On Neural Architecture Search for Resource-Constrained Hardware Platforms

Qing Lu
University of Notre Dame
qlu2@nd.edu

Weiwen Jiang
University of Notre Dame
wjiang2@nd.edu

Xiaowei Xu
University of Notre Dame
xxu8@nd.edu

Yiyu Shi
University of Notre Dame
yshi4@nd.edu

Jingtong Hu
University of Pittsburgh
jthu@pitt.edu

ABSTRACT

In the recent past, the success of Neural Architecture Search (NAS) has enabled researchers to broadly explore the design space using learning-based methods. Apart from finding better neural network architectures, the idea of automation has also inspired to improve their implementations on hardware. While some practices of hardware machine-learning automation have achieved remarkable performance, the traditional design concept is still followed: a network architecture is first structured with excellent test accuracy, and then compressed and optimized to fit into a target platform. Such a design flow will easily lead to inferior local-optimal solutions. To address this problem, we propose a new framework to jointly explore the space of neural architecture, hardware implementation, and quantization. Our objective is to find a quantized architecture with the highest accuracy that is implementable on given hardware specifications. We employ FPGAs to implement and test our designs with limited loop-up tables (LUTs) and required throughput. Compared to the separate design/searching methods, our framework has demonstrated much better performance under strict specifications and generated designs of higher accuracy by 18% to 68% in the task of classifying CIFAR10 images. With 30,000 LUTs, a light-weight design is found to achieve 82.98% accuracy and 1293 images/second throughput, compared to which, under the same constraints, the traditional method even fails to find a valid solution.

1 INTRODUCTION

Machine learning has demonstrated great success in a variety of applications [10, 15, 19, 22], which leads to the ever-growing demand in the off-the-shelf solutions to application-specific systems [5, 8, 14, 23]. Designing neural networks applying the hand-crafted approach, however, involves huge expertise and labor. In response to this challenge, automated machine learning (Auto-ML) is proposed to build neural networks without human intervention; in particular, Neural Architecture Search (NAS) [26] is proposed to identify the neural architecture with competitive or even better accuracy against the best design explored by experts.

On the other side, when deploying architectures explored by NAS to real-world platforms, such as AIoT [12] and mobile embedded platforms [2–4, 13, 17], it is inevitably limited by the hardware constraints. As a result, hardware-aware machine learning [2, 6, 17, 25] has emerged to explore neural architectures with the consideration of hardware efficiency on a target fixed hardware design. Most recently, authors in [7] open the hardware space in NAS to jointly explore the architectures and hardware designs. However, almost all existing methods adopt a separated optimization flow [9]: a large network is first invented with excellent performance, and then compressed and optimized to fit into a target platform. Note that compression techniques, especially quantization [18–21], have to be considered to fit the model into resource-constrained hardware platforms, which can tremendously reduce the hardware resource consumption and related computation consumption. Consequently, such approach usually failed to find the overall optimal solutions. For example, the best quantization scheme specifically tuned for a network may be significantly inferior when applied to another network or even not implementable under certain hardware specifications.

In this paper, we delve into the NAS-based methods of design automation on hardware-constrained platforms. We aim to answer such a concrete question: for a specific task, what is the best neural architecture with the highest accuracy that is implementable given a defined set of hardware specifications? In particular, a novel co-exploration framework is proposed to investigate the optimality of neural architectures with quantization. In our framework, we parameterize the layer-wise quantization and search these parameters jointly with the hyperparameters of the architecture. A hardware model is built by searching the hardware space and validated by the design specifications. We use FPGAs as the target platform and run experiments under various configurations and specifications. Compared with the existing separately searching, the proposed joint search method is more robust, achieving 18% to 68% higher accuracy on common used data sets.

The remainder of this paper is organized as follows. In Section 2, we outline the progress in neural architecture search associated with hardware design. After that, we present the details of our design framework in Section 3. Section 4 then investigates the performance of our framework by experiment as compared with conventional methods of separate search. Finally, Section 5 remarks the conclusion and future work.

ICCAD '19, November 4-7, 2019, Westminster, CO

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

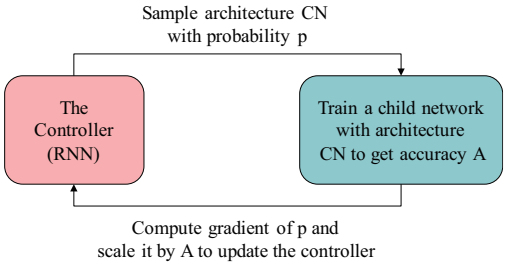


Figure 1: The Pure-Software NAS framework.

2 TODAY'S NAS: FROM PURE-SOFTWARE VIA HARDWARE-AWARE TO CO-DESIGN

Likewise the development of designing the embedded systems, the evolution of today's NAS has gone through three phases: (1) exploring structure only, called pure-software NAS in this paper; (2) considering efficiency on a fixed hardware in exploring structures, called hardware-aware NAS; (3) co-exploring hardware implementation and structures, called Co-Design NAS. In the following, we will introduce each phase in detail, and then outline the development trend of NAS in the future.

Pure-Software NAS. Figure 1 shows the NAS framework presented in [26]. In NAS, a controller (implemented as an RNN) iteratively generates a child network and obtains its accuracy A by training it on a held-out data set. Then, accuracy A will be used as the reward signal to the controller for its self-evolution for the next iteration. The search process will be stopped if the controller is converged for the maximum accuracy, or a termination condition is satisfied. Existing work has demonstrated that the automatically generated network architectures can achieve close accuracy to the best human-invented architectures on the image classification task [26, 27].

