You Only Search Once: A Fast Automation Framework for Single-Stage DNN/Accelerator Co-design

Weiwei Chen[1,2], Ying Wang[1], Shuang Yang[1,2], Chen Liu[1], Lei Zhang[1]
[1]Institute of Computer Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
Email: {chenweiwei, wangying2009, yangshuang2019, liucheng, zlei}@ict.ac.cn

***Abstract*—DNN/Accelerator co-design has shown great potential in improving QoR and performance. Typical approaches separate the design flow into two-stage: (1) designing an application-specific DNN model with high accuracy; (2) building an accelerator considering the DNN specific characteristics. However, it may fails in promising the highest composite score which combines the goals of accuracy and other hardware-related constraints (e.g., latency, energy efficiency) when building a specific neural-network-based system. In this work, we present a single-stage automated framework, YOSO, aiming to generate the optimal solution of software-and-hardware that flexibly balances between the goal of accuracy, power, and QoS. Compared with the two-stage method on the baseline systolic array accelerator and Cifar10 dataset, we achieve 1.42x~2.29x energy or 1.79x~3.07x latency reduction at the same level of precision, for different user-specified energy and latency optimization constraints, respectively.**

Keywords-Automl,* Hardware/Software co-design*, Acceleration.

I. INTRODUCTION

Deep Neural Networks (DNNs) are entering into the area of mobile and IoT, energy-efficient hardware solutions for neural networks (NNs) inference are becoming urgent needs for edge data processing [1,4]. Researchers in both the algorithmic and architecture communities are conducting an intensive study to create such a system for edge AI. In order to achieve the optimal performance and energy-utility in systems that are designed to run a specific or a specific domain of AI applications, DNN/accelerator co-design coordinating the efforts of algorithm and accelerator is necessary.

We found recent approaches [1-4] of DNN/accelerator co-design commonly fall into two categories. The first one is merely designing networks either manually or automatically targets to specific hardware, and the second is to follow a more complicated two-stage method [1-4]. First, the two-stage co-design flow will evaluate and choose some efficient basic operation units or blocks, which are selected and stacked to construct a DNN architecture with the highest accuracy for the target dataset. Then it will customize the hardware parameters for the chosen blocks and finally form the best accelerator architecture for the model in terms of performance or energy-efficiency [1-4]. Though the two-stage solution optimizes the accelerator architecture for the highest-accuracy DNN model, it cannot guarantee the optimal software/hardware design in cases when the design goal is more complicated than the single goal of model accuracy. In addition, in two-stage methods, there is no feedback from hardware performance indicator to the design of DNN architectures, because the two-stage co-design flow follows an uni-direction of optimization procedure from DNN to the hardware, and perhaps fails the optimality due to the lack of bi-direction software/hardware co-ordination.

This work was supported in part by National Natural Science Foundation of China (No. 61902375).

In this work, we introduce an automatic single-stage DNN/accelerator co-design framework, YOSO. The merit of this framework is straightforward: we jointly search in the "2-dimensional" design space where any combination of software and hardware choices are considered so that the optimality is probably achievable. The major contributions are threefold:

- We propose an RL+LSTM based automation framework, YOSO, which directly searches in the combined DNN and accelerator design space. It is different from all prior works and why we derived the name of *single-stage* co-design. This framework is easily transferable to different applications and supports more complicated multi-objective design optimization.
- We propose a method to quickly evaluate the candidate solutions performance. Notably, (1) we use an auxiliary HyperNet that directly generates the weights of a DNN, the HyperNet can fairly and efficiently evaluate the DNN's accuracy at the cost of single testing run; (2) we adopt the Gaussian Process model as the hardware performance predictor to replace the original time consuming simulation. The proposed methods help us achieve efficient search within 12 hours on single-card GPU.
- We compare our single-stage co-design method with the typical two-stage method on the baseline systolic array accelerator and Cifar10 dataset. We achieve 1.42x~2.29x energy or 1.79x~3.07x latency reduction at the same level of precision, for different energy and latency constraints, respectively.

