Optimizing FPGA-Based CNN Accelerator Using Differentiable Neural Architecture Search

Hongxiang Fan*, Martin Ferianc†, Shuanglong Liu‡, Zhiqiang Que*, 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*
‡ Hunan Normal University, Changsha, China, *s.liu13@imperial.ac.uk*
§ Corerain Technologies Ltd., Shenzhen, China, *xinyu.niu@corerain.com*

***Abstract*—Neural architecture search (NAS) aims to find the optimal neural network automatically for different scenarios. Among various NAS methods, the differentiable NAS (DNAS) approach has demonstrated its effectiveness in terms of searching cost and final accuracy. However, most of previous efforts focus on applying DNAS to GPU or CPU platforms, and its potential is less exploited on the FPGA. In this paper, we first propose a novel FPGA-based CNN accelerator. An accurate performance model of the proposed hardware design is also introduced. To improve accuracy as well as hardware performance, we then apply DNAS and encapsulate the proposed performance model into the objective function. Based on our FPGA design and NAS method, the experiments demonstrate that the network generated by NAS achieves nearly 95% accuracy on CIFAR-10, while decreasing latency by nearly 12 times compared with existing work.**

***Index Terms*—Neural Architecture Search (NAS), FPGA**

I. Introduction

The potential of neural architecture search (NAS) in artificial intelligence (AI) tasks such as image classification, object detection or speech recognition has been recognised [1], [2]. Compared to manual design methods for neural networks (NNs), current NAS methods are able to automatically find optimised neural architectures by improving their software-related hyperparameters such as the number of layers, the number of channels and topological connections [3], [4]. Among various NAS methods, differentiable neural architecture search (DNAS), which makes use of gradient values to optimize neural networks, has been shown to be effective in finding NNs with improved accuracy. However, most existing DNAS methods only focus on searching neural networks for GPU or CPU platforms.

Field-programmable gate arrays (FPGAs), due to their reconfigurability and high energy efficiency, are becoming increasingly popular in the accelerating convolutional neural networks (CNNs) [5]–[8]. This paper aims to exploit the potential of DNAS to improve the performance of FPGA-based CNN accelerators.

The main contributions of this work are the following:

- An FPGA-based CNN accelerator built on the single processing engine design approach, which enables the mapping of large neural networks into a single FPGA device (Section II);
- Adopting DNAS with an accurate performance model for the proposed FPGA-based CNN accelerator, which achieves high accuracy as well as high hardware performance on the CIFAR-10 dataset. (Section III).

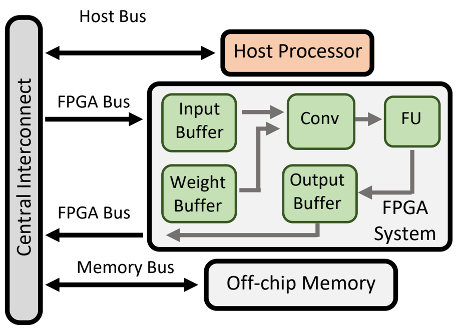


Fig. 1. System overview. FU means functional unit.

II. The Hardware Accelerator

A. FPGA-Based CNN Accelerator

The system overview of the proposed FPGA-based CNN accelerator is presented in Figure 1. It mainly consists of an FPGA system, an off-chip memory and a host processor.

The FPGA system is designed to perform all the computations of CNNs, which are mainly composed of convolutional (*CONV*) module, functional unit (*FU*), weight buffer, output buffer and input buffer. The *CONV* module is used to perform convolution. It supports three types of parallelism: filter parallelism (PF), channel parallelism (PC) and vector parallelism (PV). The FU supports operations including batch normalization, rectified linear unit (*ReLU*) activation, pooling and residual shortcut (*SC*). The computation of each convolutional layer starts from loading the weights and input data from the off-chip memory to the on-chip weight buffer and input buffer respectively via DMA. The data then flow through the *CONV* and *FU* modules sequentially to perform the main calculations. Note that only the necessary operations will be enabled in FU. The outputs of the current layer generated from the FU are then streamed back to the off-chip memory for the computation of the next layer. The double buffer technique is used in both the

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00085

TABLE I
PARAMETERS' NOTATION USED WITH RESPECT TO THE NEURAL NETWORK AND HARDWARE PARAMETERS.