Search Space: For image classification, the linear array is applied as the backbone of network architecture. In [26], each cell is a normal convolution operation. In each cell, the search space is composed of the filter size, strides, and the number of filters. In [27], authors propose to incorporate B blocks ($B = 5$ in the paper) in one cell, where each block is a 2-branch structure, mapping from 2 input tensors to 1 output tensor. And the controller determines the type of operation on each input tensor. The operations include the different size of depthwise-separable convolution, atrous convolutions, average pooling, max pooling, skip connection, etc.

Hardware-Aware NAS. Figure 2 illustrates the works on searching neural architectures targeting for fixed hardware [2, 13, 17]. In these works, mobile phones are commonly be employed to be the testbed. In order to guarantee the final system can satisfy the timing specification. The framework will test the hardware efficiency (e.g., latency, energy consumption) for each child network. As shown in Figure 2, after training, the child network will be sent to the target platform to be executed. During execution, the hardware efficiency E will be profiled. E together with the accuracy A will be applied to update the controller to explore a better neural network architecture.

More specifically, authors in [13] propose two optimization methods. Assume the hardware efficiency E stands for latency. Given the latency specification S, the first method is to maximize the accuracy

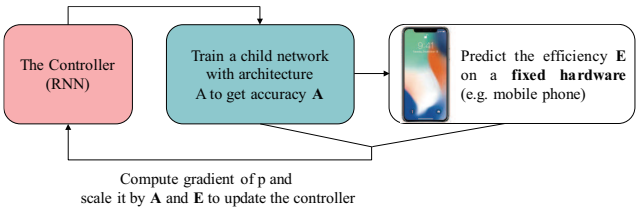


Figure 2: The Hardware-Aware NAS framework.

A, subjecting to the constraint of $E \leq S$. With this method, it still has the mono-criteria on maximizing accuracy. This method can guarantee the hardware efficiency to meet the specifications, but it cannot provide the Pareto optimal solutions. Then, a weighted product method to approximate Pareto optimal solutions is proposed. The objective function is revised as $max = A \times \frac{E}{S}^{w}$, where w is the weight factor. In this way, it enables the controller to effectively approximate Pareto solutions nearby the specification S.

All the above approaches consider the hardware efficiency during the search space. However, they neglect the hardware design freedom, which is commonly given in many AI applications (e.g., IoT, embedded systems). As a result, it will potentially lead to inferior solutions. A more elegant way to tailor the hardware design for neural architecture needs to be exploited.

Co-Design NAS. Most recently, we propose the hardware/software co-design NAS to simultaneously optimize architecture accuracy and hardware efficiency. Interestingly, we observe that the hardware design space is tightly coupled with the architecture search space, i.e., the best neural architecture depends on the hardware (hardware-aware NAS), and the best hardware depends on the neural architecture. It is therefore best to jointly explore both spaces to push forward the Pareto frontier between hardware efficiency and test accuracy for better design tradeoffs.

Specifically, our architecture search space and hardware design space co-exploration framework is shown in Figure 3(b). The proposed co-exploration can be built on any existing NAS framework [1, 2, 11, 26] by expanding it to delve into the hardware design space, where a two-level (fast and slow) exploration is iteratively conducted. In the fast exploration, the best hardware design is identified for the sampled neural architectures without lengthy training. The architectures with inferior hardware efficiency will be quickly pruned, which significantly accelerates the search process. Thereafter, the superior candidates are trained in the slow exploration for controller update using policy gradient reinforcement learning to explore the coupled architecture search space. The optimization objectives in the hardware design space can be varied according to the design specifications, such as area, monetary cost, energy efficiency, reliability, resource utilization, etc.

Near Future. The essential objective of NAS is for AI democratization. Although the Co-Design NAS has already significantly pushed forward the progress to automatically implement machine learning tasks on hardware, without considering the constrained hardware resource in edge computing where tons of AI applications are waiting to be deployed, it will easily find inferior solutions or even cannot find feasible solutions. In the following sections, we made innovations on NAS to propose quantization search for resource-constrained hardware on edge.

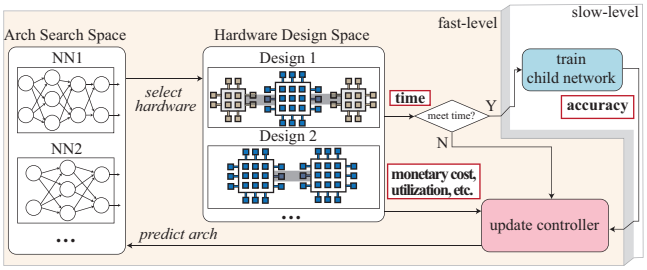


Figure 3: The Co-Design NAS framework.

3 TOMORROW'S NAS: LANDING ON EDGE WITH QUANTIZATION SEARCH FOR RESOURCE-CONSTRAINED HARDWARE

This section will present our framework on NAS and hardware co-design. Specifically, we target on jointly optimizing neural architectures together with their quantization and hardware designs with multiple objectives, which can guarantee the resultant implementation to meet the given specifications.

The overall framework is illustrated in Figure 5. We use the controller to explore the architecture space and the hardware search tool to explore the hardware space. In each episode, the controller samples a child network architecture as well as its quantization scheme. Based on this network, the hardware builder will perform a search procedure through the hardware space for the model of an FPGA-based design. During this process, each candidate model is validated by the design specifications and accordingly the result is used to generate the return to the controller. If any FPGA model is valid, the sampled quantized network is trained on a held-out dataset and feedback the controller with its test accuracy, and otherwise, the return is zero instead. The following sections will reveal the details of this framework.

3.1 Design Space and Parameterization

This paper takes the widely used convolutional neural networks and their FPGA implementation to show the proposed framework, where a serial of stacked convolutional layers are optimized and implemented on an FPGA. The proposed framework will jointly consider three design spaces: architecture space, quantization space and hardware space.

