Algorithm and Hardware Co-design for Reconfigurable CNN Accelerator

Hongxiang Fan*, Martin Ferianc†, Zhiqiang Que*, He Li‖, Shuanglong Liu‡, Xinyu Niu§, Wayne Luk*
* Dept. of Computing, School of Engineering, Imperial College London, UK
{*h.fan17, z.que, w.luk*}*@imperial.ac.uk*
† Dept. of Electronic and Electrical Engineering, University College London, UK, *martin.ferianc.19@ucl.ac.uk*
‖ Dept. of Engineering, University of Cambridge, Cambridge, UK, *he.li@ieee.org*
‡ Hunan Normal University, Changsha, China, *s.liu13@imperial.ac.uk*
§ Corerain Technologies Ltd., Shenzhen, China, *xinyu.niu@corerain.com*

***Abstract*—Recent advances in algorithm-hardware co-design for deep neural networks (DNNs) have demonstrated their potential in automatically designing neural architectures and hardware designs. Nevertheless, it is still a challenging optimization problem due to the expensive training cost and the time-consuming hardware implementation, which makes the exploration on the vast design space of neural architecture and hardware design intractable. Different with previous co-design methods for DNNs, our proposed approach is capable of locating designs on the Pareto frontier. This capability is enabled by a novel three-phase co-design framework, with the following new features: (a) decoupling DNN training from the design space exploration of hardware architecture and neural architecture, (b) providing a hardware-friendly neural architecture space by considering hardware characteristics in constructing the search cells, (c) adopting Gaussian process to predict accuracy, latency and power consumption to avoid time-consuming synthesis and place-and-route processes. In comparison with the manually-designed ResNet101, InceptionV2 and MobileNetV2, we can achieve up to 5% higher accuracy with up to 3× speed up on the ImageNet dataset. Compared with other state-of-the-art co-design frameworks, our found network and hardware configuration can achieve 2% ∼ 6% higher accuracy, 2×∼ 26× smaller latency and 8.5× higher energy efficiency.**

I. Introduction

The success of deep learning, and especially neural networks (NNs), has attracted enormous research and industrial interests in applying NNs in real-life scenarios such as in autonomous driving [1]. However, the heavy computational and memory demand of running NNs imposes a large overhead on their hardware performance, in particular while considering resource-constrained platforms [2]. Currently, there are two research directions that focus on improving the hardware performance of deployed NNs. First, algorithm-level design of efficient NNs through neural architecture search (NAS) [3], which automatically designs NN architectures with high accuracy and low computational complexity for different scenarios [4]. Second, hardware-level efforts to design highly-optimized and specialized hardware accelerators for NNs, such as *Eyeriss* [5]. However, most of the time, the algorithm-level optimization and the hardware-level design are not considered jointly, which can lead to sub-optimal solutions in terms of both the resultant algorithmic or the hardware performance. For example, the authors in [2] demonstrate that the hardware architecture designed for the regular convolution is not suitable for depthwise convolution commonly used in the NAS [4].

To address the aforementioned sub-optimality, there is a growing demand for a method that can perform NAS to design accurate NNs and at the same time, co-develop hardware designs customized for the NN found. To meet this demand, reconfigurable hardware, such as field-programmable gate arrays (FPGAs), represents an ideal platform to implement algorithm-hardware co-design. Given its reconfigurability, FPGA can be utilized to provide highly-optimized hardware, customized for different NNs found by NAS. Previous work has attempted to apply evolutionary algorithm [6]–[8], reinforcement learning [9], [10] and differentiable NAS [11], [12] on algorithm-hardware co-design for DNNs on FPGA. However, these approaches iteratively perform the network training and design space exploration for multiple times, making the process time-consuming. Also, the characteristics of the accelerator are only considered during NAS in their work. To address these issues, our contributions include:

- A novel three-phase co-design framework, which decouples network training from design space exploration of both hardware design and neural architecture to avoid iterative time-consuming optimization. A hardware-friendly neural architecture space is also proposed by considering the characteristics of the underlying hardware to construct search cells before the neural architecture searching (Section III);
- An accurate and efficient cross-entropy loss, latency and energy consumption model based on Gaussian process regression, together with a genetic algorithm, which enable fast design space exploration within few minutes (Section IV);
- A demonstration of the effectiveness of the proposed method on the ImageNet dataset. The network found and its custom hardware design lie in the Pareto frontier, and can achieve better accuracy, energy efficiency and latency in comparison to other state-of-the-art co-design methods (Section V).

