Enabling NAS with Automated Super-Network Generation

J. Pablo Muñoz[1], **Nikolay Lyalyushkin**[2], **Yash Akhauri**[1], **Anastasia Senina**[2], **Alexander Kozlov**[2], **Nilesh Jain**[1]

[1]Intel Labs, [2]Intel Corporation

{pablo.munoz, nikolay.lyalyushkin, yash.akhauri, anastasia.senina, alexander.kozlov, nilesh.jain} @intel.com

Abstract

Recent Neural Architecture Search (NAS) solutions have produced impressive results training super-networks and then deriving subnetworks, a.k.a. child models that outperform expert-crafted models from a pre-defined search space. Efficient and robust subnetworks can be selected for resource-constrained edge devices, allowing them to perform well in the wild. However, constructing super-networks for arbitrary architectures is still a challenge that often prevents the adoption of these approaches. To address this challenge, we present BootstrapNAS, a software framework for automatic generation of super-networks for NAS. BootstrapNAS takes a pre-trained model from a popular architecture, e.g., ResNet-50, or from a valid custom design, and automatically creates a super-network out of it, then uses state-of-the-art NAS techniques to train the super-network, resulting in subnetworks that significantly outperform the given pre-trained model. We demonstrate the solution by generating super-networks from arbitrary model repositories and make available the resulting super-networks for reproducibility of the results.

Introduction

The great variety of edge devices in which *Deep Learning* models might be deployed has motivated the development of solutions for optimizing these models and improving their performance on a selected device. A successful approach has been to use Neural Architecture Search (NAS) to discover efficient and robust models that can be deployed in the wild on a particular edge device.

Early NAS solutions trained each candidate architecture from scratch, taking a significant amount of time to produce decent results. *Weight sharing* has allowed more efficient NAS approaches that maintain a single structure, i.e., a super-network, sometimes referred to as the *one-shot* (Bender et al. 2018; Liu, Simonyan, and Yang 2018; Pham et al. 2018; Cai, Zhu, and Han 2019; Xie et al. 2020; Guo et al. 2020), *single-stage* (Yu et al. 2020), or *once-for-all* (Cai et al. 2020) network depending on other properties of each NAS approach. A few of these approaches are hardware-aware, for instance, by incorporating the target device's latency measurements, but are limited to a single target device , e.g., (Cai, Zhu, and Han 2019), having to run again the NAS procedure when an optimal model for a new target device is requested. More recently, NAS approaches have been able to decouple train and search stages, enabling a single training session and the repeated search of new derived models for multiple target devices with different hardware configurations (Cai et al. 2020; Yu et al. 2020). These *once-for-all* or *single-staged* super-networks produce large spaces of subnetworks. The weights of the super-network are optimized and exploration of suitable sub-networks is guided by a *search strategy* and a *performance estimation strategy* (Elsken, Metzen, and Hutter 2019). In some cases, these approaches produce models that can be immediately deployed, e.g., (Cai, Zhu, and Han 2019; Cai et al. 2020; Yu et al. 2020), while in other cases, additional fine-tuning of the candidate subnetworks can yield better accuracy. Smaller subnetworks might satisfy the requirements of resource-constrained devices while maintaining the accuracy of bigger subnetworks.

The construction of the super-network, and hence the generation of the search space, present several challenges. Given that there is no consensus on the optimality of one-shot NAS, driven by doubts on whether super-network optimization aligns with the objective of NAS (Zhang, Zhang, and Yang 2021), generating a search space that contains well performing sub-networks needs to be automated effectively. Further, existing search spaces are designed for research endeavors which may not align with tasks for which a neural network has to be deployed.

Expert practitioners have to construct these super-networks for instance by overparameterizing a well-known architecture, e.g., MobileNet-V3 (Howard et al. 2019). However, this is usually a complicated process that prevents an average user from further optimizing their existing pre-trained models so they can improve their performance when deployed in the wild.

We present BootstrapNAS, a software framework for automatic generation of NAS super-networks. BootstrapNAS is implemented in the Neural Network Compression Framework (NNCF) (Kozlov et al. 2020). NNCF works with PyTorch and TensorFlow, and supports a wide range of compression algorithms such as quantization and pruning. We utilize NNCF's graph tracing and analysis capabilities to enable the BootstrapNAS solution.

Our contributions can be summarized as follows:

- A software framework with a set of methods for automated super-network generation.
- The application of state-of-the-art methods for training the automatically generated super-network.
- Demonstration of the feasibility of the proposed methods and preliminary results on two examples of super-networks.