3.1.1 Architecture Space. We consider one neural network layer is composed of a convolutioanl operation followed by a pooling operation. For a convolutional operation, its exploration space can be parameterized, including the number of filters (N), filter height (Fh), filter width (Fw), stride height (Sh), and stride width (Sw). For a pooling operation, we employ the size parameter Ps to indicate its length and stride. As a whole, each layer can be represented by a 6-element sequence: (N, Fh, Fw, Sh, Sw, Ps), and the architectural space of each layer is $\mathcal{A} = \prod_{p \in A} |p|$, where $A = \{N, Fh, Fw, Sh, Sw, Ps\}$, and $|p|$ denotes the number of possible values of a parameter p.

3.1.2 Quantization Space. We apply the quantization to all the trainable parameters and activations in each layer to make tradeoffs between the hardware size and test accuracy. In this paper, we consider a linear quantization with fixed-point representation that is composed of the integer and fractional parts which are taken as separate parameters in our framework.

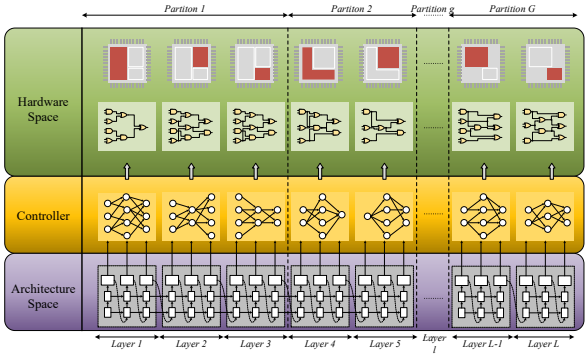


Figure 4: Overview of the proposed exploration framework.

Assuming the rectified linear unit (ReLU) as the activation function, the output A of the convolutional layer is non-negative, and we apply the unsigned quantization as

$$Q(A) = clip(round(\frac{A}{\Delta q}) \times \Delta q,\ 0,\ B - \Delta q). \tag{1}$$

where Δq is the precision and B is the range amplitude, both of which are determined by the bit width in the integer part Ai and fractional part Af, respectively. We conclude their relationship as: $B = 2^{Ai}$, $\Delta q = 2^{-Af}$.

For the weight and bias parameters W, signed quantization is applied, such that

$$Q(W) = clip(round(\frac{W}{\Delta q}) \times \Delta q,\ -B,\ B - \Delta q), \tag{2}$$

where we have the relationship between Δq, B, Wi and Wf as follows: $B = 2^{Wi-1}$, $\Delta q = 2^{-Wf}$.

Similarly to the architecture parameterization, the quantization scheme can be represented by $Q = \{Ai, Af, Wi, Wf\}$ and thus the quantization space is $\mathcal{Q} = \prod_{p \in Q} |p|$.

3.1.3 Hardware Space. Given a determined architecture together with its quantization, the implementation varies in terms of two aspects: intra-layer parallelism (single-layer accelerator design) and inter-layer parallelism (mapping accelerators to an FPGA).

For the single-layer accelerator design, we adopted the widely used tile-based paradigm [24]. We represent tiling parameters as a sequence of functions with $Tm(M)$ as the number of channels of the input tiles, $Tn(N)$ as the number of channels of the output tiles, $Tr(R)$ as the height of input tile, and $Tc(C)$ as the width of the input tile, where M, R and C are the number of channels, rows, and columns of the input feature maps, respectively.

For the mapping of single-layer accelerators to an FPGA, we partition and allocate hardware resources to accelerators. The partition scheme P, as a function of L, is a selection from all the combinations of the L layers clustered into any number of sections from 1 to L, which results in a space consisting of 2^{L-1} candidates. The hardware space is then represented by $\mathcal{H}(L) = 2^{L-1} \prod_{p \in H} |p|$, where $H = \{Tm, Tn, Tc, Tr\}$.

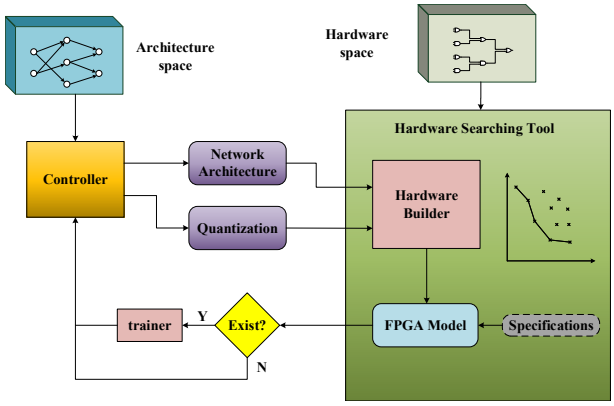


Figure 5: Hardware-architecture co-design framework.

3.1.4 Overall. The proposed framework will jointly determine architecture, quantization, and tiling parameters, together with partition of layers, to identify the neural architecture and hardware implementation, such that both test accuracy and hardware efficiency can be maximized.

We will use the reinforcement learning method to explore architecture and quantization spaces, and develop a multi-objective search algorithm to explore the hardware space. More specifically, there is a controller to control the exploration, as shown in Figure 4. Details will be discussed in the following sections.

