Poster: Scaling Up Deep Neural Network Optimization for Edge Inference†

Bingqian Lu
UC Riverside

Jianyi Yang
UC Riverside

Shaolei Ren
UC Riverside

I. Background and Motivation

Deep neural networks (DNNs) have been increasingly deployed on and integrated with edge devices, such as mobile phones, drones, robots and wearables. Compared to cloud-based inference, running DNN inference directly on edge devices (a.k.a. edge inference) has major advantages, including being free from the network connection requirement, saving bandwidths, and better protecting user privacy [1]. To achieve a satisfactory user experience, the DNNs running on an edge device must be judiciously optimized [2]–[8]. Nonetheless, the space of DNN designs is exponentially large, involving combinations of network architectures and compression schemes. Furthermore, there is not a single DNN model that performs optimally on all edge devices. Thus, device-aware DNN optimization is mandated [1], [5], [9], [10].

Designing an optimal DNN for even a single edge device needs repeated design iterations and can be very time-consuming [11], [12]. Worse yet, DNN model developers often need to serve extremely diverse edge devices. For example, in the mobile market alone, there are thousands of system-on-chips (SoCs) available. Only top 30 SoCs can each take up more than 1% of the share, and they collectively account for 51% of the whole market [1]. As a consequence, it has become crucially important to scale up the optimization of DNNs for edge inference.

II. State of the Art and Limitations

Network architecture search (NAS) is an important knob for DNN optimization. In general, device-aware NAS is guided by an objective function, e.g., $\text{accuracy} \times \text{weight}_{1} \times \text{energy} + \text{weight}_{2} \times \text{latency}$. Because of the large search space, it is crucial to efficiently evaluate the resulting inference accuracy/latency/energy performance given a DNN candidate [13]–[17]. Towards this end, recent studies have proposed performance predictors or lookup tables to evaluate the objective function in terms of NAS (as well as model compression), without actually deploying or running each candidate DNN on the device [3], [5], [6], [13]–[18].

Nonetheless, building latency/energy predictors or lookup tables is time-consuming by itself [3], [4], [8], [15], [17], [19]. For example, according to [8], the average latencies of 4k random sample DNNs are measured on a mobile device and then used to build an average latency predictor for that specific device (plus additional 1k samples for testing). Assuming 30 seconds for each measurement, it takes a total of 40+ hours to collect training and testing samples. More crucially, the resulting performance predictors or lookup tables only provide good predictions for the device on which the performance is measured, and cannot transfer to a different device. In addition, even with performance predictors available for each edge device, the total search cost (e.g., evolutionary search [14]–[17]) for many diverse devices is still non-negligible.

In summary, the existing device-aware DNN optimization still takes a large amount of time and efforts for even a single device [3], [4], [8], [18], [19], and cannot scale well to extremely diverse edge devices.

III. Proposed Approach: Learning to Optimize

A. Overview

Our key idea is learning to optimize: instead of performing DNN design optimization repeatedly (once for an individual device), we learn an optimizer from DNN optimization on sample devices, and then apply the optimizer to new unseen devices and directly obtain the optimal DNN design.

More specifically, we take a departure from the existing practice by: (1) leveraging new performance predictors that can estimate the resulting inference latency/energy performance given a DNN-device pair; and (2) using an automated optimizer which directly outputs the optimal DNN design given a device. This is illustrated in Fig. 1.

B. Training Performance Predictors and Optimizer

Our proposed design builds on top of two-stage training.

Stage 1: Training performance predictors. The accuracy predictor follows the same design as the ones used in [3], [19], since it is measured on a reference dataset without dependence on devices. On the other hand, the latency/energy predictors can be based on neural networks, which takes as input both device feature $d$ and DNN design representation $x$ and outputs the respective performance. They are each trained by running DNNs with sampled designs on training devices and using mean squared error as the loss function. The key difference between our design and [3], [19] is that our predictors use device features as part of the input and hence can apply to new unseen devices without building new predictors.

We denote the set of training device features as $\mathcal{D}^T$, where each element $d \in \mathcal{D}^T$ corresponds to the feature of

†This work was supported in part by the U.S. National Science Foundation under grants CNS-1910208 and CNS-2007115.
one available training device. We denote the trained accuracy/energy/latency predictor neural network by \( \text{Accuracy}(x) \), \( \text{Energy}_d(x; d) \), and \( \text{Latency}_d(x; d) \), respectively, where \( \Theta_A, \Theta_E, \) and \( \Theta_L \) are learnt parameters for the three respective networks. Thus, the predicted objective function \( f(x; d, \lambda) \) can be expressed as

\[
-f(\lambda) + \lambda_1 \cdot \text{Energy}_d(x; d) + \lambda_2 \cdot \text{Latency}_d(x; d).
\]

The accuracy/energy/latency predictor neural networks are called performance networks, to be distinguished from the optimizer network we introduce below.

**Stage 2: Training the automated optimizer.** The current practice of DNN optimization is to repeatedly run an optimizer (e.g., search-based algorithm), once for a single device, to minimize the predicted objective function [3], [19]. Nonetheless, obtaining the optimal DNN design \( x^*(d, \lambda) \) is non-trivial for each device. Here, we leverage the strong prediction power of a fully-connected neural network parameterized by \( \Theta \) to approximate the optimal DNN design function \( x^*(d, \lambda) \). We call this neural network optimizer network, whose output is denoted by \( x_\Theta(d, \lambda) \) where \( \Theta \) is the network parameter that needs to be learnt. Once \( \Theta \) is learnt, when a new device arrives, we can directly generate the corresponding optimal DNN design choice \( x_\Theta(d, \lambda) \).

