Received July 27, 2020, accepted August 18, 2020, date of publication September 7, 2020, date of current version September 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3022327

APNAS: Accuracy-and-Performance-Aware Neural Architecture Search for Neural Hardware Accelerators

PANITI ACHARARIT[1],
MUHAMMAD ABDULLAH HANIF[2], (Graduate Student Member, IEEE),
RACHMAD VIDYA WICAKSANA PUTRA[2],
MUHAMMAD SHAFIQUE[3], (Senior Member, IEEE),
AND YUKO HARA-AZUMI[1], (Member, IEEE)
[1]Department of Information and Communications, Tokyo Institute of Technology, Tokyo 152-8550, Japan
[2]Institute of Computer Engineering, Faculty of Informatics, Vienna University of Technology, 1040 Vienna, Austria
[3]Division of Engineering, New York University Abu Dhabi (NYU AD), Abu Dhabi 129188, United Arab Emirates

Corresponding author: Paniti Achararit (paniti@cad.ict.e.titech.ac.jp)

This work was supported by JSPS KAKENHI under Grant 20H04154.

ABSTRACT Designing resource-efficient deep neural networks (DNNs) is a challenging task due to the enormous diversity of applications as well as their time-consuming design, training, optimization, and evaluation cycles, especially the resource-constrained embedded systems. To address these challenges, we propose a novel DNN design framework called accuracy-and-performance-aware neural architecture search (APNAS), which can generate DNNs efficiently, as it does not require hardware devices or simulators while searching for optimized DNN model configurations that offer both inference accuracy and high execution performance. In addition, to accelerate the process of DNN generation, APNAS is built on a weight sharing and reinforcement learning-based exploration methodology, which is composed of a recurrent neural network controller as its core to generate sample DNN configurations. The reward in reinforcement learning is formulated as a configurable function to consider the sample DNNs' accuracy and cycle count required to run on a target hardware architecture. To further expedite the DNN generation process, we devise analytical models for cycle count estimation instead of running millions of DNN configurations on real hardware. We demonstrate that these analytical models are highly accurate and provide cycle count estimates identical to those of a cycle-accurate hardware simulator. Experiments that involve quantitatively varying hardware constraints demonstrate that APNAS requires only 0.55 graphics processing unit (GPU) days on a single Nvidia GTX 1080Ti GPU to generate DNNs that offer an average of 53% fewer cycles with negligible accuracy degradation (on average 3%) for image classification compared to state-of-the-art techniques.

INDEX TERMS Neural architecture search, neural processing arrays, embedded systems, accelerator, performance, accuracy, efficiency, machine learning, deep learning, DNN, deep neural networks, CNN, convolutional neural network.

I. INTRODUCTION

The accuracy offered by state-of-the-art deep neural networks (DNNs) has led to their use in a variety of artificial intelligence applications, including object detection, speech recognition, event detection, machine translation, and autonomous driving [1]–[10]. Advancements in DNNs coupled with the evolution of specialized DNN hardware accelerators have led to the application of high-accuracy DNNs not only in cloud-based services, but also in a number of embedded applications such as autonomous driving, robotics, surveillance, and wearable healthcare [11]–[13]. However, due to large differences in resource availability with the cloud-based systems, deploying DNNs on embedded systems requires redesigning the DNNs with several optimizations, such as (iterative) pruning and quantization with

The associate editor coordinating the review of this manuscript and approving it for publication was Byung Cheol Song.

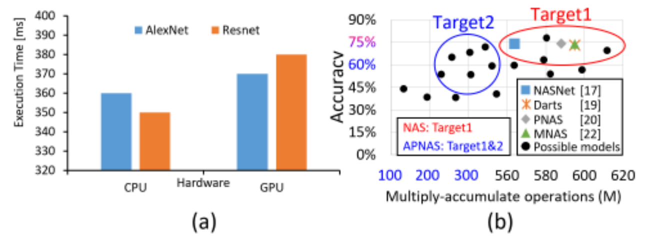


FIGURE 1. Motivational examples for proposed accuracy-and-performance-aware neural architecture search (APNAS) framework (a) Execution time required to process AlexNet and ResNet-50 on a CPU and GPU with a batch size of 64 (data taken from [24]); (b) overview of design space exploration by conventional neural architecture search techniques (e.g., [17], [19], [20], [22]) and proposed APNAS method.

potentially multiple rounds of re-training, to meet the constraints of the underlying hardware architectures [14]–[16]. Redesigning DNNs is a time-consuming task, as designers must explore a large number of parameters to achieve both satisfactory accuracy and real-time processing (i.e., the short execution time [15]).

Up to now, many specialized DNNs have been designed that offer high accuracy for specific application domains. For example, convolutional neural networks (CNNs) are designed for image classification and object detection tasks, while recurrent neural networks (RNNs) are designed for speech recognition tasks. Even among specialized DNNs, there exist multiple sub-types of DNNs, and there is no golden rule for selecting a DNN to achieve the highest accuracy. Moreover, each DNN has numerous hyper-parameters (e.g., number of layers and number of filters in each layer) that can be tuned to achieve higher accuracy. Therefore, several neural architecture search (NAS) techniques have been proposed to efficiently explore this enormous design space based on various search algorithms, such as reinforcement learning (RL) [17], [18], gradient descent [19], and Bayesian optimization [20].

These studies demonstrated that automatically generated DNNs can outperform several manually designed DNNs in terms of accuracy. However, because most studies on NAS [17]–[20] considered only accuracy, the generated DNNs may be computationally expensive. This may result in a long execution time, which is particularly problematic for resource-constrained embedded platforms. Although several recent studies on NAS considered both accuracy and computation amount (or execution time) [21]–[23], they required time-consuming evaluation by a hardware device or simulator, resulting in a very long exploration time, which can significantly delay design and deployment cycles. This is especially problematic in the context of tight time-to-market requirements and escalating competition. As a result, there is a need for analytical models that estimate the computation amount of the target hardware and that can be integrated in an efficient NAS framework.

The focus on NAS techniques in this study is motivated by the fact that different DNN designs have different computational efficiency on different hardware architectures even with similar accuracy. For example, Fig. 1(a) presents the execution time required to process two well-known DNNs (AlexNet and ResNet-50) on a central processing unit (CPU; Intel Xeon E5-2687W) and graphics processing unit (GPU; NVIDIA Tesla K80). As illustrated in this figure, AlexNet is faster than ResNet-50 on a CPU, whereas ResNet-50 is faster than AlexNet on a GPU. From these results, it can be inferred that designing a single unified DNN for all hardware platforms is not feasible.

Moreover, to achieve the highest accuracy while satisfying user-defined constraints, DNNs must be designed considering the architectural characteristics of the underlying hardware devices. Clearly, using manual DNN designs to achieve not only high accuracy but also computational efficiency for each combination of target application and underlying hardware architecture is impractical. To address the aforementioned challenges, it is essential to develop *a new hardware-aware NAS technique that can efficiently evaluate both accuracy and execution time to accelerate DNN exploration while encompassing the diversity of applications and underlying (resource-constrained) hardware architectures. Furthermore, to expedite the search process, it is necessary to employ analytical models for estimating the computation amount of the target hardware.*

In this paper, we propose a novel NAS framework called *accuracy-and-performance-aware neural architecture search (APNAS)*, which considers both accuracy and performance[1] to automatically generate DNNs suitable for neural hardware accelerators in resource-constrained embedded systems without the need for hardware devices or time-consuming simulators. Our framework leverages the features of efficient neural architecture search (ENAS) [18], which is a framework for reducing the search time for NAS. As a result, we are able to perform a fast search for DNNs.

This work focuses on neural processing array-based hardware architectures, as they have been intensively studied to efficiently accelerate DNNs [25]–[29]. We focus mainly on CNNs, which are widely used owing to their remarkable accuracy in image classification and recognition applications [30]–[33].

Fig. 1(b) illustrates the main differences in design goals between conventional NAS techniques [17]–[20] and our APNAS framework. Whereas conventional NAS techniques mainly aim to obtain models with high accuracy (i.e., "Target1" in the figure), our APNAS framework aims to obtain models that achieve fewer computations with comparable accuracy to models sought by conventional NAS (i.e., both "Target1" and "Target2" in the figure). In APNAS, we formulate the NAS based on RL, which utilizes a RNN controller as its core. The RNN is responsible for generating sample DNNs, which are then evaluated to compute the reward in RL. To define this reward, we use the cycle count required to execute a CNN on a neural processing array-based hardware accelerator as the performance metric in addition to

[1]Hereafter, we use performance to indicate the execution time.

the accuracy. To estimate the cycle count, we propose new analytical models based on the characteristics of the underlying hardware architecture and the CNN model. These characteristics enable accurate estimation without the need for time-consuming hardware simulations. A tunable parameter is introduced in the reward to enable a trade-off between accuracy and performance according to the system/user requirements. It is also worth noting that our framework can model hardware constraints and can thus be easily applied to neural processing array-based hardware architectures of different sizes.

