NASIB: Neural Architecture Search withIn Budget

Abhishek Singh[*]
MIT Media Labs
abhi24@mit.edu

Anubhav Garg
Cisco Systems
anubhgar@cisco.com

Jinan Zhou
Cisco Systems
jinazhou@cisco.com

Shiv Ram Dubey
IIIT Sri City
srdubey@iiits.in

Debojyoti Dutta
Cisco Systems
dedutta@cisco.com

Abstract

We propose a new approach for Neural Architecture Search, called NASIB, which adapts and attunes to the computation resources (budget) available by varying the exploration vs exploitation trade-off. We reduce the expert bias by searching over an augmented search space by introducing *Superkernels*. Our method can yield better networks for different computation resources and different domains beyond image classification of natural images, where one might lack bespoke architecture motifs and domain expertise. We show, on CIFAR10, that it is possible to search over a space that has 12x more candidate operations than the traditional prior art in just 1.5 GPU days, while reaching close to state of the art accuracy. While our method searches over an exponentially larger search space, it could lead to novel architectures that require lesser domain expertise, compared to the majority of the existing methods.

1 Introduction

Neural Architecture Search (NAS) embodies a wide variety of techniques to determine the optimal neural network architecture based on a variety of constraints while optimizing for one or more objectives (Zoph & Le (2017); Baker et al. (2016); Tan et al. (2018); Dong et al. (2018); Cai et al. (2019)). However, in the current form, NAS has the following two limitations which inhibits it from getting used across heterogeneous platforms and diverse tasks:

1. The search method is designed with the assumption that the user will run it till convergence. It is not clear how to leverage these methods when the computational budget for performing architecture search is significantly lower or higher than that used by such methods.
2. The search space used is quite narrow (relatively) and is built on already optimal structural decisions, hence does not provide any significant benefit over a random search.

In this paper, we propose a simple, intuitive and effective solution to both the problems using a common framework, called *NASIB* and present a trade-off between the computational budget and exploration over a vast space of neural network architectures. Unlike the traditional NAS frameworks which start with a pre-filtered fixed search space (where the filtering is done by domain experts), we start with an unbiased search space by considering a huge space of various possible candidate operations (e.g., for CNNs we search over all possible kernels with all combinations of kernel height and width less than 10, which amounts to **81** different kernels at every layer) and dynamically reduce it by decreasing the sampling probability of the ineffective operations. We propose an architecture design method which searches over an exponentially larger search space, requires lesser domain

[*]Work done while at Cisco

expertise, and present an algorithm that performs architecture search under a given budget by adapting to the computation resources available. We empirically demonstrate superiority of the proposed method over existing methods by searching over a space having roughly **12X** times more candidate operations than the majority of the recent search methods under same computation budget without any compromise on the performance of the discovered architecture. To ensure the efficiency of our framework under such a huge set of structural decisions, we use computational budget available for search by reducing the *rate* of dynamic pruning of the search space since such pruning rates affect the amount of resources consumed, a proxy for the cost. This adaptive pruning is the trade-off between exploration and exploitation. Related work has been discussed in Section D of the Appendix.

2 NASIB

2.1 From Computation Resources to Epochs

One key aspect of this work is to map available computation resources in a quantitative way which can be utilized by the search algorithm. Different GPUs exhibit significantly different computation speeds and CPU speed, memory access, interconnection network topology and many more interwoven components are also a key factor in the overall speed which need to be taken into account. Therefore, for our search algorithm to adapt to the available computation resources, a comprehensive search space would be required. However, from an end user's perspective, it is relatively convenient to specify only the number of hours/days and provide the whole system/cluster as a black box to the search algorithm. Hence, rather than developing a complex mapping of computation resources, we calculate total number of epochs available as *total gpu days/time for one epoch*.

2.2 Differentiable Architecture Search

Algorithm 1 NASIB - Neural Architecture Search withIn Budget

```
total_epochs = GPU_days_available / compute_time_for_single_epoch()
Initialize the base network with superkernel at every layer
while epoch < total_epochs (budget) do
    for each layer $\ell$ in the network do
        1. current_policy = sample from $P_1$ and $P_2$ using Bernoulli distribution
        if current_policy = $P_1$ then
            sample candidate operations based on distribution from eq. 3
        else
            sample candidate operations based on distribution from eq. 4
        end
        2. Update weights w based on eq. 1
        3. Update architecture parameters $\alpha$ based on eq. 2
        4. Update sampling probability for $P_1$
    end
end
For each layer, retain the operations with highest $\alpha$.
```

The proposed method is built upon the existing differentiable architecture search methods (Liu et al. (2018); Cai et al. (2019); Xie et al. (2019)). We sample two candidate operations at every layer as described in Cai et al. (2019) to ensure the network fits on GPU memory.

$$x_\ell = \sum_{j=1}^{N} O_{\ell,j}(x_{\ell-1}) \quad (1) \qquad x_\ell = \sum_{j=1}^{N} \frac{\exp(\alpha_j^\ell)}{\sum_{i=1}^{N}\exp(\alpha_i^\ell)} O_{\ell,j}(x_{\ell-1}) \quad (2)$$

Here, $O_{\ell,j}$ is the j^{th} candidate operation at the layer ℓ, x_ℓ is the output of the layer ℓ and N is the total number of candidate operations. All operations are assigned a single scalar α, also called architecture parameters. The search method runs in two different modes, training of weights is done by performing one step of forward pass and backward pass using eq. 1 on the training data and then after completing one epoch, architecture parameters are trained using eq. 2 on the validation data. In both the steps, the value of N=2 even though the number of candidate operations are higher as we do sampling of two candidate operations as described in Cai et al. (2019).