3.2 Update the Controller

The architecture and quantization parameters are both optimized to generate high accuracy. As shown in Figure 5, we employed reinforcement learning method to explore the space $\mathcal{A}$ and $\mathcal{Q}$ where the controller interacts with the environment modeled as a Markov Decision Process (MDP). In each episode, the controller rolls out a sequence of actions under a stochastic policy. These actions, used as the architecture parameters A and quantization parameters Q, are mapped to a *quantized* child network. Next, we evaluate the sampled child network in two stages. In the first stage a hardware searching tool is developed to verify whether the sampled network is implementable under the constraints of design specifications. If the result is positive, i.e. there exists an implementable hardware model, and the second stage will launch to train and validate the child network on a held-out dataset. After the child network validation is finished, a reward signal

$$R(a,q) = \begin{cases} 0, & \mathcal{H}(a,q) = \varnothing \\ Acc, & \text{otherwise} \end{cases} \tag{3}$$

is returned to the controller for updating. In the above formula, $\mathcal{H}(a,q)$ represents the hardware space given sampled parameters a and q. We follow the Monte Carlo policy gradient algorithm [16] to update the controller using

$$\nabla J(\theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\gamma^{T-t}\nabla_\theta \log \pi_\theta(a_t|a_{(t-1):1})(R_k - b) \tag{4}$$

where m is the batch size and T is the total number of steps in each trajectory. The rewards are discounted at every step by an exponential factor γ and the baseline b is the exponential moving average of the rewards.

3.3 Co-Explore Architecture and Quantization

Much like the architectures, the quantization also determines the overall performance and computational complexity of a network. Therefore, it is natural to automate the design of quantization together with the design of the architecture. Under constrained resource in hardware, the joint exploration of architecture and quantization space is actually the optimization of the trade-off between structural complexity and data cleanness. Therefore, the reward signal is the reflection of how efficiently the hardware freedom are utilized.

The implementation of architecture-quantization joint search may vary by different settings and discretion, but generally there are two types of methods characterized by the number of controller used: a single controller to predict both architecture and quantization, or two controllers to predict them separately. In this paper, we focus on the single-controller method and extend the RNN-based controller in [26]. As displayed in Figure 6, we simply insert 4 additional steps into the controller, each step sampling one of the aforementioned quantization parameters.

For comparison, we list all the plausible methods for performing the automate design of neural architecture with quantization. The difference among these methods reside in the space to be explored.

- *quantization* search. This is the traditional method that employs the controller to search only the quantization for a given architecture.
- *architecture* search. This is the reverse procedure of quantization search, where the quantization is fixed and the objective is to find the architecture to best fit this quantization.
- *joint* search. This is the exploration of architecture and quantization as related space and what we intend to investigate in this project.

3.4 Explore Hardware Space

We revisit the hardware space discussed in Section 3, this time with consideration of our hardware model. In our specific model, the specifications are in the LUT usage of the target FPGA and the throughtput in frames-per-second, though the framework can be adapted to other aspects without much effort. In particular, only Tn and Tm are variable while Tr and Tc are dropped due to their irrelevance to the target specifications. The other parameters such as quantizations are given as constant instead of variable. As a result, the actual hardware space to explore is

$$\mathcal{H}(L) = 2^{L-1}\prod_{i=1}^{L} N_{i-1}N_i \tag{5}$$

whose size grows exponentially with L. To avoid exploring $\mathcal{H}$ exhaustively, we have developed an efficient searching algorithm using dynamic programming. The hardware model and searching algorithm are explained in details as follows.

3.4.1 Tile-Based Implementation. In the tile-based model, the main processing unit is the quantized computation engine (QCE) which is composed of an array of multipliers, an adder tree, a truncator, and the accumulation registers (Figure 7). For implementation on the FPGAs, the consumption of lookup-tables (LUTs) by each

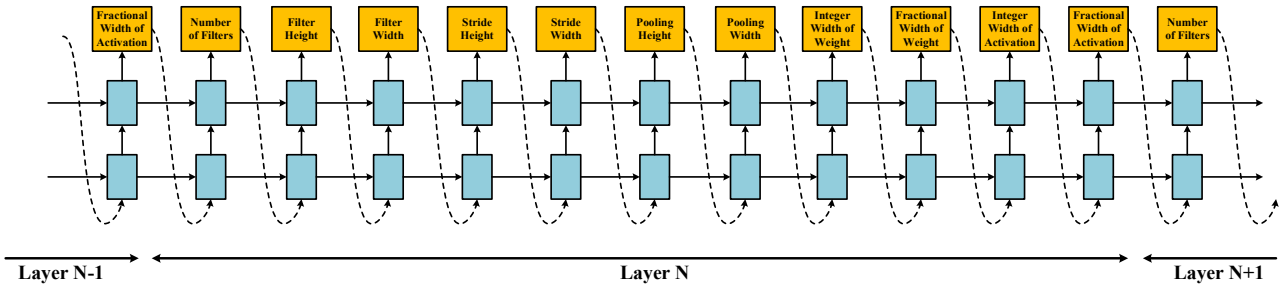


Figure 6: The controller in sync search samples both architecture and quantization parameters layer by layer.

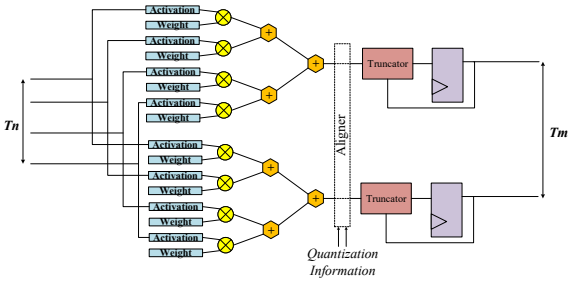


Figure 7: RTL architecture of quantized computing engine.