Parameter	Description
EFF_{io}	Efficiency of IO
CLK_{io}	Clock frequency of IO
CLK_{pe}	Clock frequency of processing engine
H_{in}	The height of input feature map
W_{in}	The width of input feature map
H_{out}	The height of output feature map
W_{out}	The width of output feature map
PF	Filter parallelism
PC	Channel parallelism
PV	Vector parallelism
BW	Bandwidth between off-chip and on-chip memory
N_c	The number of channels
N_f	The number of filters
G	The number of groups in group convolution
K	The kernel size of convolution

input and weight buffers, to enable an overlap of the input and weight transfer times, thus decreasing the overall latency.

The host processor is used to set different hyperparameters for the FPGA system, such as the input image size and the number of channels, while running various CNN layers. The FPGA system adopts the APB bus. The peripheral component interconnectexpress 3.0 (PCIe3) is the host bus, and DDR4 is used in off-chip memory.

B. Performance Model

To speed-up our optimization process for searching the optimal NN architecture on the given hardware, we propose an accurate performance model to predict the hardware performance given an NN architecture.

Because the proposed CNN accelerator computes the network's output layer-by-layer, the total latency is given by:

$$T_{total} = \sum_{i=0}^{N} T_i, \tag{1}$$

where N is the number of layers in the network and T_i is the latency incurred by the ith layer. As channel-major computational pattern is adopted, the accelerator generates outputs in totally $BF = N_f/PF$ batches with each batch producing $PF \times H_{out} \times W_{out}$ output pixels. Therefore, we further divide T_i into:

$$T_i = \sum_{j=0}^{BF} T_{i,j}. \tag{2}$$

The per-batch latency $T_{i,j}$ in layer i is dominated by the following three parts: *(1)* $T_{i,j}^{load}$: time for loading input data and weights from off-chip memory into the input buffer and the weight buffer, *(2)* $T_{i,j}^{store}$: time for storing the outputs back to the off-chip memory, *(3)* $T_{i,j}^{compute}$: time for the computation of the jth batch in layer i. To approximate the estimated performance as close as possible to the real performance,

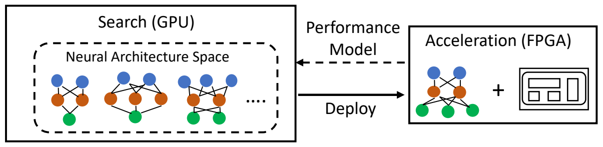


Fig. 2. An overview of DNAS for FPGA-based CNN accelerator.

we calculate the execution time in terms of clock cycles. Therefore, these three parts can be formulated as:

1) Loading time, which is composed of the input data loading time $T_{i,j}^{input}$ and the weights loading time $T_{i,j}^{weight}$:

$$T_{i,j}^{input} = \frac{N_{c_i} \times W_{in_i} \times H_{in_i}}{BW \times EFF_{io} \times CLK_{io}} \tag{3}$$

$$T_{i,j}^{weight} = \frac{K_i \times K_i \times N_{c_i} \times PF}{BW \times EFF_{io} \times CLK_{io}} \tag{4}$$

As the input data are cached in the input buffer for reuse, the input data loading time only happens during the initial batch, which can be formulated as:

$$T_{i,j}^{load} = \begin{cases} T_{j=1,i}^{load} & = T_{i,j}^{input} + T_{i,j}^{weight} \\ T_{j\neq 1,i}^{load} & = T_{i,j}^{weight} \end{cases} \tag{5}$$

2) Storing time associated with the transfer of the output data to the off-chip memory for the use of the next layer:

$$T_{i,j}^{store} = \frac{PF \times W_{out_i} \times H_{out_i}}{BW \times EFF_{io} \times CLK_{io}} \tag{6}$$