II. SINGLE-STAGE DNN/ACCELERATOR CO-DESIGN

A. Related works and Motivation

Some recent DNN/accelerator co-design approaches strike a balance between QoR and performance. Kiseok et al. [2] present a manual co-design of the DNN and accelerator for an embedded vision task. They first design a special NN accelerator intended to accelerate the SqueezeNet and then adjust the hyper-parameters of SqueezeNet to make it more efficient on the accelerator. Yifan et al. [3] adopt an algorithm-hardware co-design approach to accelerate a ConvNet model on the embedded FPGA. They employ basic 1x1 convolution block to form the model and design highly customized computing units based on FPGA. However, this method is ad-hoc for the presented system and not portable to different applications. Hao et al. [1] move a step forward, and propose an FPGA/DNN co-design methodology as a two-stage approach: a hardware-oriented neural block organizer for high-accuracy NN search, and a top-down FPGA accelerator design that customizes specialized IP instances for each different neural blocks and then weave the IP instances together to run the organized model.

These works demonstrate the performance potential of DNN/accelerator co-design. However, they mostly follow the

two-stage design flow: they either modify an existing DNN model or design a hardware-oriented DNN model with manual or Neural Architecture Search (NAS) for high accuracy, then design a specific accelerator that works 'best' for the selected DNN model. The two-stage design flow limits the co-design method to search design points in a local space, which may not converge to the optimal solution, especially when we need to consider a complex design goal concerning more factors (e.g., bandwidth, QoS)[2,3,4]. Besides, the two-stage design flow is tedious for a specific application and accelerator architecture, for example, the manually customized IPs for neural blocks as in [1], we still need lots of engineering maneuver if transferred to other application or dataset.

B. Problem Definition

Give the DNN architectures search space $\boldsymbol{\eta}=\{\eta^1,\cdots,\eta^m\}$ with m candidates; the accelerator architecture configurations search space $C=\{c^1\cdots,c^n\}$ with n configurations; the user-provided performance constraints threshold *thres*. Our object is to automatically generate the DNN architecture $\eta*$ with associated accelerator configuration $c*$, which performance *Perf* can satisfies the *thres* while achieving the maximum accuracy A for a machine learning task. The definition can be denoted as:

$$(\eta^*,c^*)=\underset{\boldsymbol{\eta}^*\in\boldsymbol{\eta},\ c^*\in C}{argmax}\ A(\eta,c)$$
$$\text{s.t. } Perf(\eta^*,c^*)<thres \tag{1}$$

The key variables of the accelerator and the basic blocks used to build the NN, are shown in Table 1. The details of these variables are explained in the next chapters.

C. A High-level Overview of the Automated Framework

We give a high-level overview of our single-stage framework in Figure 1. Our automation framework mainly focus on solving two challenges: (1) *Tremendous search space*. (2) *Costly solution evaluation.*

For the former issue, we develop an LSTM-based RL searcher, which efficiently searches in the most rewarding direction in the design space. The search efficiency of the adopted searcher is significantly boosted by avoiding irrelevant candidates. For the latter, we propose some effective methods to evaluate the QoR and hardware performance quickly. For QoR evaluation, we build an auxiliary HyperNet that directly generates the weights of DNN candidates to bypass the expensive procedure of fully-training. For hardware performance evaluation, we utilize machine learning techniques to create performance models without performing long-time hardware simulation. The proposed approaches allow us to identify approximately 10^6 highly relevant hardware/software implementations from 10^{15} possible solutions in few hours, so that the DNN-hardware co-design search is resolvable. The YOSO is carried out in three steps:

Step 1: Fast evaluator construction. The first step takes the target machine learning task, the basic accelerator architecture (e.g., systolic array architecture) and the user constraints as inputs, and then train a HyperNet used to derive different NN architectures in the search stage. After that, performance samples are taken from the accelerator simulator and used to build a performance predictor. In this work, we use energy and latency as the performance metrics for demonstration.