QCE scales with Tn, Tm, and the bit-width of the layer as it configures the size of the above components. Due to the data format inconsistency between the activation and weight (inter-layer), and the inconsistency between activations of consecutive layers (intra-layer), we customize the QCEs in each layer, according to the 6 parameters Wi, Wf, Ai, Af, Ai', and Af', where Ai', and Af' are the bitwidth of the activation from the last layer. For the inter-layer inconsistency, the problem is handled by informing the multipliers and adders of the number of integer and fractional bits of each operand. This makes no difference to the fixed-point multipliers as the only effect is on the position of the decimal point. As for the adders, this information means to perform data alignment by specifically extending the MSB (integer part) and LSB (fractional part) to certain numbers, which however does not incur any extra logic. On the other hand, the intra-layer consistency involves truncating the partial sum produced by the adder tree and tailoring the registered result to a target format. This inconsistency directly affects the truncator in size.

With the above model, the total size and latency of a single-layer accelerator can be approximated. As mentioned, the size of layer l is a function of Tn, Tm, and bit width of weight and activations, incoming and outgoing, i.e.

$$Lut_i = qce(Tn_i, Tm_i, Ai_{i-1}, Af_{i-1}, Ai_i, Af_i, Wi_i, Wf_i), \quad (6)$$

where qce is the LUT approximator of the QCE for FPGAs and is predefined as a library. Besides, the latency of the single-layer accelerator can be explicitly approximated by computation as

$$Lat_i \approx \left\lceil \frac{M_i}{Tm_i} \right\rceil \times \left\lceil \frac{N_i}{Tn_i} \right\rceil \times R_i \times C_i \times Fh_i \times Fw_i. \quad (7)$$

Equation (6) and (7) are used to calculate the LUT usage and latency of a single-layer accelerator, upon which our multi-layer accelerator model is based. If a multi-layer partition g contains consecutive layers from i to j operating in a pipelined fashion, then we have the overall size and latency as

$$Lut_{g:i\sim j} = \sum_{k=i}^{j} Lut_k, \; Lat_{g:i\sim j} = \max_{i\le k\le j} Lat_k. \quad (8)$$

Suppose a number of G partitions covering a total of L layers iterate their operations on the same FPGA, the total LUT usage and latency are then

$$Lut_{1\sim G:1\sim L} = \max_{1\le g\le G} Lut_{g:i\sim j}, \; Lat_{1\sim G:1\sim L} = \sum_{g=1}^{G} Lat_{g:i\sim j} \quad (9)$$

3.4.2 Searching Algorithm. As implied by (5), the problem of searching the hardware space $\mathcal{H}$ involves deciding parameters $Tn \in [1, N]$, $Tm \in [1, N']$, as well as partitioning the L layers into $G \in [1, L]$ clusters. Let rL and rT be the required LUT usage and throughput limits by the design specifications, we introduce $\mathbf{P}_L^{\langle rL, rT\rangle}$ to represent both this problem and solution set to this problem: given specification pair $\langle rL, rT\rangle$, $\mathbf{P}_L^{\langle rL, rT\rangle}$ returns all the possible solutions for implementing the CNN accelerator of L layers. The task of our searching algorithm is to verify whether $\mathbf{P}_L^{\langle rL, rT\rangle} = \varnothing$. In order to address this task, we also need to introduce the single-layer search problem as the basic tool: we use $\mathbf{p}_l^{\langle rL, rT\rangle}$ to represent the problem of searching for the hardware solutions to a single layer l under the constraint of $\langle rL, rT\rangle$ and again its solution set. We shall solve the problem $\mathbf{P}_L^{\langle rL, rT\rangle}$ by incrementally solving the basic problem $\mathbf{p}_i^{\langle rL', rT'\rangle}$, from $i = 1$ to L.

For any solution $s \in \mathbf{P}_L^{\langle rL, rT\rangle}$, we define three functions $f_1(s)$, $f_2(s)$ and $f_3(s)$ respectively as 1) the number of LUTs consumed by the last partition of s, 2) the overall latency of s, and 3) the sum of latency of all the partitions in s except the last one. For any two solutions s_1 and s_2, if we have $f_1(s_1) \le f_1(s_2)$, $f_2(s_1) \le f_2(s_2)$, and $f_3(s_1) \le f_3(s_2)$, then s_1 is considered superior to s_2, and all the solutions that is not inferior to any other solution compose the Pareto Frontier of the solution set. Our algorithm is based on the fact that the existence of a solution is equivalent to the existence of the frontier of the solution set. Following this observation, we search for the frontier of $\mathbf{P}_L^{\langle rL, rT\rangle}$ such that the space is significantly pruned. If the network has a depth of $l + 1$, there are only two scenarios of how layer $(l + 1)$ is related to the first l layers:

Algorithm 1 Dynamic Search in Hardware Space

Input: L, rL, rT
Output: solution set $\mathbf{S}$

```
Initialize rl = rL, rt = rT,
Initialize S_0 = {s} where f_i(s) = 0 for i=1, 2, 3
for each l = 1, 2, ..., L do
    for each s ∈ S_{l-1} do
        compute rl and rt using Equation (10)
        compute s_l = F(p_l^{⟨rl,rt⟩})
        for all s' ∈ s_l do
            append s' to the last partition of s and add the result to S_l
        end for
        compute rl and rt using Equation (11)
        compute s_l = F(p_l^{⟨rl,rt⟩})
        for all s'' ∈ s_l do
            set s'' as a new partition to s and add the result to S_l
        end for
        update the frontier S_l = Fr(S_l)
    end for
end for
S = S_L
```

(1) layer (l+1) is appended to the last partition of the previous l layers, and
(2) layer (l+1) forms a partition itself.