In our evaluation, we generated CNNs by APNAS for two different image classification datasets (CIFAR-10 and CIFAR-100) for two different hardware resource constraints with a neural processing array size of 8×8 and 16×16. We demonstrated that APNAS successfully generated CNNs that were aware of both validation accuracy and computational complexity. When compared to a manually designed CNN (i.e., ResNet [4]), APNAS generated CNNs that required on average 52.78% and 53.57% fewer cycles with only 3.43% to 2.65% average accuracy degradation on the CIFAR-10 and CIFAR-100 datasets, respectively. Moreover, APNAS required only 0.55 GPU days on Nvidia GTX 1080Ti GPU to search for the model, which is significantly faster than the performance of state-of-the-art NAS techniques. Through extensive evaluations using different parameter settings for the reward function and in-depth analyses of the results, we also provide useful findings and a discussion of the effects of the parameter settings on the trade-off between the accuracy and performance of the generated CNNs.

In summary, the main features and contributions of APNAS are as follows:

- APNAS can efficiently search for CNNs for the target hardware architecture from two aspects. First, unlike existing NAS techniques that utilize time-consuming evaluation on the target hardware device or its simulator, our method does not require such evaluation. Instead, we propose an analytical model that provides an abstract yet accurate estimation of the performance (i.e., cycle count) required for the inference of a CNN on neural processing array-based hardware. Second, among a variety of NAS techniques, our method also leverages features from approaches such as ENAS to accelerate the exploration. Integrating these techniques makes it possible to search for hardware-aware CNNs faster than other hardware-aware NAS techniques.
- In the reward function of the RNN controller, we introduce a tunable parameter to configure the trade-off between the accuracy and the performance of the generated DNNs.
- We present useful findings and observations by varying different parameters that construct our proposed models. By analyzing the generated CNNs in detail, we discuss potential improvements for efficiently exploring a wider design space to achieve a better accuracy/performance trade-off in a reasonable exploration time.

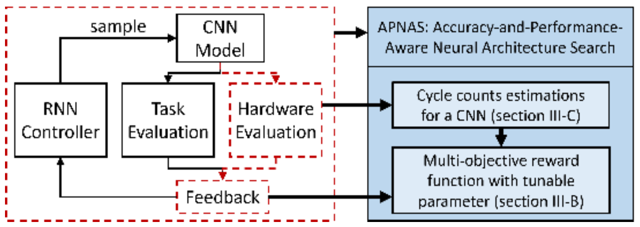

sample
CNN Model
RNN Controller
Task Evaluation
Hardware Evaluation
Feedback
APNAS: Accuracy-and-Performance-Aware Neural Architecture Search
Cycle counts estimations for a CNN (section III-C)
Multi-objective reward function with tunable parameter (section III-B)

FIGURE 2. Proposed accuracy-and-performance-aware neural architecture search (APNAS) framework, where novel contributions in the blue box correspond to the red dashed boxes.

The remainder of this paper is organized as follows. First, Section II briefly reviews the fundamentals of the neural processing array-based hardware architecture and existing NAS techniques. Then, Section III presents an overview of the proposed APNAS and our analytical model for cycle count estimation of a neural processing array-based hardware architecture. Section IV evaluates the effectiveness of APNAS against manually designed CNNs and existing NAS techniques, and proposes directions for future research. Finally, Section V concludes the paper.

II. PRELIMINARIES

In this section, we provide an overview of the neural processing array-based hardware that is focused on as DNN hardware architecture in this paper, followed by recent studies NAS algorithms.

A. NEURAL PROCESSING ARRAY-BASED DNN HARDWARE ARCHITECTURE

Neural processing array-based hardware architectures are a type of domain-specific architectures that mainly aim to accelerate matrix-multiplication-based applications. The core unit, which is neural processing array (NPA), is composed of a two-dimensional network of homogenous processing elements (PEs). Each PE is composed of an adder, multiplier, and several local registers, and communicates with only its neighboring PEs/inputs. The PEs are orchestrated in a pipelined manner while each PE performs a multiply-accumulate (MAC) operation on the received data and stores the result (i.e., partial sum) in its local output register. This partial sum is then transferred to one of the neighboring PEs in the next clock cycle to be used in the subsequent MAC operation.

There are several types of NPAs depending on the dataflow exploited for processing. Two commonly used dataflows are as follows: (i) weight-stationary, where the weights are kept stationary inside the PEs for MAC operations [34]–[39]; and (ii) output-stationary, where the partial sums of the outputs are kept inside the PEs to minimize the data movement cost [40]–[42]. Weight-stationary dataflow is often adopted for recent NPAs, such as the tensor processing unit (TPU) developed by Google [25].

Our work also focuses on a weight-stationary dataflow-based NPA, such as the massively-parallel neural array (MPNA) accelerator [27], which is composed of a smaller-scale NPA suitable for embedded systems. Fig. 3

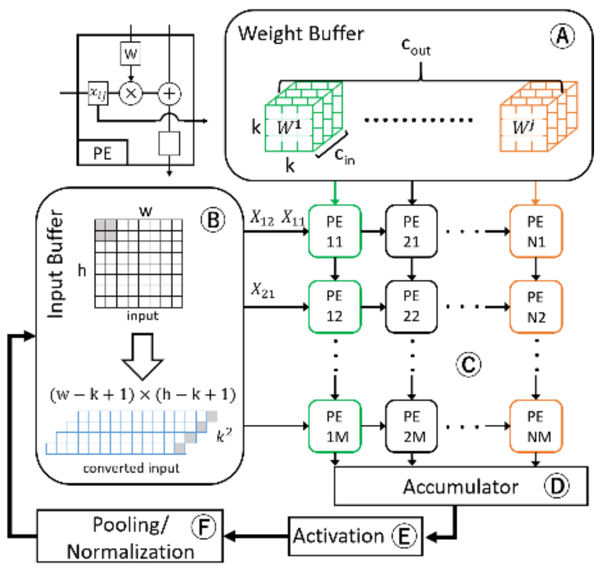


FIGURE 3. Overview of core modules of a neural processing array.

provides a brief overview of the key units constituting the accelerator: Ⓐ, the weight buffer; Ⓑ, the input buffer; Ⓒ, the $M \times N$ NPA; Ⓓ, the accumulator unit; Ⓔ, the activation unit; and Ⓕ, the pooling/normalization unit.

Here, we describe the operation of the weight-stationary NPA starting from Ⓐ. The weights are loaded in the NPA through vertical connections and are kept stationary inside the PEs for the corresponding computations, where weights from a single filter are mapped to the same column of the NPA. The number of weights in a filter is given as $k \times k \times c_{in}$, where $k \times k$ is the kernel size and c_{in} is the number of channels in the input. If the number of weights in a filter is larger than the number of rows M in the NPA, the filter is divided into multiple segments of (at most) M elements each. Note that segments from N different filters that use the same input are mapped to the NPA together to exploit parallelism.

In Ⓑ, the inputs to the PE array are first pre-processed to acquire the shape required by the NPA to perform a matrix multiplication operation. For an input of size $w \times h$ and a filter with kernel size $k \times k$, each input block of size $k \times k$ (highlighted in gray in Ⓑ) is flattened into a column of the new matrix and stacked from right to left. In other words, the rightmost column of the inputs is the first group of data that are used for processing in Ⓒ.

The inputs are then fed to Ⓒ from top to bottom in a pipelined manner. For example, the first input, X_{11}, is fed to the top-left PE11 and multiplied by the weight W_{11}^{1} in the first cycle. In the second cycle, PE21 receives PE11's output (i.e., $W_{11}^{1} \cdot X_{11}$) and the new input, X_{12}. At the same time, PE12 receives the first input (i.e., X_{11}) from PE11. Each PE performs a MAC operation individually: PE11, PE21, and PE12 perform $W_{11}^{1} \cdot X_{21}$, $W_{11}^{1} \cdot X_{11} + W_{12}^{1} \cdot X_{12}$, and $W_{11}^{2} \cdot X_{11}$, respectively. Similarly, data transfer is repeated from the top-left to right-bottom of Ⓒ. Because all of the PEs work in parallel, the same number of MAC operations can be performed in one cycle as the number of PEs.

The accumulation results of all columns (i.e., results of PEM#, where #=1, $\cdots$, N) are all accumulated in Ⓓ. Then, the activation function is performed in Ⓔ, followed by the pooling and normalization on the results in Ⓕ. The generated results of one convolutional layer of a DNN are then used as input to the next layer, and the same process is repeated for processing.