II. Background

A. Algorithm and Hardware Co-design

The joint design of NNs and hardware has been recently an activate research area [13]. Based on an evolutionary algorithm

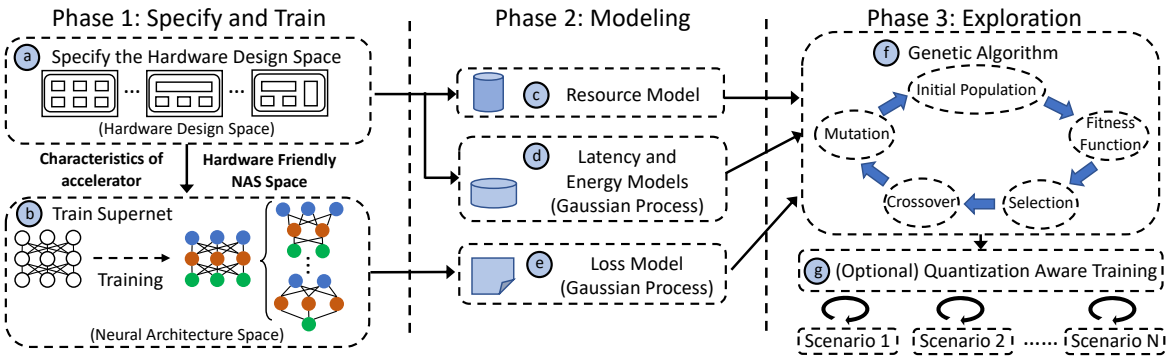

Fig. 1. The overview of the proposed framework.

(EA), Lin *et al.* [6] propose a two-stage method for algorithm-hardware co-design. Although their work claims 1.3× speedup and 1.6× energy savings, their results are based entirely on simulations, and the performance is estimated without running on a real hardware. At the same time, EA has been adopted in other NAS methods [7], [8]. However, training cost is expensive and the generated NNs lack in accuracy.

Reinforcement learning (RL) is another approach used for the algorithm-hardware co-design [9]. Nevertheless, a common drawback in these RL-based algorithm-hardware co-design approaches is that they demand significant number of GPU hours for search of both algorithm and hardware-defining parameters, which is unfeasible for real-life applications. To reduce the search cost, differentiable NAS (DNA) has been used in algorithm and hardware co-design [11]. However, it has been demonstrated in [14] that the NNs found by DNA can only achieve similar accuracy to the NNs generated by random search.

The once-for-all (*OFA*) proposed in [15] provides another paradigm for NAS. The progressive shrinking algorithm has been demonstrated to be effective in training the supernet. However, their work only optimizes neural architectures without searching the optimal hardware architectures. Also the neural architecture space in their paper does not consider the characteristics of underlying hardware design, which leads to sub-optimal hardware performance. Compared with their work, we are able to achieve a higher accuracy and hardware performance as demonstrated in Section V. Although [16] tries to search the accelerator architecture, they only focus on processing engine (PE) connectivity and compiler mappings for an ASIC design.

B. Gaussian Process

Gaussian process (GP) is a model built around Bayesian probabilistic theory which can embody prior knowledge into the predictive model and can be used for regression of real valued non-linear targets [17]. A GP is specified by a mean function and a covariance function-kernel. A common choice for kernels includes polynomial, Gaussian or Matérn kernels [17]. The mean function represents the supposed average of the estimated data. The kernel computes correlations between inputs and it encapsulates the structure of the hypothesised function. GP allows fast estimation, which is especially useful in design space exploration [18]. However, [18] only explored latency estimation for a single layer, while this paper adopts GP to estimate the loss, latency and energy consumption of the whole NN.