These two scenarios differ by the constraints to the problem of searching the last layer $\mathbf{p}_{l+1}^{\langle rl,rt\rangle}$. For every solution $s \in Fr(\mathbf{P}_l^{\langle rl,rt\rangle})$, where $Fr(S)$ denotes the frontier of solution set S, we update the layer $l+1$ in both two manners under updated constraints. In the first case,

$$\begin{cases} rl &= rL - f_1(s) \\ rt &= (\frac{1}{rT} - \frac{f_3(s)}{clock\_rate})^{-1} \end{cases} \tag{10}$$

and in the second case,

$$\begin{cases} rl &= rL \\ rt &= (\frac{1}{rT} - \frac{f_2(s)}{clock\_rate})^{-1} \end{cases} \tag{11}$$

When $\mathbf{p}_{l+1}^{\langle rl,rt\rangle}$ is solved, it would also be sufficient to keep only its frontier with respect to only f_1 and f_2. Then $Fr(\mathbf{p}_{l+1}^{rl,rt})$ is achieved by combining $Fr(\mathbf{P}_l^{\langle rl',rt'\rangle})$ and $F(\mathbf{p}_{l+1}^{\langle rl,rt\rangle})$ in the corresponding way of how layer $l+1$ is derived. The full procedure is shown in Algorithm 1.

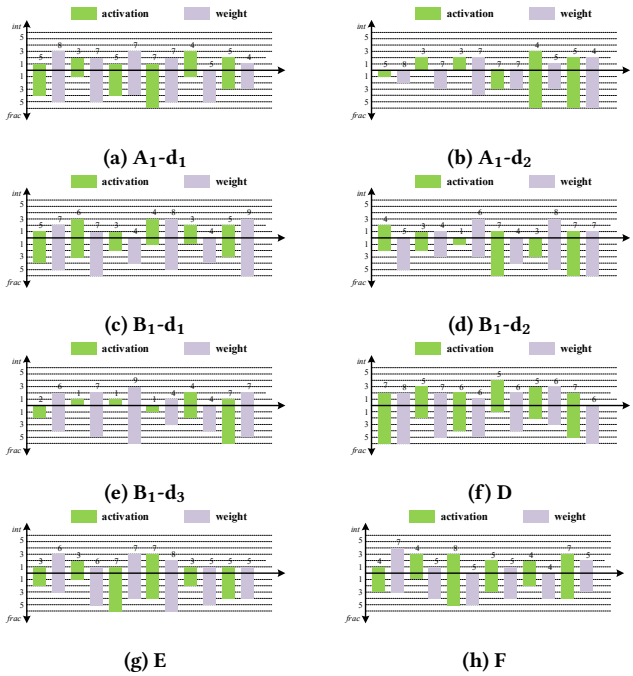
4 EXPERIMENT

We apply the joint architeture and quantization search method to design CNN accelerators. The design objective is the classification task on CIFAR-10 dataset that satisfies two constraints of available LUTs and throughput tolerance. This dataset provides 50000 images for training and 10000 for testing, and the entirety of them are used in the search process. For the training set, augmentation techniques are applied for tuning a network which consists of normalization, rotation, shifting, and random flip.

The architecture and quantization space used in the experiment is listed in Table 1. For the child network, we assume each layer is composed of certainly a convolutional operation followed by rectified linear units, and possibly zero-padding and max-pooling operations before and after it. After the convolutional layers, two fully connected layers are tailing to output the prediction distributions, which are not included as part in our hardware model. We train the child CNN for 30 epochs with Stochastic Gradient Decent (SGD) algorithm taking batches of 128 images, learning rate of 0.01, and momentum of 0.9. Once the training is finished, the reward is the test accuracy averaged over the last 5 epochs. After the search, the best sample are selected and tuned for 150 epochs, along with 64 batch size and decaying learning rates from 0.01 downward to 0.0001. The highest accuracy along the tuning process is what is finally reported. On the other side, the controller consists of a two-layer LSTM cell with 35 hidden units at each layer accompanied with an embedding and fully-connected layer at each time step of corresponding dimensions. To train the controller, we apply the

Table 1: The architecture and quantization space of each CNN layer used in the experiment

Parameter	Symbol	Value
# filters	N	(24, 36, 48, 64)
filter height	Fh	(1, 3, 5, 7)
filter width	Fw	(1, 3, 5, 7)
stride height	Sh	(1, 2, 3)
stride width	Sw	(1, 2, 3)
pooling size	Ps	(1, 2)
activation integer bits	Ai	(0, 1, 2, 3)
activation fractional bits	Af	(0, 1, 2, 3, 4, 5, 6)
weight integer bits	Wi	(0, 1, 2, 3)
weight fractional bits	Wf	(0, 1, 2, 3, 4, 5, 6)



(a) A_1-d_1 (b) A_1-d_2 (c) B_1-d_1 (d) B_1-d_2 (e) B_1-d_3 (f) D (g) E (h) F

Figure 8: Quantization details of the sampled designs

Table 2: Architectural information of the sampled designs. A_1 and A_2 are the best architectures found by NAS in 1000 episodes and their layered hyper-parameters are given in the form of (N, Fh, Fw, Sh, Sw, Ps). Then we remove the strides from the search and get B_1 and B_2 whose parameters of each layer are listed as (N, Fh, Fw, Ps). D, E, and F are the results from our joint search process with the same architecture space but no strides.