B. ENAS: EFFICIENT NEURAL ARCHITECTURE SEARCH

A variety of techniques have been studied to automate DNN design for specific classification tasks. Among various automation techniques, NAS [17] was the first work, where an RNN was used to generate DNN models and was trained with RL using a reward function. We note that here, architecture represents model to define the detailed structure and should not be confused with hardware architecture in computing platforms. In NAS, the RL reward function considers the validation accuracy of the generated model so that the RNN can learn to generate better neural networks that offer higher validation accuracy. *Although NAS provided a breakthrough in complex DNN design, it had a critical bottleneck in search time, which grew exponentially longer with an increase in the exploration parameters.*

Therefore, subsequent studies aimed to accelerate the search using various approaches. For instance, an evolutionary-algorithm-based technique repeatedly performs tournament selection from promising models, resulting in 6.67× speedup in [43]. Sequential model-based optimization learns a surrogate model to guide the search to enable 8.89× speedup in [20]. Gradient-based optimization techniques, such as Darts [19], relax the search space to be continuous, and obtained 300-500× speedup. Several techniques have also been extended from NAS that also used RL, including ENAS. Unlike NAS, ENAS reuses the weights of previously generated models to shorten the training process by representing the superposition of all generated models as a directed acyclic graph (DAG) and forcing all generated models to share weights between nodes in the DAG. ENAS successfully achieved a speedup of more than 4,000× and 3-8× relative to NAS [17] and Darts [19], respectively, with only negligible accuracy loss. Recently proposed, random search-based techniques, such as that proposed in [44], also utilize weight sharing and DAG in their search to further accelerate the search time. However, on average, these techniques can not generate a model comparable to other techniques without generating and evaluating many models (i.e., 2,000 models). *In summary, among these automation techniques, although Darts [19] may achieve the highest accuracy, ENAS is among the faster techniques with less than 1% accuracy degradation. Therefore, we leverage the underlying features of weight-sharing-based techniques as the baseline of our work. In our experiment, these features are applied to the RL-based technique (i.e., ENAS) and evaluated.* ENAS assumes that the generated model has a start block, several nodes (or hidden layers) with a skip connection, factorized reduction blocks, a global average pooling layer, and a softmax layer

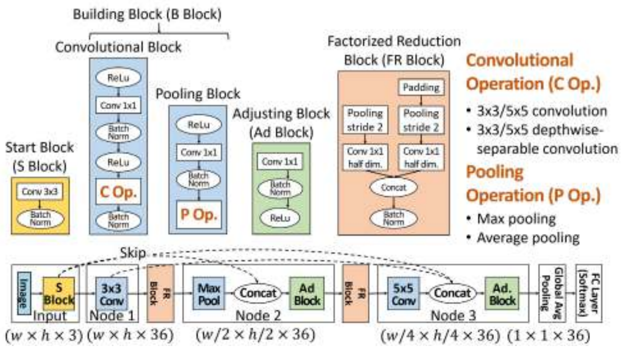


FIGURE 4. **Convolutional neural network (CNN) generated by efficient neural architecture search (ENAS). The upper part of the figure displays the main building blocks used to construct CNNs in ENAS, while the lower part illustrates the structure of an example CNN generated by ENAS.**

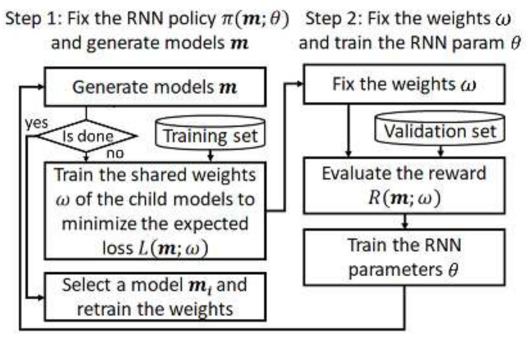

Step 1: Fix the RNN policy $\pi(\boldsymbol{m};\theta)$ and generate models $\boldsymbol{m}$
Step 2: Fix the weights ω and train the RNN param θ
Generate models $\boldsymbol{m}$
yes
Is done
no
Training set
Train the shared weights ω of the child models to minimize the expected loss $L(\boldsymbol{m};\omega)$
Select a model $\boldsymbol{m}_i$ and retrain the weights
Fix the weights ω
Validation set
Evaluate the reward $R(\boldsymbol{m};\omega)$
Train the RNN parameters θ

FIGURE 5. **Recurrent neural network (RNN) training process in efficient neural architecture search (ENAS) [18].**

(see Fig. 4). Given the number of nodes, the RNN aims to determine (i) the structure of each node, which should be composed of a building block and adjusting block; and (ii) the insertion of a skip connection. Only the building block has multiple candidates (i.e., convolutional block and pooling block) and is selectable for each node, while other blocks have only one type individually. A start block initializes the input data channels as c_{in}. In each node, a building block is placed that can be either a convolutional block ($3 \times 3.5 \times 5$ convolution or $3 \times 3.5 \times 5$ depthwise separable convolution [45]) or a pooling block (average/max pooling). An adjusting block is used to keep the number of output channels in each node equal to the number of input channels when taking a skip connection; therefore, an adjusting block may be used only in the second node and later. A factorized reduction block is used to reduce the spatial dimensions of the data by a factor of 2 to decrease the number of computations in the subsequent blocks. ENAS always specifies two factorized reduction blocks whose locations are fixed according to the total number of nodes L, that is after nodes $\lfloor L/3 \rfloor$ and $2 \times \lfloor L/3 \rfloor$.

An example model generated by ENAS with three nodes is illustrated in the lower part of Fig. 4. Initially, a start block is used to transform the input from a $w \times h$ red-green-blue (RGB) color image with three channels into a $w \times h$ matrix with 36 channels. The start block is followed by three nodes and two factorized reduction blocks between the nodes. As the number of nodes is 3 (i.e, $L = 3$), the factorized blocks are placed after nodes 1 and 2. In nodes 1, 2, and 3, the RNN places a 3×3 convolution block, max-pooling block, and 5×5 convolutional block, respectively. In nodes 2 and 3, skip connections are inserted. The inserted features in the nodes from the previous layers are concatenated with the output of the convolutional/pooling block, and the output channel size is adjusted using an adjusting block. At the end, global average pooling is used to average the activation values of all channels, and the result is then passed through a softmax layer. Note that the softmax layer here also contains a fully connected layer for performing the classification.

ENAS generates neural network models with the following two steps, as illustrated in Fig. 5: training the shared weights of the child models (Step 1), and training the RNN (Step 2). In Step 1, for a target task, neural network models m are generated based on the RNN policy $\pi(m; \theta)$, and the shared weights ω of the child models are trained to minimize the expected loss $L(m; \omega)$ through back-propagation on a training dataset. Then, in Step 2, based on those weights ω, the reward function $R(m; \omega)$ is evaluated on a validation dataset (e.g., using the accuracy on a mini-batch of validation images for image classification), and the RNN parameters θ are trained; in other words, the RNN policy $\pi(m; \theta)$ is updated. Then, the trained RNN again generates child models m. These steps are iteratively performed for a number of epochs, where a pair of Steps 1 and 2 is regarded as one epoch. Finally, a model with the highest reward m_i is selected to retrain the weights from scratch. Note that since the weights shared during the search are not optimal for the final model, ENAS does not guarantee that the selected model remains optimal after retraining. ENAS aims to enhance the efficiency of the search algorithm and does not attempt to generate an optimal model, but rather, a satisfactory one.

It should be noted that in the flow of ENAS, the reward function is set to maximize the validation accuracy only, that is, without consideration of the computational complexity of the model. However, the computational complexity is one of the most crucial parameters for resource-constrained machine learning systems, such as edge devices and smart cyber-physical systems (e.g., autonomous vehicles and robotics); therefore, ENAS cannot be used to generate neural networks that offer both near-optimal accuracy and high computational efficiency.

III. APNAS: ACCURACY-AND-PERFORMANCE-AWARE NAS

A. OVERVIEW

As mentioned earlier *lightweight* neural network models are preferable for embedded deep learning systems as they reduce computation. However, ENAS would not explore these models, as it is unaware of the computational complexity of the target device or other system constraints.

This motivated us to develop a new NAS that is aware of both the validation accuracy and computational complexity (hereafter referred to as performance). Without loss of generality, as the underlying computing platform, we consider a state-of-the-art neural processing array (MPNA [20]), which is a representative embedded DNN hardware architecture with limited PEs that supports the acceleration of both convolutional and fully connected layers with efficient data reuse. It should be noted that MPNA also supports an NPA structure similar to the one used in the Google TPU [25].

