Black Box Search Space Profiling for Accelerator-Aware Neural Architecture Search

Shulin Zeng[1*], Hanbo Sun[1*], Yu Xing[2], Xuefei Ning[1]
Yi Shan[2], Xiaoming Chen[3], Yu Wang[1], Huazhong Yang[1]
[1]Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China
[2]Xilinx, Beijing, China, [3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

Abstract— Neural Architecture Search (NAS) is a promising approach to discover good neural network architectures for given applications. Among the three basic components in a NAS system (search space, search strategy, and evaluation), prior work mainly focused on the development of different search strategies and evaluation methods. As most of the previous hardware-aware search space designs aimed at CPUs and GPUs, it still remains a challenge to design a suitable search space for Deep Neural Network (DNN) accelerators. Besides, the architectures and compilers of DNN accelerators vary greatly, so it is quite difficult to get a unified and accurate evaluation of the latency of DNN across different platforms. To address these issues, we propose a black box profiling-based search space tuning method and further improve the latency evaluation by introducing a layer adaptive latency correction method. Used as the first stage in our general accelerator-aware NAS pipeline, our proposed methods could provide a smaller and dynamic search space with a controllable trade-off between accuracy and latency for DNN accelerators. Experimental results on CIFAR-10 and ImageNet demonstrate our search space is effective with up to 12.7% improvement in accuracy and 2.2x reduction of latency, and also efficient by reducing the search time and GPU memory up to 4.35x and 6.25x, respectively.

I. Introduction

Neural Architecture Search (NAS), first proposed in [1], can automatically discover good architectures for a specific application. Researchers have demonstrated that NAS can discover Deep Neural Network (DNN) architectures with surpassing accuracy than the human-designed ones (e.g., *VGGNet*, *GoogleNet*, *ResNet*). However, there are two major issues in the early NAS frameworks. One is that the lengthy search time can be up to thousands of hours due to the large search space and the training process of each candidate network. The other is that many NAS frameworks are only optimized with accuracy as the mono-objective. This could lead to a violation of latency constraints when deployed on the DNN accelerators.

Recently, many researchers have begun to study the efficiency and multi-objective optimization of NAS. A typical NAS workflow that could address these issues is shown in Fig.1. The first step is to change the Search Space (SS) from a larger one into a smaller one. The basic idea is to manually design the network backbone and the structure of each candidate block based on expert experience, thus limiting the SS to the selection of candidate blocks at each layer. The second step is to automatically find the optimal candidate network from the SS by a carefully designed search strategy. For example, Reinforcement Learning (RL) [2], evolutionary algorithms [3], and Differentiable NAS (DNAS) method [4] are common search strategy algorithms in recent NAS frameworks. The final step is the evaluation of multi-objective optimization. That is, not only should we obtain high accuracy, but also ensure a low latency when deployed on the target hardware platform. Accuracy can be obtained by training and validating on the target data sets directly, while the evaluation of latency requires a simulator or a latency model obtained by running the candidate blocks on the target accelerator in advance.

As mentioned above, it should be noted that the second step is an algorithm-level optimization, while the first and third steps are highly dependent on the target platform, including the hardware architecture and the software compiler. Prior work [5], [6] has studied the implementation of hardware-aware NAS frameworks on the algorithm level. However, most of these hardware-aware NAS frameworks are targeted for mobile CPUs, other hardware platforms such as Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) are not discussed. Take FBNet [5] as an example, its candidate blocks in the SS are inspired by MobileNet [7], which is specially designed for mobile CPU platform. As there are more and more DNN accelerators based on FPGAs and ASICs, their architecture design and compilation tools are quite different from each other. We can not guarantee that the CPU-optimized MobileNet can still achieve high utilization and low latency on these accelerator platforms. As reported in [8], VGG-16 can achieve a high utilization rate of 87.30% while MobileNet-v1 can only achieve 28.62% on a DNN accelerator. Since the SS restricts the upper bound of the network performance obtained by NAS, ignoring the relationship between the SS and the accelerator platform will lead to the failure of NAS to obtain the optimal network. Besides, in most cases, we are unable to obtain any information about the hardware architecture and the software compiler, making it hard to accurately evaluate the latency of each candidate network on these black box accelerators.