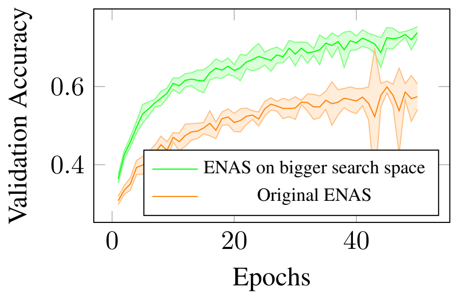

Figure 1: Mean accuracy with standard deviation for the two set of experiments on the ENAS implementation. The green curve corresponds to the accuracy obtained with a bigger search space and the orange curve corresponds to the running of the original ENAS.

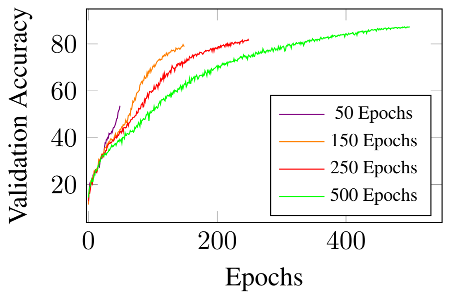

Figure 2: Search progress under different computational budget on CIFAR10 dataset with superkernel search space. We keep all hyperparameters same except the computational budget, which is provided in such a way that they get mapped to the mentioned value of target epochs.

2.3 Policy based Sampling from Search Space

To account for the exploration and exploitation phase during the search, we use two policies P_1 and P_2, respectively. For exploration, we encourage sampling of the underexplored operations by storing their sampling frequency. Under the policy P_1 and any given layer ℓ, the sampling probability p^i_ℓ of a operation i is computed by calculating the negative softmax of previously sampled frequencies as per eq. 3,

$$p^i_\ell = \frac{\exp(-freq_\ell[i])}{\sum_{j=1}^{N} \exp(-freq_\ell[j])} \quad (3) \qquad p^i_\ell = \frac{\exp(\alpha^\ell_i/\lambda)}{\sum_{j=1}^{N} \exp(\alpha^\ell_j/\lambda)} \quad (4)$$

The negative sign in front of all the sampled frequency numbers allows to assign higher probability to the operations with lower sampling frequency in the previous training. Hence, it encourages the exploration of unexplored operations. The reasoning behind using softmax of frequencies instead of random sampling or upper confidence bound, commonly used methods in reinforcement learning for exploration, is to explicitly encourage those operations which have been sampled with low frequency before to encourage the diversity in the architecture search. For the policy P_2, an operation is sampled with probability based on the softmax of architecture parameters, α as shown in eq. 4. Since the policy P_2 is expected to be purely exploitative in nature, we keep the value of λ to be very low. P_1 is sampled with exponentially decaying probability p_1, hence, even though we start with a big search space, the search space gets pruned *softly* as we proceed in the exploitation phase guided by the policy P_2.

2.4 Searching for CNN Architectures

In order to reduce the expert bias in the design of the search space, we introduce the notion of superkernel. A superkernel of size m can be viewed as a square having size m which encompasses all possible combinations of rectangles of dimensions $i \times j, \forall i, j \leq m$. Thus, a superkernel of size m would result in m^2 different rectangles. All such rectangles can be viewed as kernels of dimensions $i \times j$. We use macro search space for architecture search and at every layer we utilize a single superkernel. Therefore, at every layer, we allow m^2 candidate operations to be searched over, but at any given time only two operations are sampled from the superkernel which reduces the memory requirement. The overall search space for the architecture search with superkernel is m^{2*l} while for other existing methods it is n^l. Here l is the number of layers, n is number of candidate operations and m is superkernel size, and typical values for them are 12, 5, and 9 respectively. Detailed experimental setup can be found in section A.3 of Appendix.

3 Experiments and Results

We perform several sets of experiments to test our hypothesis and demonstrate the efficacy of the proposed method. All experiments are performed on CIFAR10 dataset using Nvidia V100 GPU. To test our hypothesis on leveraging more number of candidate operations, we use the same code base of ENAS (Pham et al. (2018)) and simply increase the search space by adding more number of

Table 1: Comparison of performance on two standard benchmarks. Search time is reported in GPU days. $\#OPs$ refers to the number of candidate operations in the search space.