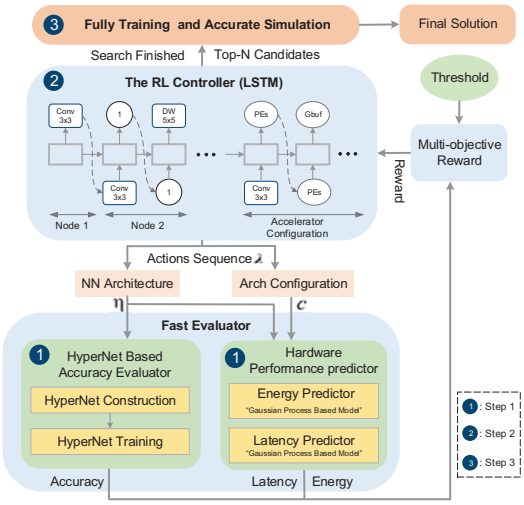

Figure 1. The high-level overview of our single-stage framework, YOSO.

Step 2: Effective design search. An LSTM based RL searcher keeps generating the solutions iteratively. It receives the QoR and performance results from the evaluator to obtain the multi-objective reward, and finally update the controller towards the most rewarding design search direction.

Step 3: Determining the final solution. After the search process reaches a certain number of iterations, we accurately evaluate the top-N promising candidates with the hardware simulation and fully-training, and select the best one as output.

D. Reinforcement Learning Based Search Strategy

The RL searcher network is an LSTM with 120 hidden units, as we treat this combined optimization problem as a sequence generation task. Each candidate solution in the search space is a concatenation of the DNN architecture and the accelerator configurations, denoted as: $\lambda=(\eta,c)=(d_1,\ldots d_S,d_{S+1},\ldots d_{S+L})\in\Lambda^{S+L}$, where S, L are the number of hyper-parameters to present a DNN architecture and an accelerator configuration, respectively. In our experiment, λ consists of 44 hyper-parameters (where S=40, L=4). The LSTM samples each parameter in the λ via a softmax classifier in an autoregressive flow: when generating the *i-th* parameter of λ, previously generated parameters are fed as input. At the initial step, we feed zero as input. The RL can capture the relationship from the reward that can reduce the search attempts. The search cost is reduced by avoiding search solutions that have apparently inferior performance. In this work, we use a multi-objective reward signal $R(\lambda)$.

$$R(\lambda)=A(\lambda)+\alpha_1[l(\lambda)/t\_lat]^{\omega_1}+\alpha_2[e(\lambda)/t\_eer]^{\omega_2} \tag{2}$$

where $A(\lambda)$, $l(\lambda)$, $e(\lambda)$ are the three metrics referring to the *accuracy*, *latency,* and *energy consumption* of λ; t_lat, t_eer are the latency and energy threshold; α_1, ω_1, α_2, ω_2 are four application-specific constants. Consequently, the goal is to find the hyper-parameter λ^* that maximize the expected cumulative reward R it receives in the long run.

$$\lambda^*=\underset{\lambda\in\Lambda}{\arg\max}(E_{p(\lambda;\varphi)}[R(\lambda)]) \tag{3}$$

φ presents the parameters of LSTM that need to be learned in

the search process, p is the probability to select the λ. The REINFORCE algorithm is adopted to update the LSTM parameters, with a moving average baseline to reduce the variance.

$$\nabla_{\varphi} L(\varphi) = -E[(R(\lambda) - b)\nabla_{\varphi} \log p_{(\lambda;\varphi)}(\lambda)] \quad (4)$$

L is the loss function, b presents the average baseline. It is very effective to insert the average baseline mechanism that reduces the variance of gradient estimation while keeping the bias unchanged [6], which can significantly expedite the search.

E. HyperNet Based Accuracy Evaluator

HyperNet Structure. Figure 2 gives a glimpse of the HyperNet on image classification. The HyperNet is stacked by two kinds of blocks as [5]: normal cell and reduction cell. They have a similar structure but with different feature shapes (each function in reduction cell has stride of 2 while the norm cell is 1). Every cell receives input from the previous two cells. When zooming into the cell, it is a directed acyclic graph consisting of an ordered sequence of B nodes(in this work, we use 7 nodes). Each node presents a latent representation (e.g., the feature map) and have the same feature shape in one cell. Each edge is associated with an operation from the candidate operations set. Each node is computed based on two previous feature nodes:

$$I_i = \theta_{(i,j)}(I_j) + \theta_{(i,k)}(I_k) \quad s.t.\ j<i\ \&\ k<i \quad (5)$$

where I_i, I_j, I_k is the *i-th, j-th,* and *k-th* nodes, respectively. $\theta_{(i,j)}$ and $\theta_{(i,k)}$ indicate two operations between nodes. I_0 and I_1 nodes are the outputs of previous two cells. The output of the cell is the concatenation of these nodes that do not give input to other nodes. In this work, we only use 'Relu' as the activation function; 6 operations are included in the operations set: *conv3x3, conv5x5, DWconv3x3, Dwconv5x5, max pooling, average pooling*. So there exists $(6 \times (B-2))!)^4 \approx 5 \times 10^{11}$ candidate DNN architectures in the search space.