Due to the limited number of PEs, embedded DNN accelerators can compute only a small portion of a neural network at a time, as mentioned in Section II-A. Therefore, it is essential to reduce the computation (hereafter referred to as number of cycles or cycle count) on the PE array. In addition, we aim to develop an efficient NAS technique that can explore an enormous design space in a short time. Unlike state-of-the-art hardware-aware NAS techniques (e.g., [21]–[23]) that involve hardware-based execution or time-consuming hardware simulations to estimate the performance of a design candidate, in this study, we propose novel analytical models to accurately estimate the computation requirement of the NPA-based devices, thereby significantly expediting the NAS process.

To achieve the above goal, we incorporate both the validation accuracy and cycle count in the reward function during the search, the details of which are provided in Section III-B. In the MPNA, computations on the PE array dominate the total computation of the generated network model while other modules work in the background [25]; therefore, during the search, we estimate the cycle count required by the MPNA to perform all MAC computations of convolutional layers on the PE array. In other words, because we do not need to run a time-consuming hardware simulator during RNN training, in contrast to ENAS, our approach does not affect the search time. To demonstrate the effectiveness of our estimation-based approach, we discuss the estimation accuracy by comparing the estimated cycle count and actual hardware performance in Section III-D. In addition, we evaluate the search time in Section IV.

B. MULTI-OBJECTIVE REWARD FUNCTION

In APNAS, we search models that require a lower cycle count while mitigating the validation accuracy degradation as expressed in our reward function $R_{AP}(m; \omega)$ in (1):

$$R_{AP}(m; \omega) = A_m - \alpha \times C_{m\_norm} \tag{1}$$

where A_m and C_{m_norm} represent the validation accuracy and normalized cycle count of network models m, respectively, and a coefficient α is a weight to trade-off between the cycle count and accuracy. The cycle count refers to the number of cycles of all convolutional layers executed on the PE array to infer one image. Note that as described in Fig. 4, not only building blocks but also the other blocks contain a convolutional layer(s) (i.e., 3×3 or 1×1). A_m, C_{m_norm}, and α, all take the range of [0, 1].

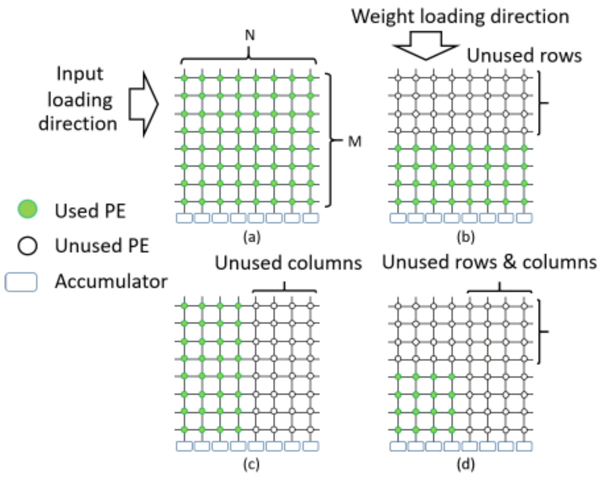


FIGURE 6. Four cases of cycle counts estimation: when weights are loaded into (a) all columns and rows (CASE 1), (b) all columns but only some rows (CASE 2), (c) all rows but only some columns (CASE 3), and (d) some columns and rows of the processing element (PE) array (CASE 4). The small rectangles at the bottom represent the accumulator in all cases.

To handle the cycle count in the same range as the accuracy, we obtain the cycle count by a min-max normalization method [46] as follows:

$$C_{m\_norm} = \frac{C_m - C_{min}}{C_{max} - C_{min}} \tag{2}$$

where C_m is the cycle count of models m, and C_{max} and C_{min} are the maximum and minimum cycle count, respectively among random generated models prior to training. In our evaluation, we used 1,000 random models.

C. CYCLE COUNT ESTIMATION FOR CNN

The cycle count on the PE array pertains mainly to two processes: weight loading ($C_{w,l}$) and MAC operations ($C_{MAC,l}$) in each convolutional layer l. Thus, (3) holds:

$$C_m = \sum_{l}(C_{w,l} + C_{MAC,l}) \tag{3}$$

1) $C_{w,l}$ COMPUTATION

When the PE array size is small, we cannot load all of the weight into the PE array at once. Instead, we must fold the excess weights and load them after all computations using the current weights are completed. Moreover, as explained in Section II-A, our target MPNA supports parallel weight loading along with the columns of the PE array. Therefore, the cycle count for weight loading, $C_{w,l}$, can be estimated as follows:

$$C_{w,l} = M \times \left\lceil \frac{(k \times k \times c_{in})}{M} \right\rceil \times \left\lceil \frac{c_{out}}{N} \right\rceil \tag{4}$$

where k represents the filter size, c_{in}/c_{out} represents the input/output channels, and M/N represents the row size/column size of the PE array.

2) CYCLE COUNT ESTIMATION FOR PERFORMING COMPUTATIONS RELATED TO A SINGLE MAPPING

For one mapping of MAC operations on the NPA, we can calculate the cycle count of the MAC operations depending on how the PE array is used. Here, we can divide the use of the PE array into four different cases, as depicted in Fig. 6. The following text explains the cycle count measurement for each case.

- **CASE 1. Using all columns and rows (Fig. 6(a)).** Because the input data are transferred from the top-left to the right-bottom of the PE array, $M+N-1$ cycles are necessary to perform all MAC operations in addition to another one cycle to feed the results to the accumulator. Additionally, it takes $C_{in}-1$ cycles to feed the unfolded inputs to the PE array (recall the process of Ⓑ in Fig. 3), where $C_{in}=(w-k+1)\times(h-k+1)$. Then, the cycle count for this case (C_1) can be estimated as follows:

$$C_1 = M + N + C_{in} - 1 \tag{5}$$

- **CASE 2. Using all columns but some rows (Fig. 6(b)).** We can regard the used PEs as a smaller PE array. This case is similar to Case 1; however, only each segment of the filter weights ($k \times k \times c_{in}$) is smaller than the row size. The cycle count reduction depends on the number of the *unused* PEs, C_{UR}:

$$C_{UR} = \left\lceil \frac{k \times k \times c_{in}}{M} \right\rceil \times M - k \times k \times c_{in} \tag{6}$$

Then, the cycle count for this case (C_2) can be estimated using C_1 (in Eq. (5)) and C_{UR} as follows:

$$C_2 = C_1 - C_{UR} \tag{7}$$

- **CASE 3. Using all rows but some columns (Fig. 6(c)).** Unlike Case 2, because the number of columns used depends on the output channels c_{out}, the number of *unused* PEs, C_{UC} can be obtained as follows:

$$C_{UC} = \left\lceil \frac{c_{out}}{N} \right\rceil \times N - c_{out} \tag{8}$$

Similarly to C_2, the cycle count for this case (C_3) can be estimated as follows:

$$C_3 = C_1 - C_{UC} \tag{9}$$

- **CASE 4. Using some columns and rows (Fig. 6(d)).** Lastly, when both columns and rows are partially used, the cycle count, C_4, can be estimated using the number of unused rows and columns (C_{UR} and C_{UC}, respectively). As mentioned in Case 2, the cycle count depends on the number of used columns and rows, not on the number of used PEs; thus, the following equation holds:

$$C_4 = C_1 - (C_{UR} + C_{UC}) \tag{10}$$

3) $C_{MAC,l}$ COMPUTATION

To perform all MAC operations in one convolutional layer, it may be necessary to fold and map the weights on the PE array iteratively if the number of operations is larger than the PE array size. Thus, depending on the computations and the PE array size, we obtain the MAC operation cycle count for each convolutional layer l by combining the above four cases, as described in Fig. 7, and the following explanations.

① If the output channels (c_{out}) exceed the column size (N) but the filter weights ($k \times k \times c_{in}$) do not exceed the row size (M), only the exceeded output channels are folded and loaded into the PE array iteratively (corresponding to CASE 2). Then, the last iteration handles the remaining, if any (corresponding to CASE 4). Thus, the calculation of $C_{MAC,l}$ is composed of the product of the number of output channel folds and C_2, which is added to C_4 (required only if CASE 4 occurs).

② Unlike ①, if the filter weights exceed the row size but the output channels do not exceed the column size, only the exceeded filter weights are folded and loaded into the PE array iteratively (corresponding to CASE 3). Then, the last iteration handles the remaining (corresponding to CASE 4). Thus, the calculation of $C_{MAC,l}$ is composed of the product of the number of filter weight folds and C_3, which is added to C_4 if necessary.

③ If the filter weights and output channels exceed the row and column sizes, respectively, both are folded and loaded into the PE array iteratively (corresponding to CASE 1). Furthermore, if only the output channels are equal to the integral multiple of the column size, CASE 2 also occurs. Thus, the calculation of $C_{MAC,l}$ is composed of the product of the number of filter weight folds, the number of output channel folds, and C_1 in the first term, which is added to ①.