Recently, there has been some work paying attention to the NAS targeted for FPGA-based accelerator. Hao [9] proposed a hardware-oriented DNN model design by introducing a DNN template to guide the DNN generation with predictable performance and resource utilization. Jiang [10] built up an FPGA-implementation aware NAS framework with a graph model to provide the basic support for latency analysis. Both of these two work aimed at providing a NAS solution for template-based accelerator on FPGA, and their SS only consisted of simple operations such as 3×3 Conv layer and 2×2 Pool layer. In this paper, we focus on another type of DNN accelerator, which is based on Instruction Set

*: Both authors contributed equally to this work.

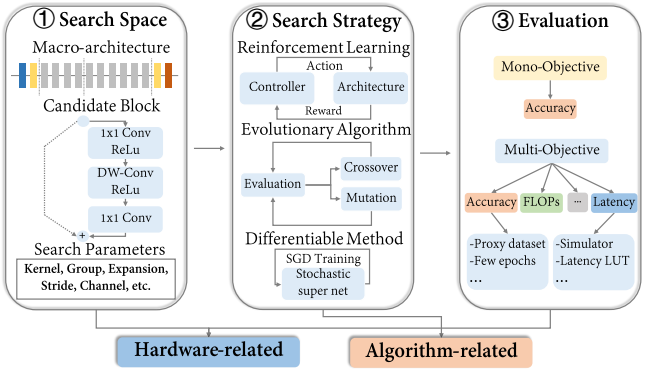

Fig. 1. A typical workflow of NAS consists of three steps: (1) search space, (2) search strategy, and (3) evaluation.

Architecture (ISA) and widely used on both the FPGAs and ASICs. Besides, we treat the DNN accelerator as a black box hardware model, meaning that we have no access to any prior knowledge about the DNN accelerators. In addition, the SS we studied is composed of complex network blocks, which are commonly used in the state-of-the-art [5], [3], [6] NAS framework. To overcome these challenges, we propose a black box profiling-based search space tuning method and a layer adaptive latency correction method to build a general accelerator-aware NAS framework with a controllable tradeoff between accuracy and latency.

The main contributions of this paper are as follows.

- We propose a black box profiling-based search space tuning method to generate a smaller and dynamic search space for the target DNN accelerator platform with a controllable tradeoff between accuracy and latency.
- We propose a layer adaptive latency correction method with a policy-aware latency LUT for fast and accurate latency estimation.
- We demonstrate a general hardware-aware NAS framework using an ISA-based DNN accelerator on FPGA as an example to show the efficiency and effectiveness of our proposed methods.
- Experimental results show that on the CIFAR-10 and ImageNet data sets, our search space can achieve an average of 1.4% improvement in accuracy, 1.9x reduction of latency, 2.7x and 4x reduction of the search time and GPU memory, respectively.

II. Preliminary

A. Neural Architecture Search

The process of a NAS algorithm usually goes as follows. At each iteration, an architecture $a \in \mathcal{A}$ sampled from the SS $\mathcal{A}$, is a directed acyclic graph (DAG). This architecture is further assembled as a candidate network $N(w, a)$, where w is the weight to be trained. The objective evaluation results of the candidate network will be used to instruct the sampling process. The goal of the NAS problem is to discover the architecture that maximizes the chosen objective, which can be formulated as:

$$\begin{aligned} a^* &= \arg\max_{a \in \mathcal{A}} R_v(N(w^*, a)) \\ \text{s.t. } w^* &= \arg\min_{w} L_t(N(w, a)) \end{aligned} \tag{1}$$

In the purest formulation of NAS, for each architecture a, the w^* should be found by optimizing the loss of the candidate network on the training data set, and then the architecture that can achieve the best performance on the validation data set should be found.

Originally, the reward R in Equ.1 is the accuracy mono-objective. To introduce multiple objectives other than the accuracy, there are mainly two ways: as the constraints, or use the weighted sum of multiple objectives as the reward. Using the weighted sum in the reward can provide explicit trade-off in the searching process, in which case the scalarized reward of multiple objectives R can be written as:

$$R(a) = (1 - \beta) Acc_v(N(w^*, a)) + \beta T(a) \tag{2}$$

where the T could be latency, FLOPs, and energy.

The NAS frameworks can be classified into three categories according to the search strategy: (1) reinforcement-learning-based methods [1], [2], [11], that employ RL-based methods to optimize the controller using the objective evaluation results of the candidate network; (2) evolutionary algorithms [12], in which the trained candidate networks are treated as the population, and the next generation of candidate networks are sampled using crossover and mutations; and (3) differentiable methods [5], [4], in which relaxed differentiable formulation of the NAS problem is used, and gradient-based methods are used to train both the weights and the architecture parameters.