III. Algorithm-Hardware Co-design

The problem of algorithm-hardware co-design can be defined as follows:

$$\min_{\alpha\in\mathcal{A},\beta\in\mathcal{B}} \min_{\boldsymbol{w}_\alpha} \mathcal{L}\left(\boldsymbol{w}_\alpha, \alpha, \beta\right) \tag{1}$$

The $\mathcal{A}$ denotes the NN architecture space and $\mathcal{B}$ represents the hardware design space. To minimize the loss $\mathcal{L}$, we aim to find the optimal hardware configuration $\beta \in \mathcal{B}$ and NN architecture $\alpha \in \mathcal{A}$ with the associated weights $\boldsymbol{w}_\alpha$.

In this paper, we decouple the training of weights $\boldsymbol{w}_\alpha$ and the optimization of α and β into two separate steps. First, we train a *supernet*, encompassing all our NN architecture options, with respect to the weights $\boldsymbol{w}_\alpha$ using the following objective function based on a cross-entropy (CE) loss:

$$\min_{\boldsymbol{w}_\alpha} \sum_{\alpha\in\mathcal{A}} CE\left(\boldsymbol{w}_\alpha, \alpha\right), \tag{2}$$

During this process, we randomly sample sub-NNs from the supernet and independently train each sampled network to minimize the overall loss. Then, once the training is finished, we perform the optimization with respect to the α and β using the overall objective function $\mathcal{L}$ containing both the CE loss and hardware costs as follows:

$$\min_{\alpha\in\mathcal{A},\beta\in\mathcal{B}} \mathcal{L}\left(\alpha, \beta\right). \tag{3}$$

Aiming at solving (2) and (3), a novel algorithm-hardware co-design framework is proposed, which is illustrated in Figure 1. To make sure the framework is applicable to any reconfigurable hardware system, we generalize it into three phases: *1)* Specify and Train, *2)* Modeling and *3)* Exploration. Note that the first and second phases are only required once, while the Exploration is briefly performed given a specific deployment scenarios, which makes our framework efficient.

Phase 1: Specify and Train — To define the hardware design space for exploration, it first requires the users to specify a reconfigurable hardware system to accelerate NNs. Then, the neural architecture search space is built based on the supported

operations provided by the underlying hardware system. The neural architecture search space often considers different algorithmic configurations, ordering and connections between operations inside the NNs [3]. Note that, our framework does not apply any restrictions on the neural architecture space, and it can be changed accordingly for different reconfigurable hardware designs. Therefore, our framework is general enough to cover any reconfigurable hardware, and has potential to gain higher accuracy and hardware performance.

During the training, in order to efficiently solve (2), we use the progressive shrinking algorithm [15] to train all the sub-networks within the supernet by random sampling of candidate NNs. All the sub-networks share the same set of parameters within the supernet. Once the training is finished, we can quickly sample a sub-network from the supernet without extra effort. All these sub-networks form the final neural architecture space, which enables exploration at the algorithm-level in later phase.

Phase 2: Modeling — In the second phase, we model different metrics: CE loss (CE), latency, energy and resource consumption, to enable fast exploration in the last phase.

For loss, latency and energy models, we adopt the GP regression for fast estimation. The training data used for GP regression is obtained by randomly sampling a small number of sub-networks from the supernet. These sampled NNs are then evaluated on the dataset to get the CE loss, and run on our reconfigurable hardware with different configurations to obtain their latency and energy consumption. For the resource model, we propose to use a simple analytic formulation to estimate the DSP and memory resource consumption.

Phase 3: Exploration — In the last phase, as the regression models for loss, latency, energy and resources are available, Genetic algorithm (GA) is adopted for fast design space exploration in both neural architecture and hardware design search spaces. The loss (fitness) function is defined as follows:

$$\mathcal{L} = \eta \times CE + \mu \times Latency + \lambda \times Energy + Res_{PT}. \quad (4)$$