④ Unlike ③, if only the filter weights are equal to the integral multiple of the row size, CASE 3 occurs instead of CASE 2. Thus, the calculation of $C_{MAC,l}$ is composed of the product of the number of filter weight folds, the number of output channel folds, and C_1, which is added to ②.

⑤ If neither the filter weights nor output channels are equal to the integral multiple of the row or column sizes, respectively, all four cases occur. Thus, the calculation of $C_{MAC,l}$ is composed of the product of the number of filter weight folds, the number of output channel folds and C_1 in the first term, which is added to ①, ②, and C_4.

⑥ Finally, if the filter weights and output channels are both equal to the integral multiple of the row and column sizes, respectively, only CASE 1 occurs. Thus, $C_{MAC,l}$ is composed of the product of the number of filter weight folds, the number of output channel folds, and C_1.

We provide two examples of parameter combination to calculate the cycle count for a convolutional layer with a PE array size of 8×8 as follows. First, let us consider a convolutional layer with an input image size of 32×32, kernel

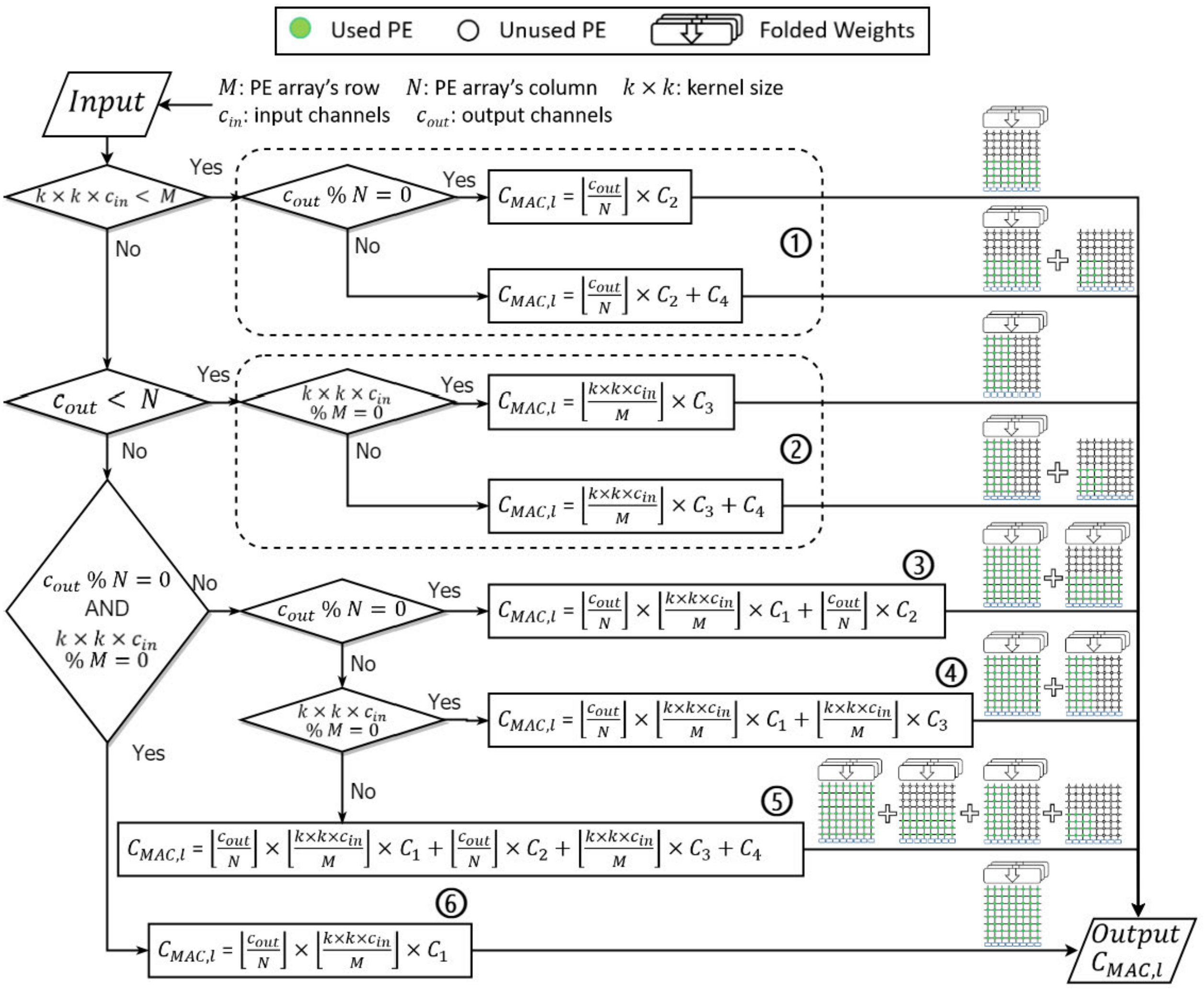

Used PE
Unused PE
Folded Weights
Input
M: PE array's row
N: PE array's column
k × k: kernel size
c_in: input channels
c_out: output channels
k × k × c_in < M
Yes
No
c_out % N = 0
C_MAC,l = ⌈c_out/N⌉ × C_2
C_MAC,l = ⌈c_out/N⌉ × C_2 + C_4
①
c_out < N
k × k × c_in % M = 0
C_MAC,l = ⌈k×k×c_in/M⌉ × C_3
C_MAC,l = ⌈k×k×c_in/M⌉ × C_3 + C_4
②
c_out % N = 0 AND k × k × c_in % M = 0
③
C_MAC,l = ⌈c_out/N⌉ × ⌈k×k×c_in/M⌉ × C_1 + ⌈c_out/N⌉ × C_2
④
C_MAC,l = ⌈c_out/N⌉ × ⌈k×k×c_in/M⌉ × C_1 + ⌈k×k×c_in/M⌉ × C_3
⑤
C_MAC,l = ⌈c_out/N⌉ × ⌈k×k×c_in/M⌉ × C_1 + ⌈c_out/N⌉ × C_2 + ⌈k×k×c_in/M⌉ × C_3 + C_4
⑥
C_MAC,l = ⌈c_out/N⌉ × ⌈k×k×c_in/M⌉ × C_1
Output C_MAC,l

FIGURE 7. Flowchart of cycle count estimation for each convolutional layer to be executed on the processing element (PE) array. In total, there are six patterns to calculate according to the amount of computation and the PE array size.

size of 3×3, input channel number of 36, and output channel number of 36. Because both the filter weights ($3 \times 3 \times 36$) and output channels (36) exceed the row and column sizes and neither are equal to the integral multiple of the row and column sizes, condition ⑤ is applied to calculate the cycle count. Next, for a convolutional layer with the same input image size, kernel size, but input channel number of 32 and output channel number of 36, because both the filter weights and output channels exceed the row and column sizes but only the filter weights are equal to the integral multiple of the row size, condition ④ is applied to calculate the cycle count.

D. CYCLE COUNT ESTIMATION VERSUS SIMULATION

To evaluate the accuracy of the proposed estimation approach presented in Section III-C, we compared the total cycle count obtained by the equations with that obtained by the hardware execution of MPNA.[2] We assumed two different PE array sizes (i.e., 8×8 and 16×16), which we also used in our evaluation in Section IV. As can be seen in Fig. 7, when the PE array size is small, conditions ① and ② are unlikely to occur, especially for large DNNs. Thus, it is sufficient to evaluate conditions ③-⑥.

Table 1 presents the results for PE array sizes 8×8 and 16×16 in the upper and lower tables, respectively. For each table, the condition, input channels (c_{in}), output channels (c_{out}), estimated cycle count (C_m) obtained by (3), and hardware simulator results (MPNA-Sim) are summarized in columns 1-5. We used the same input image size as CIFAR-10 and CIFAR-100 (i.e., 32×32) and a kernel size of 3×3 for the comparison. It should be noted that in this study, we applied zero-padding to the input of each convolutional layer to keep the number of rows and columns of the output equal to that of the input. We observed that independent of the parameter combinations and PE array size, our estimation provided the same results as the hardware simulation while avoiding time-consuming evaluation. Thus, we conclude that our esti-

[2]This is a cycle-accurate hardware simulator.

TABLE 1. Comparison of total cycle count between our estimation (Est.) and that of a hardware simulator (MPNA-Sim).