Although great progress has been made from the search strategy and evaluation perspective, there is still a lack of research on the impact of the SS. The SS of prior work is designed based on the prior knowledge of how DNNs perform on the General-Purpose Processors (GPP), such as GPUs and CPUs. When it comes to the Specialized Domain Accelerators (SDA) on FPGAs and ASICs, due to the heterogeneity of the various platforms, it's quite difficult to acquire unified prior knowledge. The mismatch between SS and hardware will be detrimental to the effectiveness of NAS.

B. DNN Accelerator

DNN accelerator is a kind of SDAs to provide up to 100x-1000x energy efficiency than GPPs in the neural network computing domain. There is a lot of work focusing on the design of DNN accelerators implemented in FPGAs [13], [14] and ASICs [15], [16]. There are two main design tracks in the DNN accelerators: (1) mapping DNN layers into several computing blocks using High-level Synthesis (HLS) or Register-Transfer Level (RTL) templates on FPGA, such as fpgaConvNet [17] and DnnWeaver [18], and (2) ISA-based architecture [19], [15], that utilizes compiler to map DNN network to the instructions for the underlying hardware.

Recently, hardware-aware NAS work [10], [9] has taken the template-based FPGA accelerator designs into consideration. However, there is currently no research on NAS frameworks for the second type of DNN accelerator as far as we know. Since the hardware architectures and software compilers of the ISA-based DNN accelerators are of various types and have their own characteristics, it is impractical to analyze each kind of accelerator design to obtain a unified prior knowledge for the SS. In this paper, we regard all kinds of DNN accelerators

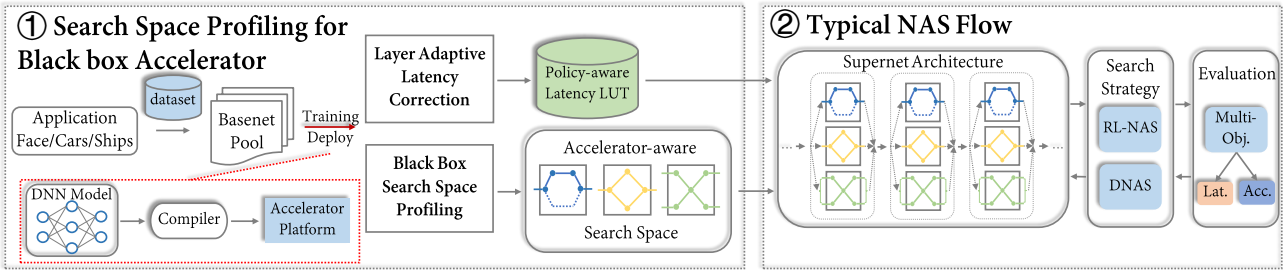

Fig. 2. A general accelerator-aware NAS framework consists of two parts: (1) search space profiling for black box accelerator, and (2) typical NAS flow.

TABLE I
ACCURACY, LATENCY, SEARCH TIME, AND MEMORY CONSUMPTION OF FOUR DIFFERENT SEARCH SPACE ON CIFAR-10 WITH A 10-LAYER NETWORK BACKBONE USING RL-NAS AND DNAS: RESNET-BASED, VGG-BASED, MOBILE-BASED, AND ALL OF THE THREE BLOCKS.

Method	DNAS				RL-NAS			
Search Space	Acc. (%)	Lat. (ms)	Time (min)	Mem. (MB)	Acc. (%)	Lat. (ms)	Time (min)	Mem. (MB)
Res	86	1.287	165	1800	83.5	1.385	140	1200
Mobile	85.4	0.825	125	1700	82.7	0.865	104	1400
VGG	83.8	0.775	70	1800	82.2	0.812	65	1300
All	**84.1**	**0.859**	**326**	**6000**	**73.8**	**1.113**	**180**	**3000**

as a black box model, thus the method of building a latency model [10] will no longer be applicable here. The latency Look-Up Table (LUT) proposed in [5] is a fast and effective method to estimate the latency layer by layer for a black box accelerator, but it ignores the inter-layer effects introduced by layer fusion policies [20] of the compiler. Thus, a latency correction model is proposed to evaluate the latency of each layer more accurately.

III. FRAMEWORK

In this section, a general accelerator-aware NAS framework is demonstrated at first. Next, we will give a comprehensive illustration of the proposed black box search space profiling method and a layer adaptive latency correction method.