The η, μ and λ are user-defined hyper-parameters which denote the importance for the CE loss, latency and energy consumption. The CE, $Latency$ and $Energy$ are the regression results of GP-based CE loss, latency and energy models. Different values for η, μ and λ may lead to different results and this is explored in Section V. The Res_{PT} term is defined as follows:

$$Res_{PT} = \begin{cases} 0, & DSP_{used} \leq DSP_{avl}, MEM_{used} \leq MEM_{avl} \\ \gamma, & DSP_{used} > DSP_{avl}, MEM_{used} > MEM_{avl} \end{cases} \quad (5)$$

where the DSP_{avl} and MEM_{avl} represent the available DSPs and memory resources on the target hardware platform and DSP_{used} and MEM_{used} denote DSP and memory consumption provided by the resource model. The γ denotes the penalty added to the loss function when the hardware resource consumption exceeds the budget. In our very last step, if the underlying hardware supports different precision other than the one used for training the supernet, the quantization aware finetuning [19] will be enabled to tailor the resultant NNs to the hardware system. The process of GA is illustrated in Figure 1.

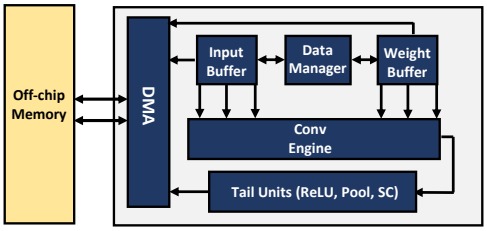

Fig. 2. Overview of the FPGA-based accelerator.

IV. Design Space and Modelling

A. Design Space

The design space is composed of two parts: hardware design space and neural architecture space.

1) Hardware Design Space: This paper adopts an example design which uses a single configurable processing unit to process different layers. Although there are other designs, such as the streaming design [11], [13] with layer-wise reconfigurability, they usually require a large amount of on-chip memory to cache all the intermediate results, which restricts the model size of the NN and limits the neural architecture space. In this paper, we adopted the single processing engine design, such that our search space encompasses larger CNNs. Note that, although we use the single engine design, our framework is general enough to be applied to any reconfigurable design such as the streaming design by changing the hardware design space.

The adopted reconfigurable design is illustrated in Figure 2. The accelerator consists of an input buffer, a weight buffer, a convolutional (*Conv*) engine and other functional modules including Shortcut (*SC*) [20], Pooling (*Pool*) and Rectified linear unit (*ReLU*) activation. The computation of the NN is performed sequentially, layer-by-layer, and only one layer is processed in the *Conv* engine at a time. This computational pattern allows the accelerator to support NNs even with a large number of layers because only one layer's input data and weights need to be cached in the on-chip memory. To achieve higher hardware performance, the accelerator is designed to support 8-bit integer operations.

The *Conv* engine supports three types of configurable parallelism: filter parallelism (PF), channel parallelism (PC) and vector parallelism (PV). Different types of convolutions require different combinations of PF, PC and PV to achieve an optimal performance. For instance, a convolution with small number of channels can achieve lower latency with the combination of low PC and high PF and PV values, since there is no available concurrency in the channel dimension. Our hardware design space is represented by memory size MEM, bandwidth BW and different parallelism levels including PF, PC and PV. The domain for both PF and PC is $\{8, 16, 32, 64, 128\}$ and PV can be chosen from $\{4, 8, 16\}$. MEM depends on the available memory resources on the

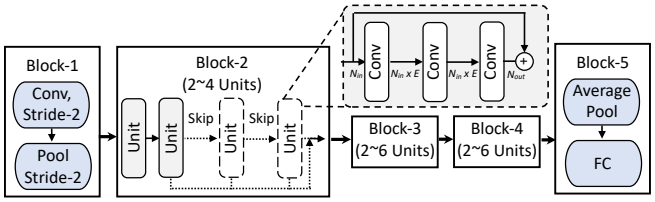

Fig. 3. The search space of neural architectures.

FPGA board and BW is selected from $\{32, 64, 128, 256\}$bits. Thus, there are totally $5 \times 5 \times 3 \times 4$ potential different configurations for the hardware design.