PE array size = 8×8

Conditions	c_{in}	c_{out}	Est. (C_m)	MPNA-Sim
③	36	32	188,316	188,316
④	32	36	171,692	171,692
⑤	36	36	214,451	214,451
⑥	32	32	150,768	150,768

PE array size = 16×16

Conditions	c_{in}	c_{out}	Est. (C_m)	MPNA-Sim
③	36	32	57,618	57,618
④	32	36	44,958	44,958
⑤	36	36	67,185	67,185
⑥	32	32	38,556	38,556

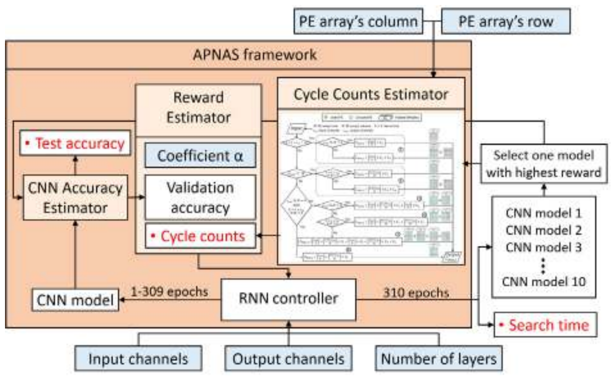


FIGURE 8. Tool flow for accuracy-and-performance-aware neural architecture search (APNAS), cycle counts estimation, multi-objective reward function and evaluation for APNAS.

mation approach is sufficient for estimating the performance of models generated by APNAS.

IV. EXPERIMENTS

This section evaluates the effectiveness of APNAS compared to state-of-the-art techniques. We first describe the experimental setup followed by the results of the proposed APNAS. By examining the results, we present useful findings and observations regarding searching neural network models for the resource-constrained MPNA.

A. EXPERIMENTAL SETUP

Fig. 8 illustrates our tool flow to evaluate the effectiveness of our proposed framework (APNAS), namely by exploring the trade-off between the validation accuracy and performance and evaluating the model search time. The orange box indicates our framework, blue boxes indicate input parameters, and boxes with red text indicate the outputs used for the evaluation.

In the first evaluation, we utilized two image classification datasets, CIFAR-10 and CIFAR-100, to evaluate APNAS against two existing works, ResNet [4] (a manually optimized neural network) and ENAS [18] (NAS that considers only validation accuracy). Because APNAS is aware of the accuracy and performance (i.e., cycle count), we evaluated these three methods in terms of the test accuracy of the image classification and the cycle count in the PE array.

- **CIFAR-10:** Dataset of 32 × 32-pixel color images that can be classified into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset contains 50,000 training images and 10,000 test images. Although validation images are not provided by the dataset, they can be easily created by using some training images for validation. In our evaluation, we used 45,000 images for training and 5,000 images for validation of the 50,000 training images.
- **CIFAR-100:** Another image classification dataset that is similar to CIFAR-10 but has 100 classes of 32 × 32-pixel color images. This dataset also contains 50,000 training images and 10,000 test images. Similarly, we used 45,000 images for training and 5,000 images for validation.

Here we describe the parameters of ENAS and APNAS, namely, those for the RNN (i.e., during model generation) and those for the target models to be generated (tabulated in Table 2). We followed [18] in the setting of epochs and the types of building blocks. In total, six different α values were set (i.e., as $\alpha = 0$ indicates ENAS, while the other five α values indicate APNAS). Two resource-constrained PE array sizes (8 × 8 and 16 × 16) were assumed. By specifying 12 nodes, both ENAS and APNAS determined the building block type and the insertion of skip connections for each node. It is also known that a larger number of filters helps achieve higher accuracy in a neural network. Therefore, we also set two different output channel numbers# for each dataset: 36 and 96 channels# for CIFAR-10, and 96 and 256 channels# for CIFAR-100. We named each model of APNAS according to the number of output channels (e.g., APNAS36, APNAS96, and APNAS256 for an output channel setting of 36, 96, and 256, respectively). In our evaluation, 10 models were generated after we continued the search process for 310 epochs, and we selected the best model (i.e., model with the highest reward) to further retrain out of the 10 models, as performed in [18].

For ResNet, we used the ResNet structure for CIFAR10 with a different number of layers to provide a comparable cycle count to ENAS for each output channel size. The detailed parameters of the ResNet models are provided in Table 3. The model was composed of a 3 × 3 convolutional layer, a stack of 6*n* 3 × 3 convolutional layers (where *n* is an integer), and a fully connected layer. In every 2*n* layers, the number of output channels was doubled as {16, 32, 64}, and the input was sub-sampled by the convolution with a stride of two. Skip connections were included in every two layers. The network ended with a global average pooling layer, a fully connected layer, and a softmax layer. We set two different settings for each dataset corresponding to each output channel size of ENAS: $n = 5, 18$ for CIFAR-10 and $n = 18, 200$ for CIFAR-100.

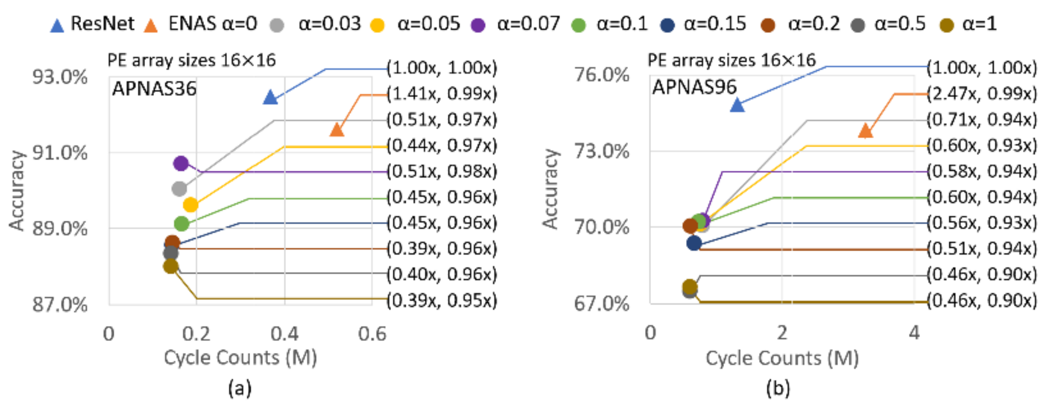


FIGURE 9. Visualization of cycle count and accuracy from five average search results: (a) for CIFAR-10, (b) for CIFAR-100 (please refer to [4] and [48] for accuracy results of ResNet on CIFAR-10 and CIFAR-100, respectively).

TABLE 2. Recurrent neural network (RNN) and model parameters for each dataset.

	Dataset	CIFAR-10	CIFAR-100
RNN	Training data#	50,000	
	Validation data#	5,000	
	Minibatches#	128, 64	
	Epochs#	310	
	α Ratio	0 (ENAS), 0.05, 0.1, 0.2, 0.5, 1	
Model	PE array	8×8, 16×16	
	Nodes#	12	
	Building block	3×3 convolution, 3×3 depth-wise convolution 5×5 convolution, 5×5 depth-wise convolution, average pooling, max pooling	
	Input & output channels	36, 96	96, 256

TABLE 3. Parameters of ResNet for each dataset.

Type of ResNet	ResNet-32	ResNet-110	ResNet-1202
Data set	CIFAR-10	CIFAR-10, CIFAR-100	CIFAR-100
n#	5	18	200
Corresponding output Channel# in ENAS	36	96	256

In the second experiment, for the CIFAR-10 image classification task, we compared the RNN runtime and the performance of the model generated by APNAS against state-of-the-art NAS techniques (i.e., NASNet, Hierarchical evolution, PNAS-5, Random search WS, DARTS, ENAS) and hardware-aware NAS techniques (i.e., AmoebaNet, PNAS-1, ProxylessNAS).

In the evaluation of the RNN runtime, APNAS was performed on a single Nvidia GTX 1080Ti GPU similar to ENAS and DARTS, while AmoebaNet, ProxylessNAS, and other techniques were performed on Tesla K40, V100, and P100 GPU, respectively. We converted GPU time on Tesla K40, V100, and P100 GPU to an equivalent GPU time on 1080Ti by multiplying by 2 as noted in [47], 1.5 as noted in [44] and 1.0 respectively. We used the total GPU days as a metric to evaluate the runtime overhead for each technique; thus, it was not necessary to distinguish between multiple and single GPU usage.

In the performance comparison, when the size of the PE array is fixed, the cycle count depends on the number of parameters. Therefore, we used the number of parameters instead of cycle count to simplify the comparison between the performance of APNAS and state-of-the-art NAS techniques. It is important to note that we included the parameters but not the cycle count of the batch normalization layer in our calculation. Nevertheless, it was reasonable to use the number of parameters to compare the performance, as there were very few parameters in batch normalization layer, and its proportion of the entire model was small (less than 4%). Among the many APNAS models, we selected the APNAS model with the highest test accuracy for comparison, where the number of channels was 36 and 96.