Network	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	#paras	Acc w/o BN	Acc w/ BN
A_1	(64,3,3,1,1,1)	(48,7,5,1,1,1)	(48,5,5,2,1,1)	(64,3,5,1,1,1)	(36,5,7,1,1,1)	(64,3,1,1,2,2)	300,804	87.76%	88.96%
A_2	(24,3,3,1,1,1)	(36,5,5,1,1,1)	(64,5,5,2,1,1)	(64,5,5,1,1,1)	(24,5,5,1,2,1)	(64,3,3,1,2,1)	234,748	87.46%	88.87%
B_1	(64,3,3,1)	(64,3,5,1)	(64,3,3,2)	(64,5,5,2)	(64,5,3,1)	(64,7,7,1)	464,960	89.71%	90.30%
B_2	(64,5,3,1)	(64,3,5,1)	(64,3,5,2)	(64,5,5,2)	(64,5,3,1)	(64,7,7,1)	490,688	89.38%	90.49%
D	(48,5,3,1)	(48,3,1,2)	(36,1,7,2)	(36,7,3,1)	(24,5,5,1)	(24,1,1,1)	70,776	83.65%	84.31%
E	(48,5,1,1)	(48,5,3,2)	(36,1,5,1)	(64,7,7,2)	(64,7,3,2)	(48,5,3,1)	289,220	86.99%	88.27%
F	(64,1,5,1)	(36,1,7,1)	(64,5,7,2)	(48,5,3,2)	(48,7,7,1)	(36,1,5,1)	26,5640	87.03%	88.42%

Table 3: Implementation information of the sampled designs. For network A and B, the designs are found by quantization search to certain architectures in Table 2. For D, E and F, the quantization and implementation on hardware are designed together with their architectures. The quantization details are shown in Figure 4.

Design	rL	rT	Acc w/o quantization	Acc w/ quantization	#LUTs	Throughput (frames/s)	parameter size (kbits)
A_1-d_1	100,000	500	87.76%	80.23%	99,871	556	1,867
A_1-d_2	100,000	1000	87.76%	25.79%	99,848	1157	1,189
B_1-d_1	100,000	500	89.71%	87.64%	96,904	512	3,463
B_1-d_2	100,000	1000	89.71%	64.35%	98,752	1020	2,784
B_1-d_3	300,000	2000	89.71%	50.93%	285,441	2083	2,835
D	30,000	1000	83.65%	82.98%	29,904	1293	457
E	100,000	1000	86.99%	82.76%	94,496	1042	1,923
F	300,000	2000	87.03%	84.92%	299,860	2089	1,217

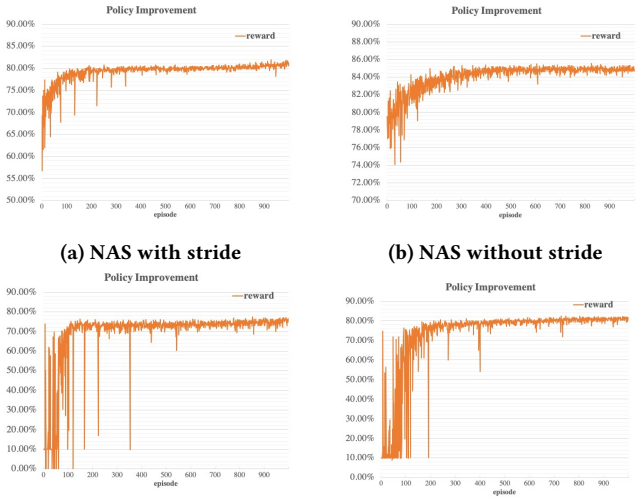
Stochastic Gradient Ascent (SGA) algorithm with a learning rate of 0.2 and batch size of 5. The baseline is the exponential moving average of the previous rewards. Finally, we build the hardware model based upon an Altera Cyclone IV FPGA platform where we set the global clock rate as 100 MHz. In order to be build practical FPGA synthesis, we set the depth of the child network to have 6 layers, and designate the allowable LUT usage at three scales: 30,000, 100,000, and 300,000.

For comparison, we first perform the NAS to find architectures with good performances in accuracy and then search the best quantization to fit them under some hardware specifications. The sampled networks with highest accuracy on the test set and their best design with quantization are shown in Table 2 and 4, respectively. We use the floating-point accuracy in Table 2 as the reference of our design of quantization. Note that the same architecture is about 1% less performant without the batch normalization (BN), but that is just what we shall refer to. The reason is twofold: 1) the BN involves operations that require additional hardware support for both computation and memory, and more importantly 2) it will make the quantized network extremely unstable whose output may have a prohibitively large variance for the searching algorithm. Next, we use the joint method to search architecture and quantization together with the specifications under which the quantization search fails to provide a good result in a general sense with respect to the best architectures found. As a result, we have three designs of the CNN accelerators with different hardware specifications. The details of these designs are reported in Table 3.

Table 4: Quantization search result for the sampled networks A and B in Table 2. The best accuracy on the test set of CIFAR10 in 2000 episodes are reported.

Network	rT	rL=30,000	rL=100,000	rL=300,000
A_1	500	10.65%	80.23%	86.16%
	1000	x	25.79%	84.90%
	2000	x	x	x
A_2	500	55.45%	85.92%	86.26%
	1000	x	67.30%	76.51%
	2000	x	x	x
B_1	500	10.02%	87.64%	87.34%
	1000	x	64.35%	87.43%
	2000	x	x	50.93%
B_2	500	10.20%	85.31%	88.53%
	1000	x	43.81%	86.50%
	2000	x	x	16.71%

Table 4 shows the quantization search results with throughput requirement of 500, 1000, and 2000 frames per second. It is clearly noted the drop in accuracy with increasing throughput is very sharp. For example, the accuracy of the quantized network B_1 drops from 87.64% to 64.35% with doubling 500 MHz throughput requirement at rL=100,100. It is implied although the architecture has an excellent original performance, it is too sensitive to quantization to be suitable for resource-constrained hardware design. On the other hand, our joint search method has found a solution to achieve



(a) NAS with stride (b) NAS without stride

(c) Joint: rL=30000, rT=500 (d) Joint: rL=300000, rT=500

Figure 9: The plotted training process: comparison between NAS and Joint search.