Automated Generation of Super-Networks

Table 1: Notation

Ω	Super-network	L^Ω	Set of layers of Ω
a_i	Subnetwork i	l_i^Ω	Layer i of Ω
a_{min}	Minimal subnetwork	L^i	Set of layers of a_i
a_{max}	Maximal subnetwork	l_j^i	Layer j of a_i
m	Pre-trained model	L_s^i	Set of *static* layers of a_i
A	Set of all subnetworks	L_e^i	Set of *elastic* layers of a_i
A_o	Set of Pareto-optimal subnetworks		

Super-network. A super-network, Ω, is a neural network that is composed of a set of layers, L^Ω, which for our purposes we divide into two subsets, L_e^Ω (elastic layers) and L_s^Ω (static layers), i.e., $L^\Omega = L_e^\Omega \cup L_s^\Omega$, and a set of weights, W, associated with those layers. Notice that in our formulation, we start by considering a super-network, as a typical neural network, and it is through BootstrapNAS' top-down approach that *elasticity* (defined below) is automatically added, and which allows the later derivation of subnetworks.

Subnetwork. A subnetwork or child model, a_i, is a neural network that shares some (or all) of the elements of the super-network, Ω, s.t., if L^i is the set of the layers in a_i, then $L^i \subseteq L^\Omega$. Notice that L^i is also composed of the two types of layers introduced above, i.e., $L^i = L_s^i \cup L_e^i$.

Static Layers. We refer to L_s^i (or L_s^Ω for that matter) as *static* layers, and, as we will discuss below, they have a fixed configuration for all the subnetworks and the super-network, i.e., $\forall l(l \in L_s^i \Leftrightarrow l \in L_s^\Omega)$.

Elastic Layers. L_e^i are those layers that can have variable values in their properties. For instance, in the case of a convolution layer, they might have variable values for its *width* (number of channels) or *kernel size*, e.g., a layer with x or y number of channels, and zxz kernels., e.g., 7x7. We refer to L_e^i as *elastic* layers.

We denote A to be the set of all the subnetworks (which includes the super-network). The literature uses the term *elasticity* to describe the property of layers that can vary their configurations, e.g., layer j in subnetwork a_i i.e., l_j^i, might have x number of *active* channels, while another subnetwork, a_k, might have y number of *active* channels for the same layer, l_j^k. Although the same layer is present in both subnetworks, their layer configurations might be different. Thus, subnetworks are partitions of the super-network, and have different values for the properties of their layers, e.g., width or kernel size in the case of convolution layers, unless the particular subnetwork in consideration is also the super-network.

Pre-trained Model to Super-Network. BootstrapNAS uses NNCF's capabilities to trace a given pre-trained model, m, and convert it into a super-network, Ω, s.t., if L^m are the layers in m, $\forall l(l \in L^m \Leftrightarrow l \in L^\Omega)$ and $W^\Omega = W^m$. The conversion procedure must guarantee that both models, that is, the pre-trained model and the super-network, will produce similar results on a dataset, D_{val}, i.e., $Cost(m, D_{val}) \cong Cost(\Omega, D_{val})$. This is validated by BootstrapNAS after the super-network has been generated. Before conversion of a pre-trained model to a super-network all layers are static, i.e., $L^\Omega = L_s^\Omega$. BootstrapNAS detects layers that can be made elastic, e.g., a convolution layer, by checking the type of the underlying operation in the layer in consideration and comparing it to the supported operations that can be made elastic. The selected static layers becomes elastic without rewriting the model's code by injecting a mechanism that can capture inputs and parameters before the layer's execution, apply transformations on the layer's tensors and run the underlying operation with the modified parameters and inputs.

In addition to varying its width and internal layer properties, a subnetwork, a_i, can be shallower than its parent super-network, so it is possible that $|L^i|$ may be less than $|L^\Omega|$. The change in depth is accomplished by *omitting* individual layers or groups (blocks) from the super-network to derive subnetwork a_i. BootstrapNAS accomplishes this omission of layers by temporarily removing them from the computational graph if the subnetwork is selected during training. During the conversion of the pre-trained model to a super-network, BootstrapNAS automatically detects layers or blocks that can be skipped by analyzing groups of layers, and determining whether they could be skipped/removed without creating inconsistencies between the output tensors of the previous block and the input dimensions of the following block. BootstrapNAS checks for inconsistencies that might occur when $activating$ certain number of channels in a layer, since this number has to be consistent with its adjacent layer(s).

BootstrapNAS automatically generates the NAS search space by creating a configuration of elasticity for each layer. It starts by considering the maximum possible value of a layer's property based on the value of this property on the original pre-trained model, e.g., number of channels for a layer, and then including alternative configurations with smaller values and steps. For instance, if the pre-trained model used 512 channels in a layer, BootstrapNAS can generate alternative configurations, e.g., $\{512, 256, 128\}$ for the possible number of channels in the derived subnetworks. The number of alternatives, stopping criteria and decreasing step is easily configurable. Otherwise, defaults are used. The search space generation also takes into account the blocks that might be skipped to model how subnetworks will vary in depth. A super-network can easily end up deriving billions of possible subnetworks, depending on how many possible

configurations BootstrapNAS might allow on each layer.