B. EXPERIMENTAL RESULTS

The validation accuracy and performance results among baseline models are visualized in Figs. 9, 10, and 11. In each figure, the baseline neural network models (i.e., ResNet and ENAS) are represented as triangles in the plots, while APNAS models, generated by varying the coefficient α, are represented as circles. The x-axis of each plot indicates the proportion of the cycle count, while the y-axis indicates the accuracy compared to the ResNet model. First, we describe the difference between the results in Fig. 9 and the results in Figs. 10 and 11. In Fig. 9 (a) and (b), we first intensively searched the model by adding another three values of the coefficient α into the experiments (i.e., $\alpha = 0.03, 0.07, 0.15$). We evaluated each setting five times and used the average of the results to plot the graph. We observed that each search and a specific value of α produced similar results; thus, we performed the remainder of the search in Figs. 10 and 11 with only six different values for the coefficient α, as described in Table 2 only one time per setting to speed up the experiment.

The results presented in Figs. 9, 10, and 11 demonstrate that APNAS achieved comparable accuracy to ResNet.

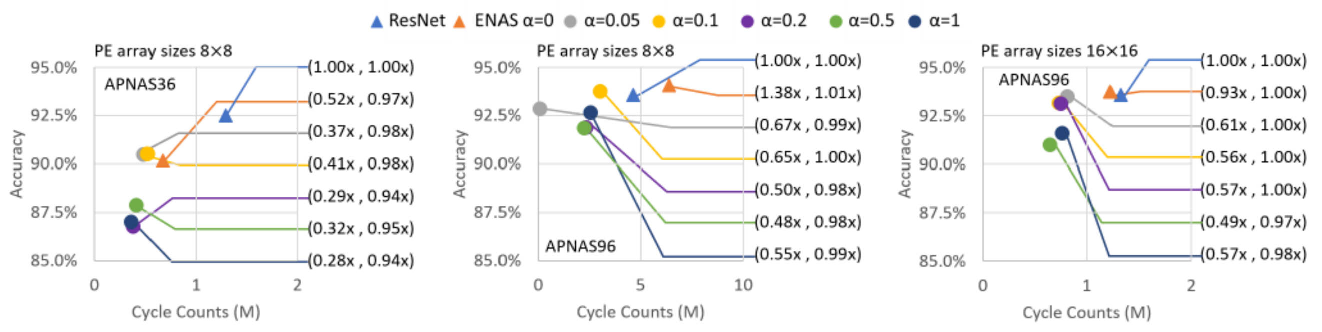


FIGURE 10. Visualization of cycle count and accuracy of image classification on CIFAR-10 dataset.

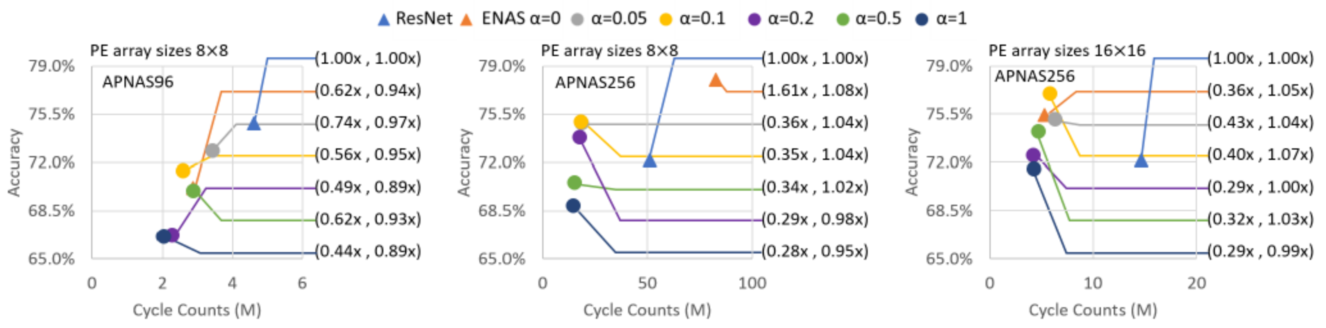


FIGURE 11. Visualization of cycle count and accuracy of image classification on CIFAR-100 dataset.

In CIFAR-10, when the number of output channels# for APNAS was 36 and the number of layers in ResNet was 32 (see Fig. 9(a) and Fig. 10), APNAS required 49% to 72% fewer cycles with negligible accuracy degradation (only 2% to 6% decrease) relative to ResNet. When the number of output channels for APNAS was 96 and the number of layers in ResNet was 110, APNAS required 33% to 52% fewer cycles at an accuracy loss of 0-3% relative to ResNet (see Fig. 10). Similar trends can be observed for CIFAR-100 in Fig. 9(b) and Fig. 11. Interestingly, for CIFAR-100, unlike for CIFAR-10, APNAS outperformed ResNet in terms of both the cycle count and test accuracy in cases in which the number of output channels# in APNAS was 256 and the number of layers in ResNet was 1,202. These results indicated that APNAS can achieve comparable or even better accuracy with significantly fewer cycles compared to manually developed state-of-the-art methods.

Furthermore, we studied the relationship between the α value and patterns of these trade-offs. Fig. 12 further presents the detailed structure of the generated models described in Fig. 9(b), where only building blocks with skip connections are described. When $\alpha = 0$ (i.e., ENAS), the RNN controller selected either a 3×3 depthwise separable convolutional layer or a 5×5 convolutional layer for each building block. Recall that a 5×5 convolutional layer has the largest number of computations among the selectable type of building blocks (the average/maximum pooling layers have the least number of computations). By increasing the α value (i.e., APNAS),

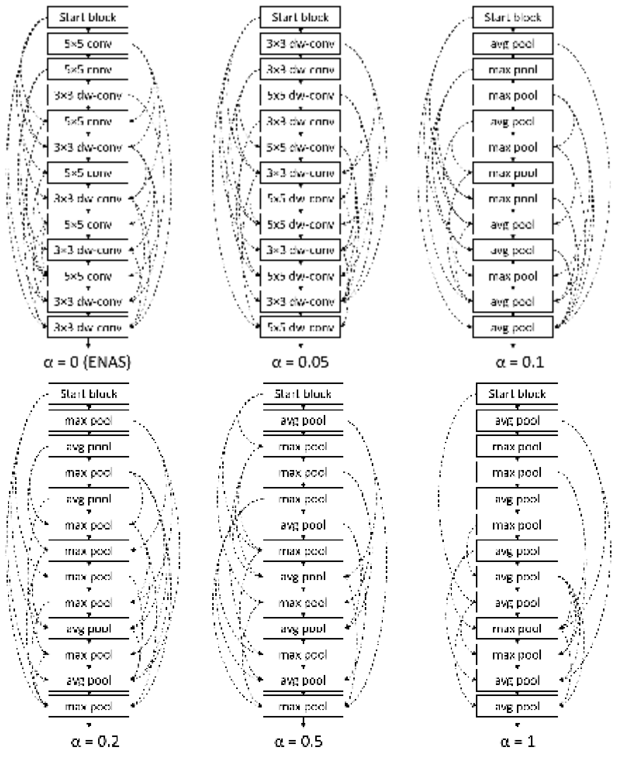


FIGURE 12. Visualization of generated models in Fig. 9(b) (i.e., 96 output channels for CIFAR-100 on a 16 × 16 processing element array).

the RNN controller tends to replace such computationally intensive layers with lighter layers in the generated models.

TABLE 4. Holistic comparison between proposed method and existing methods on CIFAR-10 dataset.

Method	Search Cost (GPU days)	Parameters	Accuracy
AmoebaNet-B*† [47]	1750	2.8M	97.45%
NASNet-A* [49]	2000	3.3M	97.35%
Hierarchical evolution [43]	300	15.7M	96.25%
PNASNet-5 [20]	225	3.2M	96.59%
PNASNet-1† [20]	N/A	1.5M	95.99%
ProxylessNAS† [23]	8	5.7M	97.92%
Random search WS [44]	2.7	4.3M	97.29%
DARTS (first order)* [19]	1.5	3.3M	97.00%
ENAS# [18]	0.45	1.1M	94.07%
APNAS96	0.55	0.7M	93.75%
APNAS36	0.55	0.1M	90.53%

* with cutout
† hardware-aware neural architecture search technique
# obtained using code publicly released by the authors

Consequently, APNAS used only $5 \times 5.3 \times 3$ depthwise separable layers and used only average/max pooling layers when $\alpha = 0.05$ and $\alpha = 0.1$, respectively. After $\alpha = 0.2$ and later, to further reduce the number of computations, APNAS reduced the number of skip connections. Although visualizations for the other figures are omitted from this paper, we also observed similar results by changing the parameters. From these analyses, we determined that APNAS successfully reduced the cycle count by adjusting the type of selected building blocks and skip connections according to the α value. In future work, we will further investigate the influences on the α value leading to saturation of the trade-offs.