For training purposes, in addition to features of real available training devices \( D_T \), we can also generate a set of additional synthetic device features \( D_S \) to augment the training samples. We denote the combined set of devices for training as \( D = D_T \cup D_S \), and the training set of optimization parameters as \( \Lambda_T \) which is chosen according to practical needs (e.g., latency may be more important than energy or vice versa).

A straightforward method of training the optimizer network is to use the optimal DNN design \( x^*(d, \lambda) \) as the ground-truth label for input sample \((d, \lambda) \in (D_T, \Lambda_T)\). Nonetheless, generating many training samples of \( x^*(d, \lambda) \), even based on the predicted objective function, can be slow [3], [19]. Alternatively, we directly minimize the predicted objective function \( f(x; d, \lambda) = -\text{Accuracy}(x) + \lambda_1 \cdot \text{Energy}_d(x; d) + \lambda_2 \cdot \text{Latency}_d(x; d) \) in an unsupervised manner, without using the optimal DNN design choice \( x^*(d, \lambda) \) as the ground-truth label [20]. Specifically, given the input samples \((d, \lambda) \in (D, \Lambda)\) including both real and synthetic device features, we optimize the optimizer network parameter \( \Theta \) to directly minimize the following loss:

\[
\min_\Theta \frac{1}{N_T} \sum_{(d, \lambda) \in (D_T, \Lambda_T)} \hat{f}(x_\Theta(d, \lambda); d, \lambda) + \mu ||\Theta||.
\]

Thus, our training of the optimizer network is guided by the predicted objective function only and unsupervised. When updating the optimizer network parameter \( \Theta \), the parameters for performance predictors \( \Theta_A, \Theta_E, \) and \( \Theta_L \) learnt in Stage 1 are fixed without updating. In other words, by viewing the concatenation of optimizer network and performance predictor networks as a single neural network (illustrated in Fig. 1), we update the parameters \( \Theta \) in the first few layers while freezing the parameters \( (\Theta_A, \Theta_E, \Theta_L) \) in the subsequent layers to minimize the loss expressed in Eqn. (1). Once \( \Theta \) is optimized, we can directly obtain the optimal DNN design for a new device.

More details on the training of performance/optimizer networks can be found in [21]. Additionally, we also present in [21] another approach to addressing the high cost of building performance predictors for each device: first building performance predictors on a proxy device and then reusing them on other devices based on performance monotonicity (i.e., if one DNN \( x_A \) has a lower latency than \( x_B \) on the proxy device, then \( x_A \) is also likely faster than \( x_B \) on a new device). Most importantly, under some technical conditions, this new approach is theoretically guaranteed to be optimal.
REFERENCES

[1] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, “Machine learning at Facebook: Understanding inference at the edge,” in HPCA, 2019.

[2] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, “AutoCompress: An automatic dnn structured pruning framework for ultra-high compression rates,” in AAAI, 2020.

[3] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, “APQ: Joint search for network architecture, pruning and quantization policy,” in CVPR, 2020.

[4] H. Cai, L. Zhu, and S. Han, “Once for all: Train one network and specialize it for efficient deployment,” in ICLR, 2019.

[5] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, “On neural architecture search for resource-constrained hardware platforms,” in ICCAD, 2019.

[6] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, “NeuralPower: Predict and deploy energy-efficient convolutional neural networks,” in ACM, 2017.

[7] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “MnasNet: Platform-aware neural architecture search for mobile,” in CVPR, 2019.

[8] H. Cai, L. Zhu, and S. Han, “ProxylesNas: Direct neural architecture search on target task and hardware,” in ICLR, 2019.

[9] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, “HAT: Hardware-aware transformers for efficient natural language processing,” in ACL, 2020.

[10] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “Shufflenet v2: Practical guidelines for efficient cnn architecture design,” in ECCV, 2018.

[11] H.-P. Cheng, T. Zhang, Y. Yang, F. Yan, H. Teague, Y. Chen, and H. Li, “MSNet: Structural wired neural architecture search for internet of things,” in ICCV Workshop, 2019.

[12] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search,” in CVPR, 2019.

[13] L. Wang, Y. Zhao, Y. Jin, Y. Tian, and R. Fonseca, “AlphaX: exploring neural architectures with deep neural networks and monte carlo tree search,” arXiv preprint arXiv:1903.11059, 2019.

[14] B. Ru, X. Wan, X. Dong, and M. Osborne, “Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel,” arXiv preprint arXiv:2006.07556, 2020.

[15] X. Ning, W. Li, Z. Zhou, T. Zhao, Y. Zheng, S. Liang, H. Yang, and Y. Wang, “A surgery of the neural architecture evaluators,” arXiv preprint arXiv:2008.03064, 2020.

[16] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, “Neural architecture optimization,” in NIPS, 2018.

[17] H. Shi, R. P., H. Xu, Z. Li, J. T. Kwok, and T. Zhang, “Multi-objective neural architecture search via predictive network performance optimization,” arXiv preprint arXiv:1911.09136, 2019.

[18] D. Stamoulis, E. Cai, D. Juan, and D. Marculescu, “HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks,” in DATE, 2018.

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

[20] F. Liang, C. Shen, W. Yu, and F. Wu, “Towards optimal power control via ensembling deep neural networks,” IEEE Transactions on Communications, vol. 68, no. 3, pp. 1760–1776, 2020.

[21] B. Lu, J. Yang, and S. Ren, “Scaling up deep neural network optimization for edge inference,” arXiv preprint arXiv:2009.00278, 2020.