2) Neural Architecture Space: The example architecture space is illustrated in Figure 3. In this paper, we argue that the design of neural architecture search should consider the underlying hardware design before the NAS optimization. By analyzing the characteristics of the selected hardware architecture, we found that it is efficient in performing the regular convolution with residual addition [20]. Also, [21] demonstrate that the basic building block of *ResNet* is still one of the most effective architectures with the proper scaling strategies. Therefore, our core neural architecture search space follows the backbone of *ResNet-50* which is composed of four residual blocks with gradually reduced feature map size and increased channel sizes. In each block, we search for the number of units ranging from 2 to U_i, where U_i denotes the maximal number of units in i^{th} block. In each cell, we search for the expansion ratio (E) chosen from $\{0.5, 0.75, 1.0\}$. As there are totally 16 cells in our neural architecture space, the total number of combinations is 3^{16}. Together with the 300 different hardware configuration options, there are more than 12×10^9 different combinations in our co-design space.

B. Loss, Latency, Energy and Resource Models

1) Loss Model: Evaluating CE in Equation 2 for all 12 billion configurations on a large dataset such as ImageNet [22] is time-consuming. To enable fast evaluation, we adopt GP regression to estimate the CE for all sub-networks. To represent the neural architecture, we encode the neural architecture space, which contains 16 searchable cells, into a 16-dimension vector with each dimension representing the expansion ratio used in that cell. The expansion ratio is 0, if a cell is skipped. We construct a training dataset by randomly sampling and evaluating a certain number of sub-networks. Based on the encoded input vector and evaluated CE, we perform regression using the GP model with a Matérn covariance kernel with a constant mean function.

2) Latency and Energy Models: Measuring the hardware performance of all sub-networks for the FPGA-based design for different design parameters is time-consuming because of synthesis and place and route processes that are needed for the real hardware implementation. We again use GP regression model to estimate the latency and energy consumption. To represent the NN together with the hardware configuration, we encode it into a 19-dimensional vector with the first 16 dimensions representing the neural architecture and the last 3 dimensions being PF, PC and PV.

3) Resource Model: As DSPs and memory are the limiting resource for FPGA-based CNN accelerator [23], we primarily consider DSP and memory consumption in this paper. The DSP consumption can be described as: $DSP_{used} = (PC \times PF \times PV)/2$, which is dominated by the parallelism level used in the *Conv* engine.

The memory resources are mainly consumed by the input and weight buffers. As the input buffer needs to cache all the input feature maps in the current i^{th} layer, its usage can be represented as: $MEM_{in} = \max_{i=1,\ldots,l}(N_c^i \times H^i \times W^i) \times DW$, where N_c^i, H^i and W^i denote the number of channels, height and width of the input feature map, DW is the data width and l is the total depth of the net. As for the weight buffer, because weights are shared along PV parallelism, it only needs to cache the current PF filters, so the memory consumption can be formulated as: $MEM_{weight} = \max_{i=1,\ldots,l}(N_c^i \times PF \times K^i \times K^i) \times DW$, where K^i is the kernel size of the i^{th} layer. Due to the use of ping-pong buffer technique, the total memory consumption is: $MEM_{used} = 2 \times (MEM_{in} + MEM_{weight})$.

V. Experiments

The PyTorch and GPyTorch libraries are used for the implementation of the supernet training and the GP models respectively. ImageNet [22] dataset contains over 10,000,000 labeled images of 1000 object categories for classification. The hardware design used in all experiments is implemented on an Intel Arria 10 SX660 FPGA platform using Verilog. 1GB DDR4 SDRAM is installed on the platform as the off-chip memory. Quartus 17 Prime Pro was used for synthesis and implementation. An Intel Xeon E5-2680 v2 CPU was used as the host processor. We train the supernet on a GPU cluster with six NVIDIA GTX 1080 Ti GPUs for 4 days. A power meter is plugged in to measure the runtime power performance.

A. Accuracy of Gaussian Process-based Model

To train our GP-based loss model, 2000 sub-networks were sampled and evaluated on ImageNet [22]. We used 1500 samples for training and 500 samples for evaluation. The model was trained for 50 iterations using an Adam optimizer. The result is shown in Table I. The mean absolute error (MAE) is only 0.01005, which demonstrates the GP-based loss model is sufficiently accurate for the modeling.

TABLE I
Results of Gaussian process-based models.