Minimal and Maximal Subnetworks. A subnetwork a_i is considered to be the minimal subnetwork, a_{min}, if it configuration uses the minimal possible values for each elastic dimension on each elastic layer. On the contrary, a differente subnetwork a_i is considered to be maximal if it uses the maximal value for each elastic dimension on each elastic layer. Note that a_{max} is equivalent in its architecture to the given pre-trained model.

Super-Network Training and Subnetwork Search

The super-network generated from the pre-trained model is suitable to the application of state-of-the-art super-network training techniques. For instance, a proven algorithm is *Progressive Shrinking* by (Cai et al. 2020). As its name suggests, it trains the super-network by allowing the sampling of smaller random subnetworks at each training stage (kernel size, depth, and width), hence increasing the variety of subnetwork configurations. There are other techniques for training super-networks. For instance, instead of focusing in subnetworks of a decreasing size for each stage, BootstrapNAS can apply the "sandwich" rule proposed in (Yu and Huang 2019), in which, at each batch of data, a few subnetworks are sampled: the minimal subnetwork, a_{min}, the maximal subnetwork, a_{max} and other n randomly sampled subnetworks. The gradients are aggregated and the weights of the super-network are updated accordingly.

Knowledge distillation can also be applied during training. The soft labels from the original pre-trained model, m or from the maximal subnetwork, a_{max} can be used to compute the loss of the sampled subnetworks. Using the soft labels of a_{max} is referred to as *inplace distillation* in the literature (Yu and Huang 2019).

BootstrapNAS' implementation has an *elasticity handler* object that maintains a registry of the possible configurations that a elastic layer (or the set of elastic blocks in the case of depth) might take, allowing for an efficient sampling of subnetworks. When a subnetwork configuration is selected, BootstrapNAS activates the corresponding configuration at each layer, so the forward and backward passes can be done.

The level of elasticity depends on the size of the given pre-trained model, as well. An overparameterized pre-trained model will allow for the generation of a larger search space. Notice that although an immense number of subnetworks can be derived from the super-network, the space required to store all this information never exceeds the space required to store the super-network, which is a great benefit of weight-sharing approaches. BootstrapNAS' cost to maintain the information of the possible configurations that a layer can have is minuscule in comparison with the size of the model.

As defined above, A is the set of all the possible subnetworks that can be derived from a super-network. Once BootstrapNAS completes the super-network training stage, its next goal is to find k Pareto-optimal subnetworks. That is, BootstrapNAS constructs a set, $A_o \subseteq A$, s.t., $|A_o| = k$. BootstrapNAS currently uses the Non-dominated Sorting Genetic algorithm II (NSGA-II) by (Deb et al. 2002) as default algorithm to search for the set of Pareto-optimal subnetworks. NSGA-II evolves a population and then ranks the various configurations to produce a set of non-dominated solutions. Although BootstrapNAS currently uses NSGA-II by default, nothings prevents BootstrapNAS to incorporate other search algorithms in a subsequent search.

Evaluation of Two Examples of Automatically Generated Super-Networks

Experimental Setup. To demonstrate the capabilities for super-network generation of BootstrapNAS (and posterior super-network training and searching), we used the already well-optimized models from (Phan 2021), a repository which stores popular models that have been efficiently trained with CIFAR-10 (Krizhevsky 2009). We selected ResNet-50 (He et al. 2016) and MobilenetV2 (Sandler et al. 2018) for BootstrapNAS to generate the corresponding super-networks, train them, and then search for outperforming subnetworks. For the search stage, BootstrapNAS used NSGA-II with a population size of 50, crossover rate of 0.9, and a mutation rate of 0.02 to search for a Pareto-optimal set.

Results. BootstrapNAS successfully converted the pre-trained models into super-networks. As Illustrated by Figure 1, NSGA-II successfully discovered outstanding subnetworks after 3,000 subnetwork evaluations. These subnetworks outperformed the original pre-trained model in both objectives (MACs and accuracy).

As Illustrated in Figure 1, BootstrapNAS' ResNet-50 super-network contains a myriad of subnetworks that outperform the given pre-trained model. For instance, BootstrapNAS B-RC requires $\sim 2.81\times$ fewer MACs than the given pre-trained model while slightly improving the top 1 accuracy (from 93.65% to 93.70%). If a small drop in accuracy of $\sim 1\%$ is allowed, BootstrapNAS discovers subnetworks, e.g., BootstrapNAS A-RC, that require $\sim 3.65\times$ fewer MACs than the original pre-trained model.