A. Framework Overview

A general accelerator-aware NAS framework targeted for the black box accelerator platform is shown in Fig.2. We introduce a black box search space profiling method, which will be further explained in section III.III-B, as the first step of the proposed framework to provide an efficient and effective SS. As for the second step of the accelerator-aware NAS process, we use the accelerator-aware SS obtained in the first step to construct the supernet of each layer. Then the search strategy of RL-NAS and DNAS are used as two examples to optimize the supernet. It should be noted that our framework is also applicable to other NAS search strategies. In addition, we apply a layer adaptive latency correction method to obtain a policy-aware latency LUT. The red arrow in both steps represents a typical deployment flow of ISA-based DNN accelerators. A DNN model is at first transformed from a float-point dense network into a fixed-point sparse one with the utilization of compression methods, such as pruning and quantization. Secondly, the compiler takes the compressed model as input and then generates the files needed for deployment on the accelerator platform.

B. Search Space Profiling for Black Box Hardware

Search space limits the accuracy of NAS-derived network architecture on the target application and its latency when deployed on the accelerator platform. Without the prior knowledge of the candidate network blocks and accelerator architecture, an intuitive idea is to use all the candidate network blocks to form a large SS and apply the NAS process on this SS directly. However, this will bring quite a high search cost, including lengthy search time and large GPU memory consumption, and even lead to performance degradation of the network obtained by NAS. As shown in Table I, compared with three different SS, directly merging all candidate network blocks into a large SS leads to 1.3x-4.7x longer search time, 2.1x-3.5x larger memory consumption, and sub-optimal network architecture on CIFAR-10 data set. Thus, in the case where multiple candidate network blocks are provided, we should profile and evaluate such a large SS, so that a small and effective one can be obtained to ensure the efficiency of the NAS process and the desired performance of accuracy and latency on the target application and accelerator. Next, we will give a detailed description of the flow of search space profiling for black box hardware, as shown in Fig.3.

Search Space Base Networks. For evaluating each SS consisting of different candidate blocks, we introduce the concept of Search Space Base Networks (SSBNs) and present a cost function for evaluating the SS. As shown in Fig.2, there are multiple candidate blocks in the supernet of each layer. For a network backbone of L layers with N candidate network blocks on each layer, there will be N^L possible choices. It is impractical to evaluate all the possible choices. The idea of SSBNs is as follows: On the basis of a certain network in the SS, at most one supernet on a certain layer is changed at a time, thus constituting the SSBNs based on a Specified Network (SN). For example, for a two-layer network backbone, the possible choice of supernet on each layer are selected on $\{1, 2\}$. Based on a SN $(1, 1)$, we generate the SSBNs of $\{(1, 1), (1, 2), (2, 1)\}$. Thus, we can reduce the size of SS from N^L to $N \times L$, making it possible for efficient profiling. The idea of SSBNs is under the basic hypothesis that each network in the original SS can be regarded as a linear combination of some networks in the SSBNs. Such an assumption of linear independence between each layer of supernet has been validated in [5] and proved to be effective for the NAS process. Based on the proposed method, we present the cost function for SS:

$$Cost(SS) = \frac{1}{card(SSBN)} \sum_{Net \in SSBN} Cost(Net) \qquad (3)$$

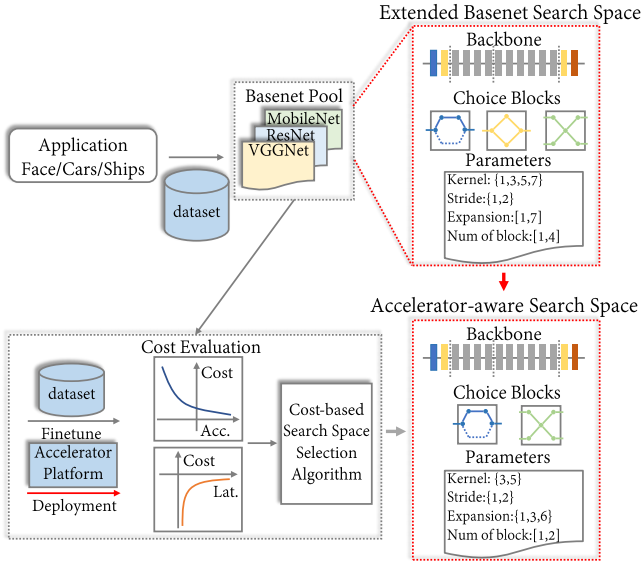

Fig. 3. The flow of search space profiling for black box accelerator. A cost-based search space selection algorithm is proposed to find a accelerator-aware SS from the extended basenet SS.

where $card$ is a counting function and Net represents a child network. $Cost(Net)$ is the cost calculated by Equ.6, which will be introduced later. The optimization goal is to minimize the $Cost(SS)$.