	Kernel Function	Mean Absolute Error
Loss Model	Matérn (3/2)	0.01005
Latency Model	Matérn (5/2)	0.06521ms
Energy Model	Matérn (5/2)	0.01804W

Similarly, 4600 random samples with different network configurations and hardware designs were collected for latency and energy modeling. We used 3000 and 1600 samples for training and evaluation respectively. The training was again

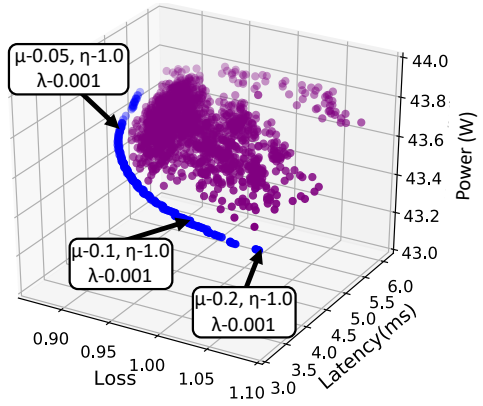

Fig. 4. The performance of various NAS-generated NNs on different candidate hardware design. Pareto-optimal is denoted by blue points.

performed with respect to 50 iterations and an Adam optimizer. As shown in Table I, the MAE of our GP-based latency and energy models is only 0.06521ms and 0.01804W. Therefore, the proposed GP-based latency and energy model can be used as an accurate estimator for the latency and energy consumption.

B. Effectiveness of Design Space Exploration

For reference and demonstration, we iterated through and evaluated all samples in the co-design space to get the reference Pareto frontier. The Pareto-optimal points, which are better in either loss or latency or energy with respect to any other point, form a Pareto frontier, which is drawn as blue points in Figure 4. Because the whole design space is too large to show in the Figure. we randomly drew 2000 non-Pareto-optimal samples as purple points to visualize the rest of the design space.

Then, to demonstrate the effectiveness of our framework, we used GA to perform design space exploration, and check whether these found configurations match the reference Pareto frontier. The time cost for the proposed GP-based models and GA to find one optimized design is only 0.1 GPU hour, which demonstrates the efficiency of our framework. In contrast, other approaches [9], [24] require tens to hundreds of GPU hours in searching. As mentioned in Section III, the user-defined hyper-parameters η, μ and λ specified in the GA represent the importance of accuracy, latency and energy consumption respectively, we chose three sets of η, μ and λ: $\{1.0, 0.2, 0.001\}$, $\{1.0, 0.1, 0.001\}$ and $\{1.0, 0.05, 0.001\}$, to demonstrate how the GA is able to find different Pareto-optimal designs according to users' requirements. The resultant designs found by the GA are highlighted by black arrows in Figure 4, which all lay on the reference Pareto frontier. Their NN architectures and hardware configurations are illustrated in Figure 5. Therefore, our framework can effectively identify the Pareto-optimal designs in the vast algorithm-hardware co-design space.

We also evaluated the resultant networks on different hardware platforms including Intel Xeon Silver 4110 CPU and NVIDIA GTX 1080 Ti GPU. The results are presented in Table II. TensorRT and CuDNN 8.11 libraries were used for GPU

TABLE II
ACCURACY, LATENCY AND ENERGY EFFICIENCY ON IMAGENET.

	CPU		GPU		FPGA		Acc
	Lat. (ms)	Enrg. Eff. (FPS/W)	Lat. (ms)	Enrg. Eff. (FPS/W)	Lat. (ms)	Enrg. Eff. (FPS/W)	
$\eta(1.0)\mu(0.05)$ $\lambda(0.001)$	26.08	0.28	7.40	0.94	4.52	5.07	**77.63%**
$\eta(1.0)\mu(0.1)$ $\lambda(0.001)$	24.06	0.30	6.57	1.06	3.66	6.27	76.30%
$\eta(1.0)\mu(0.2)$ $\lambda(0.001)$	19.18	0.38	5.03	1.38	**3.14**	**7.32**	74.91%

implementation, and the MKLDNN was used to optimize the performance of the CPU implementation. The batch size was set to one for a fair comparison. Compared with GPU and CPU implementations, the networks found for the reconfigurable FPGA-based accelerator can achieve approximately $2\times$ and $6\times$ reduction in latency and up to $5\times$ and $19\times$ higher energy efficiency.