3) Computational time for the *PE* to process the data. Since it is primarily dominated by the convolution operation ($>$ 99% of time) and the other operations (*BN, Pool, ReLU, Pool or SC*) incur only a negligible cost, the computational time can be formulated as:

$$T_{i,j}^{compute} = \frac{K_i \times K_i \times N_{c_i} \times PF \times W_{out_i} \times H_{out_i}}{G \times PF \times PC \times CLK_{pe}} \tag{7}$$

In our accelerator, these three parts are overlapped between each other to achieve high performance. Therefore, the time spent on the ith layer can be formulated as:

$$T_i = \sum_{j=0}^{BF} max(T_{i,j}^{load}, T_{i,j}^{store}, T_{i,j}^{compute}). \tag{8}$$

III. HARDWARE-AWARE NEURAL ARCHITECTURE SEARCH

A. Overview

An overview of the proposed search method is presented in Figure 2, where the search is performed on a GPU and the neural network found is accelerated on an FPGA. This paper adopts the differentiable NAS approach [9], [10] which can be formulated as an optimization problem as follows:

$$\min_{a\in\mathcal{A}} \min_{\boldsymbol{w}_a} \mathcal{L}\left(a, \boldsymbol{w}_a\right) \tag{9}$$

Given the NN architecture design space $\mathcal{A}$, the optimization process aims to find the optimal NN architecture $a \in \mathcal{A}$

with the associated weights $\boldsymbol{w}_a$ to achieve the minimal loss $\mathcal{L}(a, \boldsymbol{w}_a)$. To enable the algorithm to select the operations with the highest hardware performance while maintaining high accuracy, we define the loss function as follows:

$$\mathcal{L}(a, \boldsymbol{w}_a) = CE(a, \boldsymbol{w}_a) + \gamma \times LT(a)^{\delta}, \qquad (10)$$

where $CE(a, \boldsymbol{w}_a)$ represents the main objective associated with the overall accuracy, e.g. the cross-entropy loss of architecture a with weights $\boldsymbol{w}_a$. The term $LT(a)$ denotes the latency of the hardware design with the selected neural architecture a. The coefficients γ and δ are used to control the magnitude of the latency term during the optimization. The performance model proposed in Section II-B is used to determine the term $LT(a)$, which can speed up the optimization process.

B. Search Space

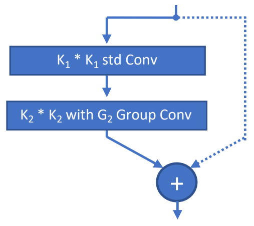
As other researchers have demonstrated [3], [11], the definition of the search space is vital for finding an NN capable of achieving high accuracy. In this paper, we adopt the NN design space used in [11]. The search space concentrates on searching architecture parameters in a layer-wise manner within a fixed micro-architecture. Table II summarizes the micro-architecture through the input shapes, available operation types, number of filters, number of repeated blocks and stride for convolutional operations. The first convolutional layer, the penultimate average pooling (*AvgPool*) and the last fully connected layers are fixed. In the middle of the NN, there are three stages and each stage contains nine searchable blocks. The details of a searchable block are illustrated in Figure 3. The block contains $K_1 \times K_1$ and point-wise (1×1) standard convolutions at the top and bottom, where K_1 is the kernel size of the first convolution. A K_2-by-K_2 group convolution with G_2 groups is in the middle, where K_2 denotes the kernel size for the group convolution. Therefore, the searchable layer-wise architecture parameters determining a include the values of K_1, K_2 and G_2. The search domain of K_1, K_2 and G_2 is presented in the table of Figure 3.

TABLE II
THE MICRO-ARCHITECTURE OF THE SEARCH SPACE.

Input Shape	Operation	Filter Number	Block Number	Stride
$32^2 \times 3$	3×3 conv	16	1	1
$32^2 \times 16$	AnyNet Block	32	9	1
$32^2 \times 32$	AnyNet Block	64	9	2
$16^2 \times 64$	AnyNet Block	128	9	2
$8^2 \times 128$	AvgPool	-	1	8
64	Fully Connect	10	1	-

IV. Experiments

The proposed hardware design is implemented on an Intel Arria 10 SX660 platform using Verilog. 1GB DDR4 SDRAM is installed on the platform as off-chip memory. Quartus 17 Prime Pro is used for synthesis and implementation. An Intel XeonE5-2680 v2 CPU is the host processor in our system. The parallelism levels PC, PV, PF are set to be 64, 1 and 64 respectively. The hardware is clocked at 220 MHz.