HyperNet Training Strategy. We uniformly sample one sub-model (one path) in the HyperNet to perform training and only update the parameters of the selected paths in the HyperNet in each iteration, as shown in the right of Figure 2. Sampling one sub-model from HyperNet is to sequentially select some of the nodes in the cell from bottom to top. Thus, to uniformly sample the network path, for *i-th* node is to be selected, it has to make two decisions: 1) choose two previous nodes as inputs and 2) apply two operations to the chosen inputs accordingly:

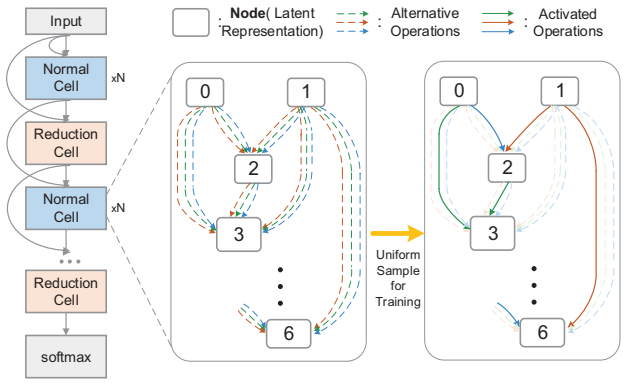

Figure 2. A glimpse of single-shot NAS HyperNet on image classification problem and training strategy.

$$p(I_0) = \cdots = p(I_{i-1}) = 2/i \quad \&\ p(\theta_{(i,j)}) = 1/6 \quad (6)$$

$p(I_{i-1})$ is the probability to choose the *i-1* node, $p(\theta_{(i,j)})$ is the probability to choose operation $\theta_{(i,j)}$.

F. Gaussian Process Model-Based Cost Predictor

For fast cost evaluation given a DNN and accelerator configuration, we adopt the Gaussian Process(GP) regressor to construct a hardware performance predictor, which achieves nearly 2000x speed improvement with less than 5% accuracy loss. We only build the energy and latency predictor, but this approach can be applied to other performance factors.

Performance Predictor. As shown in Figure 3, we compare six regression models for hardware energy prediction. The Gaussian Process(GP) regressor is chosen for it has the lowest mean square error(MSE) among all six regression models. The GP predictor can be denoted as: $y(\lambda) = f(\lambda) + \varepsilon$ .where $f(\cdot) \sim GP(\mu(\cdot), K(\cdot,\cdot))$ is the posterior distribution and $\varepsilon \sim N(\cdot \mid 0, \sigma^2)$ is Gaussian noise. $f(\cdot)$ is drawn from a Gaussian process with mean function $\mu(\cdot)$ and covariance function K, the Radial Basis Function (RBF) kernel is used for K.

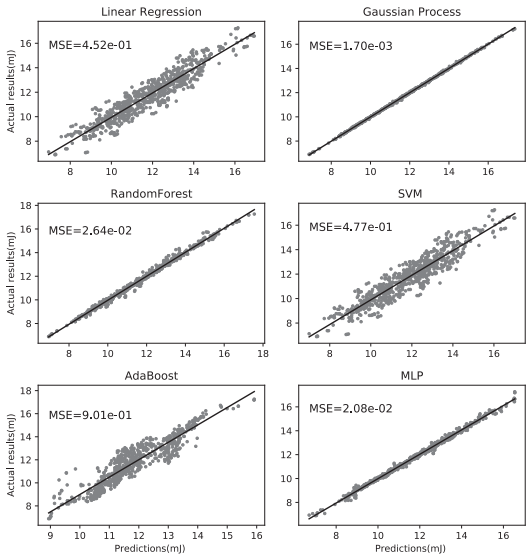

Figure 3. Comparison between different machine learning based regression models for hardware performance predictor, for energy prediction in this example. Every model is built with 3000 training samples and tested on 600 testing samples. MSE refers to the mean squared error between prediction and actual results.

