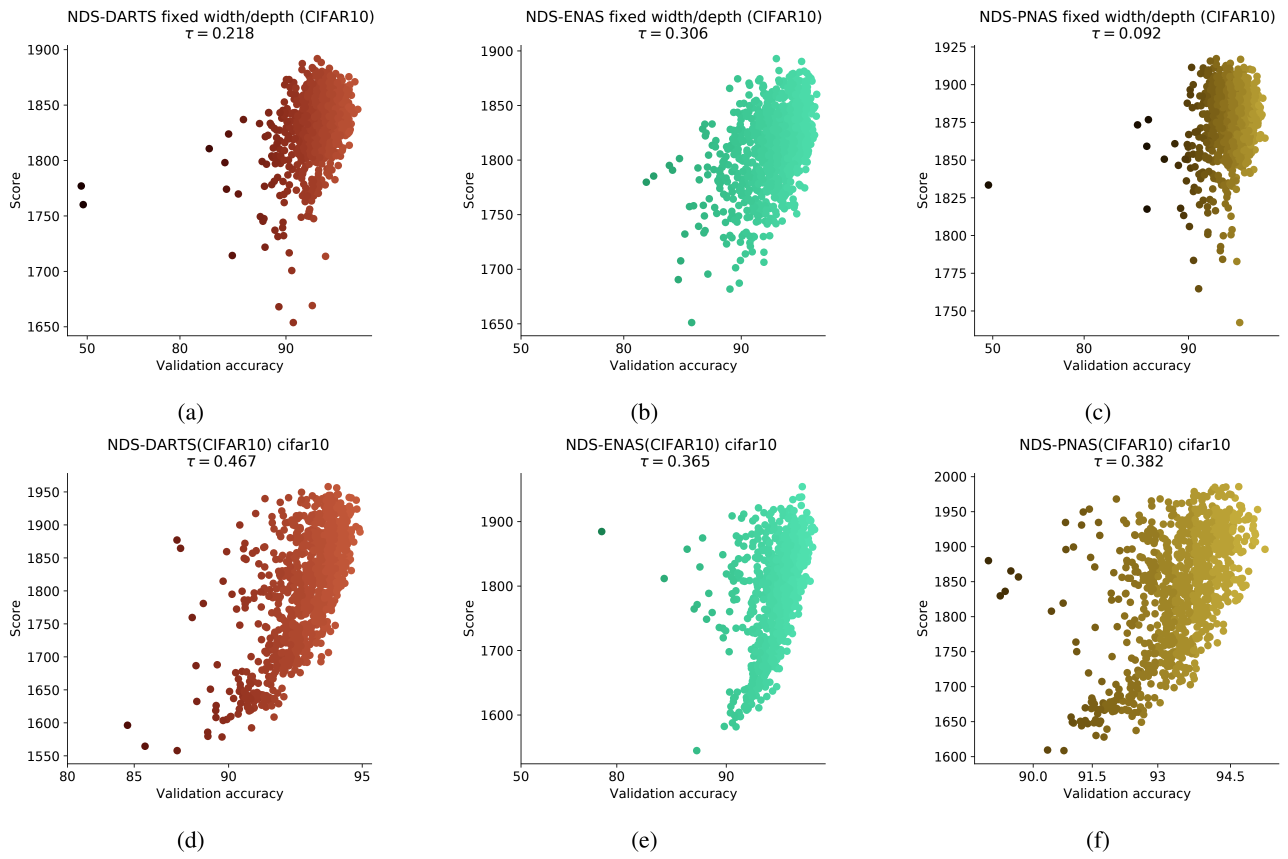


Figure 10. Further plots of our score (Equation 2) for around 1000 randomly sampled **untrained** architectures in NDS-DARTS, NDS-ENAS, and NDS-PNAS against validation accuracy when trained. The top row shows the fixed width and depth variants of the search spaces, while the bottom row shows the variable width and depth spaces.