Search Space

Param	Domain
K_1	{1, 3}
K_2	{1, 3}
G_2	{1, 2, 4}

Fig. 3. The structure of a searchable block.

TABLE III
ERROR OF PERFORMANCE ESTIMATION FOR *ResNet–50*, *VGG–16* AND *Inception*

	Estimated Time (ms)	Actual Execution Time (ms)	Prediction Error
***ResNet–50* [12]**	16.69	17.21	3.10%
***VGG–16* [13]**	77.01	80.78	4.89%
***Inception* [14]**	70.21	71.13	1.31%

A. Accuracy of Performance Model

To evaluate the accuracy of the performance model (Section II-B), three different CNN models, including *ResNet–50* [12], *VGG–16* [13] and *Inception* [14] are used in our experiments. Table III presents the result of the three models in terms of the estimated time, the actual execution time and the prediction error of the performance model. It can be seen from Table III that the prediction error is within 5% for these three models. In particular, the prediction error for Inception [14] is less than 2These results show that the proposed performance model is sufficiently accurate for estimating the actual hardware performance, so it can be used to speed up the optimization process of DNAS.

B. Effectiveness of DNAS

To demonstrate the effectiveness of the proposed method, the CIFAR–10 [15] dataset is used in our experiments. The input resolution of the network is set to be to 32-by-32. During the search phase, the supernet is trained for 90 epochs, with the first 10 epochs for warm-up training [12]. In the loss function, the γ and δ are set to be 0.3 and 0.1 respectively. The SGD is used as our optimizer for training the supernet with a cosine schedule without restart [16]. The initial learning rate for weight and architecture parameters are set to be 0.1 and 0.01 respectively. The structure of the found network by the proposed method is illustrated in Figure 4, where k_1, k_2 and $g2$ denote the kernel of the first convolution, the kernel and group of the second convolution. To get the final accuracy, we train all such found NNs from scratch for 200 epochs using the same setting as the supernet training.

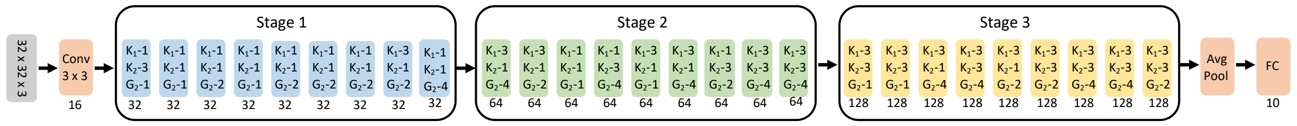

32 x 32 x 3
Conv 3 x 3
16
Stage 1
K_1-1 K_2-3 G_2-1 32
K_1-1 K_2-1 G_2-1 32
K_1-1 K_2-1 G_2-2 32
K_1-1 K_2-1 G_2-1 32
K_1-1 K_2-1 G_2-2 32
K_1-1 K_2-1 G_2-2 32
K_1-1 K_2-1 G_2-2 32
K_1-3 K_2-1 G_2-2 32
K_1-1 K_2-1 G_2-4 32
Stage 2
K_1-3 K_2-1 G_2-4 64
K_1-3 K_2-1 G_2-2 64
K_1-1 K_2-1 G_2-1 64
K_1-1 K_2-3 G_2-4 64
K_1-3 K_2-1 G_2-1 64
K_1-3 K_2-1 G_2-4 64
K_1-1 K_2-3 G_2-2 64
K_1-3 K_2-3 G_2-4 64
K_1-3 K_2-3 G_2-4 64
Stage 3
K_1-3 K_2-3 G_2-1 128
K_1-3 K_2-3 G_2-1 128
K_1-3 K_2-1 G_2-4 128
K_1-3 K_2-3 G_2-4 128
K_1-3 K_2-1 G_2-2 128
K_1-3 K_2-3 G_2-2 128
K_1-3 K_2-3 G_2-4 128
K_1-3 K_2-3 G_2-4 128
K_1-3 K_2-3 G_2-2 128
Avg Pool
FC
10

Fig. 4. The neural architecture of networks found for CIFAR–10 dataset.