C. Comparison with Manually Designed Networks

To demonstrate that the auto-generated NN architectures can outperform manually-designed networks in terms of accuracy, latency, energy and model size on our FPGA accelerator, we evaluated several commonly benchmarked NNs including *ResNet-101* [20], *VGG-16* [25] and *Inception-v2* [26] on the ImageNet. The hardware configurations with respect to these networks were manually optimized. The results are shown in Figure 6. The network found with highest accuracy ($\eta = 1.0$, $\mu = 0.05$, $\lambda = 0.001$) is nearly 1% more accurate and $3\times$ faster than *ResNet-101*. Compared with *VGG-16*, the network found can achieve nearly 5% higher accuracy while reducing the latency by nearly $10\times$. We also compared our work with the *MobileNetV2* [27] implemented in [15]. Our design achieves a similar latency while improving the accuracy by nearly 4%.

D. Comparison with Existing Co-Design Work

We compared our proposed approach with four other state-of-the-art co-design methods, including *Co-Explore* [28], *EDD* [11], *HAO* [24], and *OFA* [15]. Although there are other

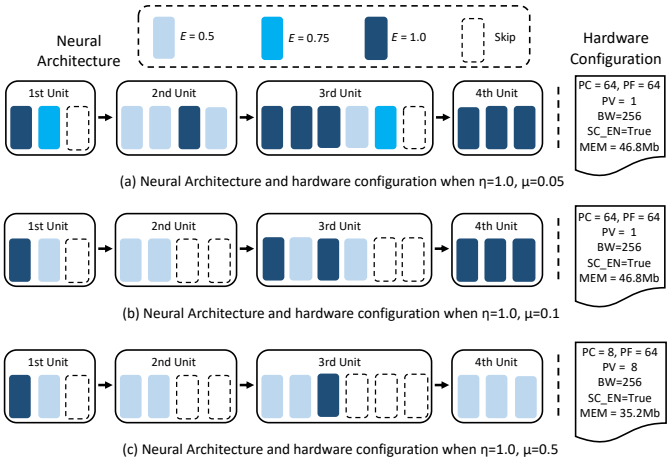

Fig. 5. Neural architecture and hardware configuration of NNs found.

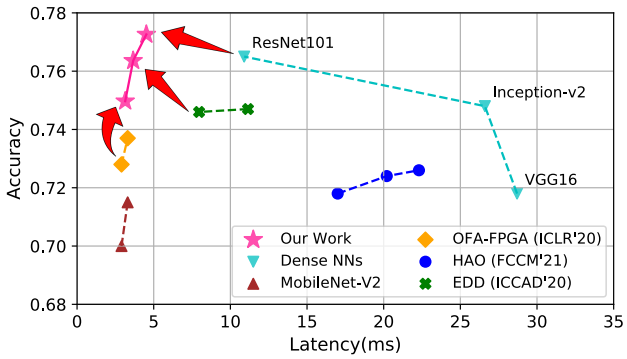

Fig. 6. Comparison of accuracy and latency among our work, the manually-designed neural networks and other algorithm-hardware co-design methods.

TABLE III
DETAILS OF HARDWARE IMPLEMENTATIONS.

	Platform	Number of DSPs	Latency (ms)	Accuracy	Energy Eff. (GOPS/W)
Co-Explore [9]	Xilinx XC7Z015	150	95.24	70.24%	0.74
EDD [11]	Xilinx ZCU102	2520	7.96	74.60%	-
HAO [24]	Xilinx ZU3EG	360	22.27	72.68%	-
OFA [15]	Xilinx ZU9EG	2520	3.30	73.60%	-
Our Work	Intel GX1150	1345	3.66	76.30%	6.27

co-design works, they suffer from low accuracy [7], [8], [13], [29] or only evaluated on a small dataset [10], [12], [30]. Therefore, we did not include them in our comparison. The results are shown in Figure 6. Table III summarizes their underlying hardware platforms and implementation details. Compared with the network generated by [28], our network found can achieve 6% higher accuracy, more than 26× speed up and nearly 8× higher energy efficiency. We can also achieve nearly 4% higher accuracy than *HAO* [24] with better hardware performance even with the latency being normalized by the DSP consumption. In comparison with *OFA* that consumes nearly twice more DSPs, we achieve a similar latency with 2.7% higher accuracy.