Cost Function for Candidate Blocks. For a better evaluation of the multi-objective constraints of accuracy and latency, we propose several cost functions to evaluate the candidate network blocks in the SS.

(a) Accuracy. For a specific application, users usually put forward a threshold requirement for accuracy. When the accuracy of the DNN model can not meet the user's requirements, this model can not be deployed on the accelerator platform. Therefore, we can use the step function to describe the impact of accuracy for a target application. Considering the discontinuity of the step function, we use the exponential function to fit the impact of accuracy. The cost function for accuracy is given as follows:

$$Cost_{acc} = \exp\left(-\frac{accuracy - thres}{scale}\right) \tag{4}$$

where $thres$ represents the user-defined accuracy threshold and $scale$ denotes the transformation coefficient, which is used to describe the tolerance of low accuracy.

(b) Latency. The latency of DNN models deployed on the accelerator should be as low as possible. In practical applications, people tend to use the reciprocal of the latency as a measurement. The cost function for latency is as follows:

$$Cost_{latency} = -\frac{1}{latency} \tag{5}$$

here the negative sign is to ensure that $Cost_{latency}$ decreases as $latency$ decreases.

(c) Total Cost Function. In practical scenarios, accuracy and latency are often negatively correlated. To describe the overall impact of accuracy and latency, the formula is shown below:

$$Cost = Cost_{acc} + \lambda * Cost_{latency} \tag{6}$$

Algorithm 1 Cost-based Search Space Selection Algorithm

Require: Basenet Pool: BP, Size Threshold: Thr
Require: Cost function: $Cost$
Require: Search space base networks generator: $SSBNG$
Ensure: Optimized Search Space
Wider search space : $WSS \leftarrow []$
for backbone B in enumerate(BP) **do**
  $SSBN \leftarrow$ SSBNG(B)
  WSS.append($SSBN$)
  Count(B) $\leftarrow 0$
  List(B) $\leftarrow []$
end for
Cost list : $CL \leftarrow []$
for network N in enumerate(WSS) **do**
  CL.append(Cost(N))
end for
Rearrange WSS in ascending order of corresponding value in CL.
for network N in enumerate(WSS) **do**
  Count(type(N)) $\leftarrow$ Count(Type(N)) + 1
  List(type(N)).append(N)
  if Count(type(N)) $> Thr$ **then**
    return List(type(N))
  end if
end for

where λ is a coefficient to balance $Cost_{acc}$ and $Cost_{latency}$. When λ is large, the network with a better latency is more likely to be found, while the network with higher accuracy is likely to be found when λ is small.

Search Space Profiling and Selection. Based on the proposed SSBNs and cost functions, we illustrate the flow of the search space profiling method, as shown in Fig.3.

Firstly, we generate the SSBNs for each candidate SS, which has a different architecture of candidate network block and multiple parameters to be searched. Secondly, once we have the basenet pool consists of different SSBNs, we need to evaluate the accuracy and latency of each candidate network in the basenet pool. The latter is easier by running each base network on the accelerator platform. As for the accuracy, we directly train the specified network of each candidate SS on the target data set with 20 epochs, which is the same as used in [9] for evaluating different operators. Then we use the weights of the specified network to finetune each base network with a few epochs to obtain the accuracy. Next, we can compute the cost of each base network using Equ.6. At last, in order to minimize the $Cost(SS)$ in Equ.3, we design a cost-based search space selection algorithm as shown in Algorithm 1. Using the proposed algorithm, we can derive an optimized SS, of which the size is much smaller and controllable by size threshold Thr, and its performance can provide a tradeoff between accuracy and latency with the user-defined parameter λ. What's more, our profiling method can generate a dynamic SS with a different number of candidate blocks at each layer. We will demonstrate that dynamic SS can be more efficient and effective than fix SS in section IV.