TABLE IV
ACCURACY AND PERFORMANCE ON CIFAR-10 DATASET

Latency (ms)			**Accuracy**
CPU	GPU	FPGA	
54.11	12.68	**2.71**	94.18%

We evaluate the found networks on three different hardware platforms: Intel Xeon Silver4110 CPU, NVIDIA GTX 1080 Ti GPU, and Intel Arria 10 SX660 FPGA. The latency is the average value of the results of running the network 1000 times. The PyTorch framework is used for implementing designs for the CPU and GPU. The batch size is set to be 1 in all the hardware platforms for a fair comparison. The final result is presented in Table IV. Compared with GPU and CPU implementations, the networks found for FPGA can achieve nearly 5 times and 20 times speedup because of the proposed DNAS approach for NN hardware design.

TABLE V
COMPARISON BETWEEN THE PROPOSED METHOD AND EXISTING WORK ON CIFAR–10

	Latency (ms)	**Search Cost GPU Hours**	**Accuracy**
reinforcement learning based NAS [17]	33.67	108000	84.53%
RNAS (ours)	2.71	25.1	94.18%

Table V compares the proposed method with the existing reinforcement learning based hardware-aware NAS approach [17] in terms of latency, search cost (GPU hours) and accuracy on CIFAR–10. When compared with [17], the proposed method reduces search time from 108000 to 25.1 GPU hours while improving accuracy by 9.65% and reducing latency by nearly 12 times.

V. CONCLUSION

This work proposes a novel hardware architecture and adopts the hardware-aware differentiable neural architecture search (DNAS) to achieve high performance on an FPGA. Future work includes expanding the search space with more choices of operations, integrating optimization for recurrent neural networks into the current optimization step, and improving the search efficiency.

ACKNOWLEDGEMENT

The support of the China Scholarship Council, United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1), National Natural Science Foundation of China (Grant no. 62001165), Corerain, Maxeler, Intel and Xilinx is gratefully acknowledged.

REFERENCES

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.
[2] A. Baruwa, M. Abisiga, I. Gbadegesin, and A. Fakunle, "Leveraging end-to-end speech recognition with neural architecture search," *arXiv preprint arXiv:1912.05946*, 2019.
[3] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," *arXiv preprint arXiv:1902.07638*, 2019.
[4] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.
[5] H. Fan, S. Liu, M. Ferianc, H.-C. Ng, Z. Que, S. Liu, X. Niu, and W. Luk, "A real-time object detection accelerator with compressed SSDLite on FPGA," in *FPT*, pp. 14–21, IEEE, 2018.
[6] H. Fan, G. Wang, M. Ferianc, X. Niu, and W. Luk, "Static block floating-point quantization for convolutional neural networks on FPGA," in *FPT*, pp. 28–35, IEEE, 2019.
[7] H. Fan, C. Luo, C. Zeng, M. Ferianc, Z. Que, S. Liu, X. Niu, and W. Luk, "F-E3D: FPGA-based acceleration of an efficient 3D convolutional neural network for human action recognition," in *ASAP*, vol. 2160, pp. 1–8, IEEE, 2019.
[8] H. Fan, H.-C. Ng, S. Liu, Z. Que, X. Niu, and W. Luk, "Reconfigurable acceleration of 3D-CNNs for human action recognition with block floating-point representation," in *FPL*, pp. 287–2877, IEEE, 2018.
[9] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.
[10] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.
[11] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," *arXiv preprint arXiv:2003.13678*, 2020.
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[14] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
[15] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," *MSc thesis, University of Toronto*, 2009.
[16] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.
[17] W. Jiang, L. Yang, E. H.-M. Sha, Q. Zhuge, S. Gu, S. Dasgupta, Y. Shi, and J. Hu, "Hardware/software co-exploration of neural architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, early access*, 2020.