VI. CONCLUSION

This paper proposes a novel algorithm-hardware co-design framework for reconfigurable NN accelerators. To reduce the search cost, we adopt genetic algorithm and Gaussian process regression, which enables fast design space exploration within few minutes. The network and hardware configuration generated by the proposed framework on our reconfigurable CNN accelerator can achieve 1% to 5% higher accuracy while reducing the latency by 2× to 10× on the ImageNet dataset, in comparison with manually-designed NNs on the same hardware. Compared with the other state-of-the-art algorithm-hardware co-design approaches, our found NNs achieve better accuracy, energy efficiency, latency and search cost. Future work includes expanding the search space with more choices of operations, integrating optimization for recurrent neural networks into the current optimization step and supporting end-to-end automation.

ACKNOWLEDGEMENT

The support of the United Kingdom EPSRC (No. EP/L016796/1, EP/N031768/1, EP/P010040/1, EP/V028251/1 and EP/S030069/1), the National Natural Science Foundation of China (No. 62001165), Hunan Provincial Natural Science Foundation of China (No. 2021JJ40357), Changsha Municipal Natural Science Foundation (No. kq2014079), Corerain, Maxeler, Intel and Xilinx is gratefully acknowledged.

REFERENCES

[1] S. Grigorescu *et al.*, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.
[2] H. Fan *et al.*, "A real-time object detection accelerator with compressed SSDLite on FPGA," in *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 14–21, IEEE, 2018.
[3] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.
[4] B. Wu *et al.*, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.
[5] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.
[6] Y. Lin *et al.*, "Neural-hardware architecture search," *Workshop on ML for Systems at NeurIPS 2020*, 2019.
[7] P. Colangelo *et al.*, "Artificial neural network and accelerator co-design using evolutionary algorithms," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2019.
[8] P. Colangelo *et al.*, "Evolutionary cell aided design for neural network architectures," *arXiv preprint arXiv:1903.02130*, 2019.
[9] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4805–4815, 2020.
[10] M. S. Abdelfattah *et al.*, "Best of both worlds: Automl codesign of a CNN and its hardware accelerator," *arXiv preprint arXiv:2002.05022*, 2020.
[11] Y. Li *et al.*, "EDD: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," *arXiv preprint arXiv:2005.02563*, 2020.
[12] H. Fan *et al.*, "Optimizing fpga-based cnn accelerator using differentiable neural architecture search," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 465–468, IEEE, 2020.
[13] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for 1ot intelligence on the edge," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.
[14] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in Artificial Intelligence*, pp. 367–377, PMLR, 2020.
[15] H. Cai *et al.*, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.
[16] Y. Lin *et al.*, "Naas: Neural accelerator architecture search," *arXiv preprint arXiv:2105.13258*, 2021.
[17] C. E. Rasmussen and H. Nickisch, "Gaussian processes for machine learning (gpml) toolbox," *The Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, 2010.
[18] M. Ferianc *et al.*, "Improving performance estimation for FPGA-based accelerators for convolutional neural networks," in *International Symposium on Applied Reconfigurable Computing*, pp. 3–13, Springer, 2020.
[19] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.
[20] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[21] I. Bello *et al.*, "Revisiting resnets: Improved training and scaling strategies," *arXiv preprint arXiv:2103.07579*, 2021.

[22] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.
[23] S. Liu *et al.*, "Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–22, 2018.
[24] Z. Dong *et al.*, "Hao: Hardware-aware neural architecture optimization for efficient inference," in *IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 50–59, IEEE, 2021.
[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[26] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.
[27] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.
[28] W. Jiang *et al.*, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4154–4165, 2020.
[29] W. Jiang *et al.*, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.
[30] W. Chen *et al.*, "You only search once: a fast automation framework for single-stage dnn/accelerator co-design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1283–1286, IEEE, 2020.