C. Layer Adaptive Latency Correction Method

Our layer adaptive latency correction method supports two different search strategies: whole supernet and single path. As for the first one, all the candidate network blocks in the supernet are optimized simultaneously. The latency of the n-th candidate block at the l-th layer can be estimated as followed:

TABLE II

RESULTS OF DNAS AND RL-NAS ON CIFAR-10 IN A PROXYLESS MANNER. THE RESNET-BASED, MOBILENET-BASED, AND VGGNET-BASED SEARCH SPACE ARE OBTAINED USING OUR PROPOSED SEARCH SPACE PROFILING METHOD WITH λ OF 0.01, 0.1, AND 1, RESPECTIVELY.

Method	DNAS (Proxyless)								RL-NAS (Proxyless)							
Search Space	Acc. (%)	Lat. (ms)	Cost (λ =) 0.01	0.1	1	Time (min) Pre.	Search	Total	Acc. (%)	Lat. (ms)	Cost (λ =) 0.01	0.1	1	Time (min) Pre.	Search	Total
Res	86.7	1.30	**0.9838**	0.9146	0.2217	23	90	**113**	86.5	1.29	**0.9848**	0.9149	0.2161	23	80	**95**
Mobile	85.6	0.83	0.9850	**0.8765**	-0.2078		68	**91**	85.7	0.84	0.9845	**0.8767**	-0.2011		64	**87**
VGG	84.0	0.78	0.9923	0.8775	**-0.2705**		42	**75**	84.2	0.80	0.9915	0.8789	**-0.2476**		51	**60**
All	84.1	0.86	0.9929	0.8881	-0.1596	0	326	326	73.8	1.11	1.0486	0.9678	0.1591	0	180	180

TABLE III

RESULTS OF DNAS AND RL-NAS ON IMAGENET IN A PROXYLESS AND TRANSFER MANNER, RESPECTIVELY. THE RESNET-BASED AND VGGNET-BASED SEARCH SPACE ARE OBTAINED USING OUR PROPOSED SEARCH SPACE PROFILING METHOD WITH λ OF 0.01 AND 1, RESPECTIVELY.

Method	DNAS (Proxyless)								RL-NAS (**Transfer**)							
Search Space	Acc. (%)	Lat. (ms)	Cost (λ =) 0.01	1	Time (min) Pre	Search	Total	Memory (GB)	Acc. (%)	Lat. (ms)	Cost (λ =) 0.01	1	Time (min) Pre.	Search	Total	Memory (GB)
Res.	71.2	3.553	**0.9912**	0.7126		355	**715**	**24**	71	3.605	**0.9922**	0.7176		90	450	**1**
VGG	68.7	1.021	0.9967	**0.0271**	360	365	**725**	**24**	68.9	1.324	0.9980	**0.2502**	360	60	420	**1**
All	69.2	1.708	0.9982	0.4185	0	1440	1440	150	67.8	2.877	1.0076	0.6635	0	180	**180**	3

$$L_{BNblock}(n,l) = L_{BN}(n,l) - L_{SN} + L_{SNblock}(l) \quad (7)$$

where $L_{BN}(n,l)$ and $L_{SNblock}(l)$ denotes the latency of SSBN with the target block being substituted and the single-layer l in the specified network, respectively.

The single path method samples a child network from the whole supernet at each iteration. Thus, we first generate N specified networks $SN_i(i = 0, 1, 2, ..., N-1)$, each consists of only one kind of candidate network block. Secondly, the $SSBN_i$ of each SN_i are generated. Next, we can get the $L_{BNblock}(n,l,i)$ in each $SSBN_i$ using Equ.7. Then, for a choice block n at the l-th layer of a child network, its latency $L_{CNblock}(n,l)$ can be estimated as followed:

$$L_{CNblock}(n,l) = \frac{1}{L} \sum_{(n',l') \in CN} L_{BNblock}(n,l,n') \quad (8)$$

where (n', l') denotes the choice block n' at layer l' in the child network. By weighted averaging the effects of all layers in the child network, we can get a more accurate latency estimation of the choice block. Using the proposed layer adaptive latency correction method, we can set up a policy-aware latency LUT for fast and accurate latency estimation.

IV. EXPERIMENTS

A. Experiment Setup

Accelerator Platform. We use Xilinx DPU [20] as the target accelerator platform, which has been released as a Vivado IP for accelerating CNNs on Xilinx FPGA. A Xilinx DPU IP, with the parallelism of 4096 operations per cycle, is synthesized with 333 MHz on Xilinx ZCU102 FPGA using Vivado 2018.2. CNN models are first trained with Caffe. Then the weights and biases are transformed from a 32-bit floating point to an 8-bit fix point using the quantization method in [19]. We evaluate the actual latency on the accelerator directly.