82.76% accuracy and meanwhile the 1000 throughput. In contrast, the original accuracy of architecture E is only 86.99%, worse than B_1 by nearly 3%, but the its accuracy is more robust to quantization with 4% degradation. With the joint search, we could even found a design using less than 30,000 LUTs but achieving 82.98% accuracy and 1293 throughput. Note there are no valid designs for almost every sample architecture even with 300,000 available LUTs.

We further compare the quantization designs for A_1 and B_1 with those using joint search for D, E, and F. As illustrated in Figure 8, the convolutional layers generally exhibit different patterns in terms of bit-width requirement. Another observation is with fixed architecture, the quantization search tends to spend more bits on the weight but not the activations, while the joint search treats these two values fairly.

5 CONCLUSION AND CHALLENGE

In this paper, we overviewed the recent development of automatic machine learning, identifying the trend towards the hardware-software co-design using NAS. A hardware-aware co-design framework is proposed to jointly explore architecture, quantization, and hardware design space. It is proved by experiment the joint search can provides much more flexibility in compressed design robust performance as compared to the traditional artificial design using fixed architecture.

In this project, however, the existence of difficulties in applying hardware-aware NAS is also identified. Compared to pure NAS, the controller is forced the burden to learn the hardware constraint from the beginning of the search process, resulting in a higher variance and early convergence to local optimality (Figure 9). On the other hand, the search process is computation-intensive and resource-/time-consuming. Lastly, the hardware exploration and design automation heavily rely on the hardware model that needs to be built with more effort. There remains a lot of room of improvement in this topic.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under Grant CCF-1820537 and CNS-1822099.

REFERENCES

[1] Gabriel Bender et al. 2018. Understanding and simplifying one-shot architecture search. In *Int. Conf. on Machine Learning*. 549–558.
[2] Han Cai et al. 2018. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).
[3] Weiwen Jiang et al. 2016. Optimal functional-unit assignment and buffer placement for probabilistic pipelines. In *2016 Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 1–10.
[4] Weiwen Jiang et al. 2017. Optimal functional unit assignment and voltage selection for pipelined MPSoC with guaranteed probability on time performance. In *ACM SIGPLAN Notices*, Vol. 52. ACM, 41–50.
[5] Weiwen Jiang et al. 2018. Heterogeneous fpga-based cost-optimal design for timing-constrained cnns. *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst* 37, 11 (2018), 2542–2554.
[6] Weiwen Jiang et al. 2019. Accuracy vs. Efficiency: Achieving Both through FPGA-Implementation Aware Neural Architecture Search. In *Proc. 56th Annual Design Automation Conference 2019*. ACM, 5.
[7] Weiwen Jiang et al. 2019. Hardware/software co-exploration of neural architectures. *arXiv preprint arXiv:1907.04650* (2019).
[8] Weiwen Jiang et al. 2019. XFER: a novel design to achieve super-linear performance on multiple FPGAs for real-time AI. In *Proc. of Int. Symp. on FPGA*. ACM, 305–305.
[9] David Koeplinger et al. 2016. Automatic generation of efficient accelerators for reconfigurable hardware. In *2016 ACM/IEEE 43rd Annual Int. Symp. on Comput. Archit. (ISCA)*. IEEE, 115–127.
[10] Boyang Li et al. 2019. Exploiting computation power of blockchain for biomedical image segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshops*.
[11] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[12] Research and Markets. 2018. Artificial intelligence in IoT: AIoT Technology, Platforms, Applications and Services by Industry Vertical 2018 - 2023. *Report* (2018).
[13] Mingxing Tan et al. 2018. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626* (2018).
[14] Tianchen Wang et al. 2019. MSU-Net: multiscale statistical U-Net for real-time 3D cardiac MRI video segmentation. In *Proc. of Medical Image Computing and Computer Assisted Interventions (MICCAI)*. 0–0.
[15] Tianchen Wang et al. 2019. SCNN: a general distribution based statistical convolutional neural network with application to video object detection. In *AAAI Conf. on AI*.
[16] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
[17] Bichen Wu et al. 2018. FBNet: hardware-qware efficient convNet design via differentiable neural architecture search. *arXiv preprint arXiv:1812.03443* (2018).
[18] Xiaowei Xu et al. 2017. Edge segmentation: empowering mobile telemedicine with compressed cellular neural networks. In *Proc. of the 36th Int. Conf. on Computer-Aided Design*. IEEE Press, 880–887.
[19] Xiaowei Xu et al. 2018. Efficient hardware implementation of cellular neural networks with incremental quantization and early exit. *ACM J. on Emerging Technologies in Computing Systems (JETC)* 14, 4 (2018), 48.
[20] Xiaowei Xu et al. 2018. Quantization of fully convolutional networks for accurate biomedical image segmentation. *Preprint at https://arxiv. org/abs/1803.04907* (2018).
[21] Xiaowei Xu et al. 2018. Resource constrained cellular neural networks for real-time obstacle detection using FPGAs. In *2018 19th Int. Symp. on Quality Electronic Design*. IEEE, 437–440.
[22] Xiaowei Xu et al. 2018. Scaling for edge inference of deep neural networks. *Nature Electronics* 1, 4 (2018), 216.
[23] Xiaowei Xu et al. 2019. Whole-heart and great vessel segmentation in congenital heart disease using deep neural networks and graph matching. In *Proc. of Medical Image Computing and Computer Assisted Interventions (MICCAI)*. 0–0.
[24] Chen Zhang et al. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proc. of FPGA*. ACM, 161–170.
[25] Xinyi Zhang et al. 2019. When Neural Architecture Search Meets Hardware Implementation: from Hardware Awareness to Co-Design. In *Proc. of ISLVLSI*. 25–30.
[26] Barret Zoph et al. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).
[27] Barret Zoph et al. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conf. on computer vision and pattern recognition*. 8697–8710.