Network	Dataset	Params	Test error	#OPs	Search time
NASNet-v3 (Zoph & Le (2017))	CIFAR10	37.4M	3.65	13	1800
AmoebaNet-B (Real et al. (2017))	CIFAR-10	34.9M	2.13	19	3150
PNAS (Liu et al. (2017))	CIFAR10	**3.2M**	3.41	8	225
ENAS (Pham et al. (2018))	CIFAR-10	4.6M	3.54	6	**0.45**
DARTS (Liu et al. (2018))	CIFAR-10	4.6M	2.76	7	4
NAO (Luo et al. (2018))	CIFAR-10	128M	**2.07**	11	200
ProxylessNAS (Cai et al. (2019))	CIFAR-10	5.7M	2.08	6	8.3
SNAS (Xie et al. (2019))	CIFAR-10	2.85M	2.8	7	1.5
Graph Hypernetworks (Zhang et al. (2019))	CIFAR-10	2.84M	5.7	8	0.84
NASIB	CIFAR10	6.71M	3.57	**81**	1.5

convolution filters with varying kernel sizes. All hyper-parameters and other experimental details are kept exactly the same as provided in their publicly released code base for performing macro search on CIFAR10 dataset. As shown in Figure 1, there is a significant increase (by a margin of **10%**) in the accuracy of ENAS with extended search space throughout the validation phase of architecture search without any change in other hyperparameters and wall clock time. Thus we empirically observe that the diversity in the search space is useful in obtaining high performance.

Similar to the study performed in Xie et al. (2019) over the relative performance of architectures during the search phase, we also compare two of our workloads in Figure 3. To provide a fair comparison we reduce the search space of NASIB-1 to match it with commonly known methods. As it can be observed, NASIB-1 outperforms few other methods when trained with a small search space similar to the other methods, we attribute this to efficient utilization of computational budget. Note that while the curve corresponding to DARTS converges faster than NASIB-1, Xie et al. (2019) showed that there is a strong inconsistency with the performance of its child network. We also plot the NASIB-2 curve which shows efficient utilization of budget under a significantly larger search space.

We show how our method adapts to the computation resources available for the NAS process by searching for architectures on CIFAR10 for four different scenarios. All four scenarios have been formed by allocating different computation budget for performing the search. In figure 2, we plot validation accuracy as the search progresses for all four workloads. It can be observed from the plot that slope of the curve decreases as the workloads approach close to their target epochs and this slope varies significantly for all four different workloads which means lower the number of epochs available, the model spends relatively lesser time in exploring the search space and hence converges quickly by entering the exploitation phase. We compare our result with other widely known NAS methods in Table 1. The proposed method uses relatively bigger search space of candidate operations at every layer in the architecture. We fine tune our discovered model and retrain it. We observe that while the performance of architectures during search is substantially different for different methods, this difference does not have any correlation after the fine tuning of the architectures. We attribute this to the use of other hyperparameters and clever training tricks which provide greater contribution to the re-tuned training of networks.

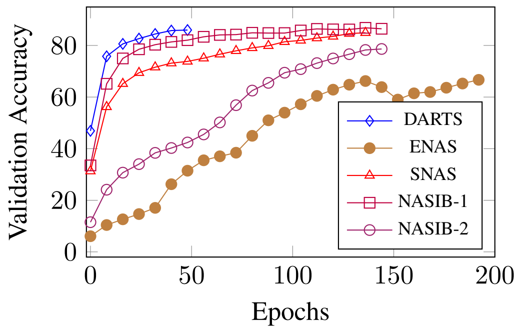

Figure 3: Here, NASIB-1 refers to the architecture search performed using the widely used narrow search space and NASIB-2 refers to the search performed using superkernels.

4 Conclusion

In this paper we have two insights. First we perform architecture search over a computational budget and second, we search over a good representative search space instead of an ad-hoc selection of few candidate operations. This paper could lead to several new directions in NAS and there are different ways in which this work can be extended e.g. combining the superkernel based search space with the multi-objective search could discover interesting architectures. Next, we could leverage the same insight in searching for the cell based architectures like RNNs and micro search space for CNNs. We believe that this work would bring a different perspective to the community working in NAS and initiate the discussion about the search space and budget oriented architecture search.

References

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. URL `http://arxiv.org/abs/1611.02167`.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HylVB3AqYm`.

Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures, 2018. URL `https://openreview.net/forum?id=B1NT3TAIM`.

Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017. URL `http://arxiv.org/abs/1712.00559`.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable Architecture Search. *ArXiv e-prints*, June 2018.

Luo, R., Tian, F., Qin, T., and Liu, T. Neural architecture optimization. *CoRR*, abs/1808.07233, 2018. URL `http://arxiv.org/abs/1808.07233`.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL `http://arxiv.org/abs/1802.03268`.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. volume abs/1703.01041, 2017. URL `http://arxiv.org/abs/1703.01041`.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. URL `http://arxiv.org/abs/1807.11626`.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rylqooRqK7`.

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rkgW0oA9FX`.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *ICLR 2017*, 2017. URL `https://arxiv.org/abs/1611.01578`.