Table 1. Key variables for DNN/accelerator co-design

Variables	Explanation
<N_Cells, R_cells>	Number of normal cells and reduction cells to form the network.
B	Number of nodes in the cell
$<I_j, I_k, \theta_{(i,j)}, \theta_{(i,k)}>$	Every node in the cell. I_j, I_k are two previous nodes to be used as inputs. $\theta_{(i,j)}$, $\theta_{(i,k)}$ present two operations to apply to the two sampled nodes.
Processing Element(PE)	PE array size (range:8x8~16x32)
g_buf	Global buffer size (range:108~1024kb)
r_buf	Register buffer size (range: 64~1024 byte)
data_flow	Four dataflows alternatives: weight stationary(WS), output stationary (OS), row stationary (RS), and no local reuse(NLR)

III. Experimental Results

A. Experiment setup

We evaluate our framework on Cifar10 data set and systolic

array based accelerator. The critical design variables to be searched are listed in Tab 1. We implement the RL controller and HyperNet based on T*ensor- Flow 1.12* with *Cuda 9.0*. For hardware performance estimation, we collect hardware performance profiles from a modified version of *nn_dataflow*. We set 6 blocks (4 norm cells and 2 reduction cells) to form the HyperNet and train it for 300 epochs using a batch size of 144, we adopt a stochastic gradient descent optimizer with a momentum of 0.9 with a standard random crop data argumentation. A cosine learning rate decay strategy is applied with the learning rate range between 0.05~0.0001. Moreover, we regularize the training with L2 weight decay (4x10^-5). We set threshold requirements for energy within 9mJ and latency within 1.2ms, so that the designs that fail these goals will be screened out and only the best designs will be compared.

B. Searching Strategy

Figure 4(a) provides the comparison of our RL based search with random search, both of the two method target on finding a higher composite score. We select every 10-th samples from 10000 iterations as examples. We use the *Reinforce* algorithm to update the RL controller, and use the controller's sample entropy that weighted by 0.0001 to the reward. Our RL based search strategy can find better results compared with random search; it gradually finds solutions that have a higher reward score.

The coefficients in Eq.2 can be adjusted to guide the search toward different optimal regions. For demonstration, We set two different reward functions that target to different optimal regions. We take every 20-th samples from 12000 iterations to project the search process. In Figure 4(b), we demonstrate the co-design search results are towards a user-demand tradeoff between accuracy and energy consumption. Figure 4(c) demonstrates our method towards trade-off between accuracy and latency. As we can see, our RL based search strategy clearly strikes better trade-off among multi-objectives.

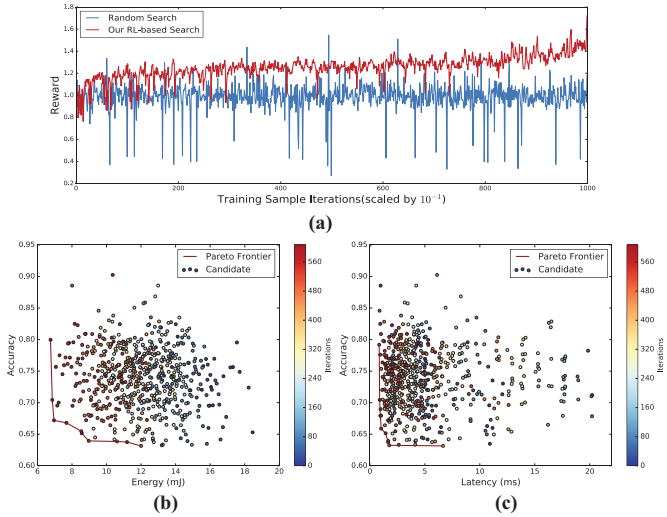

Figure 4. Demonstration of our RL based searching strategy. (a) Comparison with the random search (α_1:0.5 ω_1:-0.4 α_2:0.5 ω_2:-0.4 in Eq.2). (b): our RL method towards trade-off between accuracy and energy consumption (α_1:0.6 ω_1:-0.4 α_2:0.3 ω_2:-0.2 in Eq.2). (c): our method towards trade-off between accuracy and latency (α_1:0.3 ω_1:-0.3 α_2:0.6 ω_2:-0.4 in Eq.2). All tested with threshold requirements: *t_eer:* 9mJ *t_lat:*1.2ms, Average search runtime is near 12 hours on one P100 GPU.