NAS Environment. We implement two kinds of search strategies in our NAS framework: ENAS [11] and FBNet [5], corresponding to RL-based and Differentiable method, respectively. We follow the hyperparameters in [11] and [5] for better comparison. The search epoch is 200 for ENAS and 90 for FBNet, and we train the NAS-derived network for 200 epochs on CIFAR-10 and 100 epochs on ImageNet. Since there are no hardware constraints in ENAS, we modify its reward function by adding a penalty of latency, as shown in Equ.2. Our NAS framework and the training process for the searched models are both implemented in PyTorch. The target datasets include CIFAR-10 and ImageNet. Kindly noted that we randomly sample 100 classes from the original 1000 classes of ImageNet to reduce the overall search time.

TABLE IV

THE AVERAGE ACCURACY AND LATENCY OF THREE CANDIDATE SEARCH SPACE IN THE BASENET POOL ON CIFAR-10 AND IMAGENET DATA SETS.

Data sets	CIFAR-10		ImageNet (100 classes)	
Basenet Pool	Acc. (%)	Lat. (ms)	Acc. (%)	Lat. (ms)
Res.	83.10	1.87	60.72	3.89
Mobile.	81.22	1.12	59.77	3.52
VGG.	80.15	0.95	59.02	2.61

Search Space. Three hand-designed SS, VGG-based, ResNet-based, and MobileNet-based are provided as candidate SS. The number of candidate choices of the ResNet-based and MobileNet-based network blocks is both 9 while the VGG-based network blocks are 13. We follow a similar macro-architecture of 12 layers as in FBNet [5]. The average accuracy and latency of the three SS on CIFAR-10 and ImageNet are listed in Table IV.

B. Effectiveness and Efficiency of Search Space Profiling

Effectiveness of Cost Function Design. As shown in Table II, we obtain three different kinds of SS using our proposed search space profiling method on CIFAR-10 with $thres$ and $scale$ set to 0.85 and 2, respectively. The λ is set to 0.01, 0.1, and 1 to get the accuracy-optimized, accuracy-latency-balanced, and latency-optimized SS, each corresponding to ResNet-based, MobileNet-based, and VGGNet-based SS. Both of the RL-NAS and DNAS demonstrate that the NAS-derived network on the obtained SS has the lowest cost compared to the other two. As for the accuracy and latency, it is indeed that the ResNet-based SS has the best accuracy of 86.7% and 86.5%, the VGGNet-based one has the lowest latency of 0.78 ms and 0.8 ms, and the MobileNet-based one provides a tradeoff between accuracy and latency. The experimental results of ResNet-based and VGGNet-based SS on the ImageNet also validate the effectiveness of our cost function design, as shown in Table III.

TABLE V
RESULTS OF FIX AND DYNAMIC SEARCH SPACE OF RESNET-BASED NETWORK BLOCKS ON CIFAR-10.

	Fix Search Space			dynamic search space			
Method	Acc. (%)	Lat. (ms)	Cost	λ	Acc. (%)	Lat. (ms)	Cost
DNAS	86	1.287	0.9872	0.01	86.7	1.299	**0.9838**
			0.9173	0.1	86	1.28	**0.9169**
			0.2180	1	85.8	1.255	**0.1992**
RL-NAS	83.5	1.385	1.0003	0.01	86.5	1.292	**0.9848**
			0.9353	0.1	85.4	1.27	**0.9193**
			0.2855	1	84.7	1.25	**0.2015**

Dynamic vs Fix Search Space. We also compare the performance between dynamic and fix SS using ResNet-based network blocks as an example on CIFAR-10, as shown in Table V. The size threshold Thr of the SS is 60. The experimental results show that the dynamic SS under three different λ configurations has the lowest cost compared with fix SS. This means that our proposed search space profiling method can preserve the useful information of the original SS, while pruning away redundant information in SS, thus making the search process fastly converge to a better result more faster.

Compared with Baseline. We use the case that all candidate blocks constitute a SS as the baseline. The experimental results on CIFAR-10 show that the networks derived from our SS all have a lower cost than the baseline with 2.6% to 12.7% improvement in accuracy and 1.1x to 1.4x reduction of latency, and the total search time is reduced by 1.89x to 4.35x. As for the ImageNet, a lower cost is also achieved with 2% to 3.2% improvement in accuracy and 1.7x to 2.2x reduction of latency. The total search time of DNAS is reduced by 2x. While the baseline of RL-NAS is faster than ours. It is because we need to do the search space profiling on the ImageNet, which takes a much longer time than searching on the CIFAR-10 for transferring. However, the NAS-derived network of our SS is much better than the baseline. Besides, we can save the GPU memory from 3x to 6.25x.