In the case of BootstrapNAS' MobilenetV2 super-network, NSGA-II produces a Pareto front with several subnetworks that outperform the given pre-trained model, e.g., BootstrapNAS B-MC, requires $\sim 2.39\times$ fewer MACs than the pre-trained model while maintaining its top 1 accuracy ($\sim 93.91\%$). If a small drop in accuracy of $\sim 1\%$ is allowed, BootstrapNAS discovers subnetworks, e.g. BootstrapNAS A-MC, that require $\sim 3.56\times$ fewer MACs than the pre-trained model.

Conclusion

BootstrapNAS is a software framework within NNCF for automatic generation of NAS super-networks. BootstrapNAS takes as input a pre-trained model, analyzes its architecture, converts it into a super-network, applies state-of-the-art techniques for training the super-network, and then automatically discovers outperforming subnetworks.

Currently, BootstrapNAS' focus is on convolutional neural networks. In the future, we plan to support other domains, e.g., Natural Language Processing (NLP) models.

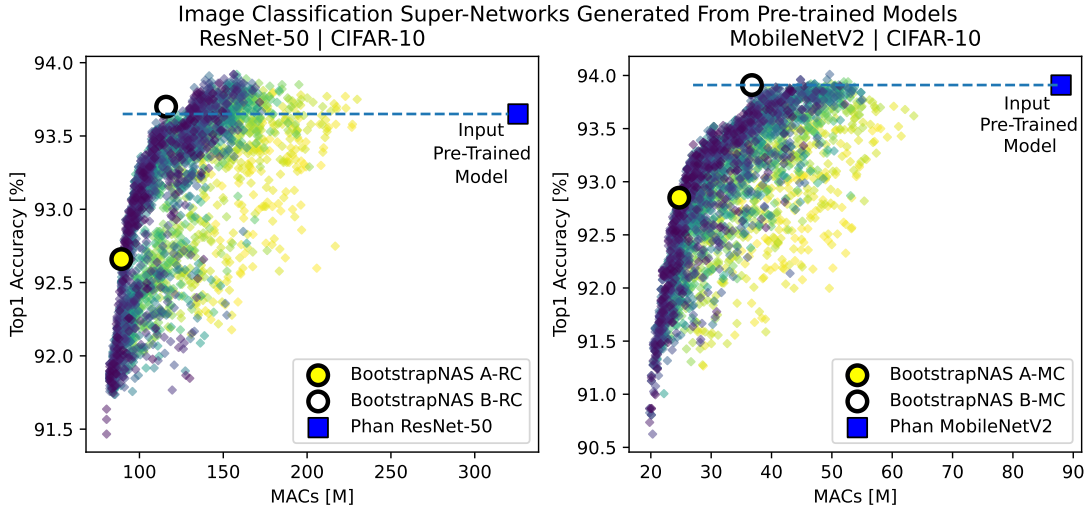

Figure 1: Subnetwork search using NSGA-II on the FP32 model spaces of two super-networks automatically generated by BootstrapNAS from pre-trained models. Each plot illustrates the progression of NSGA-II when discovering a Pareto front. The darker marks represent subnetworks evaluated late in the search process. All subnetworks above the dashed lines outperform the pre-trained models given as input, in both objectives, accuracy and MACs. All subnetworks to the left of the input model outperform the input model in MACs. We highlight two subnetworks in addition to the given pre-trained model.

BootstrapNAS is an open-source project that will be released as part of the Neural Network Compression Framework (NNCF). The example super-networks presented in this document are available for reproducibility of the results.

References

Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. V. 2018. Understanding and Simplifying One-Shot Architecture Search. In *ICML*.

Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; and Han, S. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations*.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, 20(55): 1–21.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, 544–560. Springer.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; Le, Q. V.; and Adam, H. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Kozlov, A.; Lazarevich, I.; Shamporov, V.; Lyalyushkin, N.; and Gorbachev, Y. 2020. Neural network compression framework for fast model inference. *arXiv preprint arXiv:2002.08679*.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. DARTS: Differentiable Architecture Search. arXiv:1806.09055.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameter Sharing. arXiv:1802.03268.

Phan, H. 2021. huyvnphan/PyTorch_CIFAR10.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2020. SNAS: Stochastic Neural Architecture Search. arXiv:1812.09926.

Yu, J.; and Huang, T. S. 2019. Universally Slimmable Networks and Improved Training Techniques. *CoRR*, abs/1903.05134.

Yu, J.; Jin, P.; Liu, H.; Bender, G.; Kindermans, P.; Tan, M.; Huang, T. S.; Song, X.; Pang, R.; and Le, Q. V. 2020. BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models. *CoRR*, abs/2003.11142.

Zhang, Y.; Zhang, Q.; and Yang, Y. 2021. How Does Supernet Help in Neural Architecture Search? arXiv:2010.08219.