C. Comparison with Two-stage Method

We compare our single-stage search framework with the typical two-stage method. For fair comparison, We reimplement the two-stage method by choosing some existing representative neural networks that have high accuracy [5,6,7,8,9]. These networks are designed in the same neural architectures search space with ours, all the possible accelerator configuration are enumerated to select the best configuration for each network.. Tab 2 shows the final results. *Yoso_lat* is the best solution that achieves good trade-off between accuracy and latency, which achieves the minimum latency of 0.77ms among all solutions. *Yoso_eer* presents the best solution that balances the trade-off between accuracy and energy consumption, which achieves the lowest energy consumption of 7.5mJ with comp-arable accuracy. From the results (normalized to the lowest energy and latency). YOSO that searches in the combined search space have achieved 1.42x~2.29x energy reduce or 1.79x~3.07x latency reduce at the same level of precision, under the optimization constraint of energy and latency, respectively. Overall, compared to the two-stage method, YOSO is able to deliver better DNN with more effective accelerator configuration.

Table 2. Performance Comparison

Model	Search Time (GPU*Day)	Test Error	Energy cost(mJ)	Latency (ms)	Configuration (PEs/g_buf/r_buf/data_flow)
NasNet-A[7]	1800	3.41	15.24	2.11	16*32/196KB/256b/OS
Darts_v1[5]	0.38	3.0	10.63	1.38	16*32/512Kb/512b/OS
Darts_v2[5]	1	**2.82**	11.01	1.62	14*16/256Kb/128b/OS
AmoebaNet-A[8]	3150	3.12	13.67	1.76	16*32/108Kb/1024b/OS
EnasNet[6]	1	2.89	16.65	2.25	16*32/196Kb/128b/OS
PnasNet[9]	150	3.63	17.17	2.37	16*20/512Kb/256b/OS
Yoso_lat	0.5	3.18	8.16	**0.77**	16*32/512Kb/512b/OS
Yoso_eer	0.5	3.05	**7.50**	0.97	16*32/512kb/128b/OS

*The search time of these two-stage models we choose does not include hardware search time.

IV. Conclusion

We present a fast automation framework that directly searches in the "2-dimensional" design space where any combination of software and hardware choices. This framework adopts some approaches that significantly improve search efficiency with fast performance evaluation, and the whole search is finished within 12 hours on single-card GPU. Compared with the two-stage method for image classification using Cifar10 targeting on systolic array based accelerator, we have achieved 1.42x~2.29x energy reduce or 1.79x~3.07x latency reduce at the same level of precision, for energy and latency optimization constraints, respectively.

References

[1] Cong Hao, Xiaofan Zhang, et al. "FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge." In DAC, 2019.

[2] Kwon Kiseok et al. "Co-Design of Deep Neural Nets and Neural Net Accelerators for Embedded Vision Applications." In DAC, 2018.

[3] Yifan Yang, QJ Huang, et al. "Synetgy: Algorithm-hardware Co-design for ConvNet Accelerators on Embedded FPGAs." In FPGA, 2019.

[4] Ye Yu, Yingmin Li, et al. "Software-Defined Design Space Exploration for an Efficient AI Accelerator Architecture" ArXiv abs/1903.07676, 2019.

[5] Hanxiao Liu et al. "DARTS: Differentiable architecture search." In ICLR, 2019.

[6] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le et al. "Efficient neural architecture search via parameters sharing." In ICML, 2018.

[7] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, et al. "Learning transferable architectures for scalable image recognition." In CVPR, 2018.

[8] Esteban Real, Alok Aggarwal, Yanping Huang et al. "Regularized evolution for image classifier architecture search." In AAAI. 2091.

[9] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, et al. "Progressive neural architecture search." In ECCV, 2018.