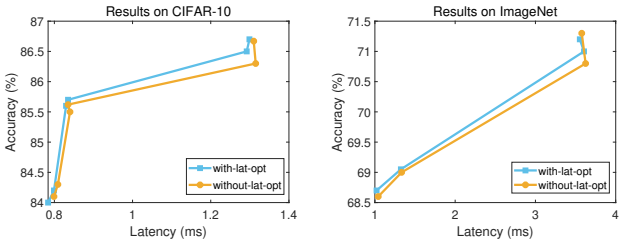

Fig. 4. Results of the Pareto curve with or without the latency optimization on: (a) CIFAR-10, and (b) ImageNet.

C. Impact of Latency Correction Method

As shown in Fig.4, we evaluate the impact of our proposed layer adaptive latency correction method on CIFAR-10 and ImageNet. The blue lines represent the results with the optimization of the latency correction method, as shown in Table II and Table III, while the orange line represents the results without the latency optimization. It can seem that a better Pareto curve can be obtained if we can get a more accurate estimation of the latency using the policy-aware latency LUT.

V. CONCLUSIONS

In this paper, we proposed a black box search space profiling tuning method to generate a smaller and dynamic search space for the target DNN accelerator without any prior knowledge of the underlying hardware architecture. Next, we design a layer adaptive latency correction method to provide a policy-aware latency-LUT for fast and accurate latency estimation. By using the proposed methods, we demonstrate a general hardware-aware NAS framework using an ISA-based DNN accelerator on FPGA. Experimental results show that on the CIFAR-10 and ImageNet data sets, our search space can achieve an average of 1.4% improvement in accuracy, 1.9x reduction of latency, 2.7x and 4x reduction of the search time and GPU memory, respectively.

ACKNOWLEDGMENTS

This work was supported by National Key R&D Program of China 2018YFB0105005 and National Natural Science Foundation of China(No. 61622403, 61621091). This work was also supported by Xilinx and Beijing Innovation Center for Future Chips, Tsinghua Xilinx AI Research Fund, Beijing National Research Center for Information Science and Technology (BNRist), and the project of Tsinghua University and Toyota Joint Research Center for AI Technology of Automated Vehicle(TT2018-01). Chen's work was supported by the Beijing Academy of Artificial Intelligence under Grant BAAI2019QN0402.

REFERENCES

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.
[2] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," in *Proc. of CVPR*, 2018, pp. 8697–8710.
[3] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," *arXiv:1904.00420*, 2019.
[4] S. Xie *et al.*, "Snas: stochastic neural architecture search," in *ICLR*, 2019.
[5] B. Wu *et al.*, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. of CVPR*, 2019, pp. 10 734–10 742.
[6] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.
[7] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. of CVPR*, 2018, pp. 4510–4520.
[8] S. Yao *et al.*, "The evolution of accelerators upon deep learning algorithms," website, 2018, https://www.hotchips.org/hc30/2conf/.
[9] C. Hao *et al.*, "Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge," in *Proc. of the 56th DAC*, 2019, p. 206.
[10] W. Jiang *et al.*, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proc. of the 56th DAC*, 2019.
[11] H. Pham *et al.*, "Efficient neural architecture search via parameter sharing," in *Proc. of ICML*, 2018, pp. 4092–4101.
[12] E. Real *et al.*, "Regularized evolution for image classifier architecture search," in *Proc. of AAAI*, 2019, pp. 4780–4789.
[13] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proc. of FPGA*, 2016, pp. 26–35.
[14] S. Han *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proc. of FPGA*, 2017, pp. 75–84.
[15] S. Zhang *et al.*, "Cambricon-x: An accelerator for sparse neural networks," in *Proc. of Micro*, 2016, p. 20.
[16] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *JSSC*, vol. 52, no. 1, pp. 127–138, 2016.
[17] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *Proc. of FCCM*, 2016, pp. 40–47.
[18] H. Sharma *et al.*, "Dnnweaver: From high-level deep network models to fpga acceleration," in *the Workshop on Cognitive Architectures*, 2016.
[19] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE TCAD*, vol. 37, no. 1, pp. 35–47, 2017.
[20] Y. Xing *et al.*, "Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators," *IEEE TCAD*, 2019.