We present a comparison between state-of-the-art NAS methods and APNAS in Table 4 in terms of the search time and parameters. Despite the fact that our GPU had the lowest computation capability among the other search methods in Table 4, APNAS required only 0.55 GPU days to find models in APNAS, which was comparable to ENAS [18] and faster than the remaining state-of-the-art NAS techniques. We can also see that when we searched for the model with 96 channels, compared to the state-of-the-art NAS techniques, APNAS found the model with (36% to 96%) fewer parameters (i.e., cycle count) and relatively high accuracy. When we performed the search for the model with 36 channels, the generated model contained much fewer parameters (91% to 99%) with only a small accuracy decrease (approximately 3% less than APNAS with 96 channels). We note that there are techniques to improve accuracy, such as preprocessing the training data by a cutout technique, as performed in [19], [47], [49]. Thus, it is possible to further improve the accuracy of the current APNAS without affecting the cycle count.

C. DISCUSSION

As discussed above, in APNAS, the α value can create and control the trade-off between the test accuracy and cycle count. However, in some cases, the cycle count when $\alpha = 0.5$ or 1 is higher than the cycle count when $\alpha = 0.2$. There are two reasons for this. The first reason involves the nature of the RNN controller. Although an RNN usually explores better models based on the current model in training, it may degrade the models by greatly changing the model structure due to the uncertainty contained in the RNN [50]. To mitigate this uncertainty, APNAS generates 10 models and selects a single best model to further retrain its weights, as performed in [18]. Although this approach can mitigate the impact of the RNN's uncertainty, generating 10 models may not yet be sufficient. In fact, the authors of [18] suggested improving model selection by re-training all of the generated models and selecting only the best model as performed in other studies [17], [20], [43], [49] at the cost of exploration time.

The second reason involves the normalized cycle count of the reward function in APNAS. As mentioned, we normalized the cycle count of the current model (C_m) by the maximum and minimum count (C_{max} and C_{min}) among 1,000 models that were initially generated at random due to the enormous design space. APNAS then attempted to reduce the cycle count C_m only down to C_{min}. This signifies that if there was no light lightweight computation model among the first 1,000 randomly generated models (i.e., relatively large C_{min}), then APNAS did not attempt to explore the models with lower cycle count. Moreover, if C_m was sufficiently close to C_{min}, the second term of the reward function was nearly zero, resulting in almost no consideration of the cycle count. This situation is likely to occor when C_{min} is relatively large, and there are two potential methods of resolving this problem. One is to utilize the absolute cycle count rather than the normalized one by tuning only the coefficient values (i.e., α). However, determining how to adjust the coefficient values is a difficult task, especially when multiple objectives are considered in the reward function. The appropriate setting may also vary according to the target dataset, underlying architecture, and model parameters. The second potential solution is to change the baseline results for the normalization. For example, we can use the cycle count of the initially generated models (i.e., at epoch 0) as the baseline and attempt to improve the reduction rate from them. In addition to handling the second term, we can also consider how to integrate multiple objectives in a single reward function, referring to a number of design space exploration studies [21]–[23], [51]–[53].

To further improve the trade-off between the accuracy and cycle count, we can consider exploiting an accuracy-enhancement technique that has been used in several manually constructed neural networks [4], [54], [55]. While the input channels and output channels are the same, this enhancement technique doubles the number of output channels# from the number of input channels# while halving both the height and width of the input, leading to halve the computation amount. For example, if the height, width, and number of input channels# of a layer are h, w, and c_{in}, respectively, the height, width, and number of output channels# are $h/2$, $w/2$, and $2 \times c_{in}$ $(=c_{out})$. This change in the number of channels# is adopted at some pre-determined interval (e.g., in ResNet), the entire network is divided into three chunks,

TABLE 5. Comparison between current and extended model when $\alpha = 1$ and processing element array = 8 × 8.

Database	Model	Output channel#	Cycle#	Accuracy
CIFAR-10	APNAS36	36	386,679	87.02%
	Ext.	18	372,306	87.68%
	APNAS96	96	2,541,996	92.67%
	Ext.	48	2,329,914	92.60%
CIFAR-100	APNAS96	96	2,021,388	66.67%
	Ext.	48	2,100,522	67.62%
	APNAS256	256	14,258,464	68.88%
	Ext.	128	12,285,584	69.42%

and this technique is applied to the last layer of each chunk. Through this technique, useful features can be extracted and preserved in the newly added channels while reducing the cycle count. Straightforward adoption of this technique will, however, increase the computation. Therefore, we can start from the halved number of channels# and apply this technique at the last layer of three chunks similarly to ResNet. The current and extended models, $c_{current}$ and $c_{ext.}$, have the following computation amount if the same computation amount ($c_{out} = 36$) is specified for the output channels in the middle chunk:

$$C_{current} = 36wh + \frac{36}{4}wh + \frac{36}{16}wh = 47.25wh$$
$$C_{ext.} = 18wh + \frac{36}{4}wh + \frac{72}{16}wh = 31.50wh$$

Note that as mentioned, we start 18 channels for the first chunk in the extended approach. By comparing these theoretical estimations, we expect that in the extended approach, the total computation amount (and thus the cycle count) can be effectively reduced. It should be noted that the effects of skip connections are not considered in these estimates.

To evaluate the effects of this technique, we evaluated the accuracy and cycle count for the case $\alpha = 1$ on an 8×8 PE array. In Table 5, APNAS repeats the results presented in Figs. 10 and 11. Based on the models explored by APNAS, we manually applied changes to the number of channels# only (Ext. in the table indicates a counterpart of each model). The table demonstrates that despite starting with a halved number of channels#, the extended models achieved comparable or better accuracy than the explored models. Furthermore, as discussed above with the estimations, the extended models successfully reduced the cycle count, except for the case of 48 channels for CIFAR-100. We found that this increase in cycle count was caused by skip connections. Because skip connections increased the computation amount but the change in the number of output channels# was not considered during exploration, this manual adoption coincidentally increased the cycle count. By allowing APNAS to consider this technique, the skip connections are expected to be appropriately inserted to suppress the cycle counts, which will be considered in future work.

Other directions for future work include the relaxation of the assumptions on the model structure and the adoption

TABLE 6. Abbreviations used in this paper.

Abbreviations	Meaning
APNAS	Accuracy-and-Performance-Aware Neural Architecture Search
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DNN	Deep Neural Network
EA	Evolutionary Algorithm
ENAS	Efficient Neural Architecture Search
GPU	Graphics Processing Unit
MAC	Multiply–Accumulate
MPNA	Massively-Parallel Neural Array
NAS	Neural Architecture Search
NPA	Neural Processing Array
PE	Processing Element
RGB	Red, Green, and Blue
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SMBO	Sequential Model-Based Optimization
TPU	Tensor Processing Unit

TABLE 7. Symbols used in this paper.

Symbols	Meaning
w	Width of an image
h	Height of an image
k	Size of a filter in a convolutional layer
c_{in}	Number of input channels in a convolutional layer
c_{out}	Number of output channels in a convolutional layer
X	Input data of a convolutional layer
W^i	Weights of a convolutional layer in channel i
M	Row size in a neural processing array
N	Column size in a neural processing array
L	Number of nodes in NAS
$\pi()$	Policy of the recurrent neural network
m	Neural network models
θ	Weights of a recurrent neural network
ω	Weights of a generated model
α	Coefficient value to trade-off accuracy and performance in APNAS
$R()$	Reward function of NAS
$R_{AP}()$	Reward function of APNAS
$L()$	Loss function of a generated model
C_{norm}	Normalized cycle count of a generated model
C_m	Total cycle count of a generated model
$C_{w,l}$	Cycle count for loading weights in layer l
$C_{MAC,l}$	Cycle count for performing MAC operations in layer l
C_{UR}	Number of unused rows in a neural processing array
C_{UC}	Number of unused columns in a neural processing array
C_{in}	Cycle count needs for feeding inputs to a neural processing array

of the newly created methods from the inheritor of NAS. Although the number of nodes# and output channels# is currently fixed in this work, their pre-determined settings may be excessive for some databases, leading to an unnecessarily large cycle count and overfitting [56]. In addition, we may miss the opportunity to explore smaller models with comparable accuracy. We thus expect that relaxing these constraints can enhance the flexibility and usefulness of APNAS. Furthermore, as mentioned above, there are data augmentation techniques, such as cutout, that can help increase the accuracy of the model with out affecting the performance. Moreover, there are other similar methods to search for DNN models, such as sequential model-based optimization, gradient descent, and evolutionary methods [19], [20], [57], that may boost the effectiveness of APNAS.

V. CONCLUSIONS

In recent years, NAS has become an attractive method for automatically searching useful neural network models by an RNN and RL. Although there are a number of studies on accuracy-aware NAS, to the best of our knowledge, no studies have considered performance (or cycle counts) to perform inference. Moreover, no studies have utilized a method to evaluate the performance without hardware devices or simulators. Therefore, targeting neural-network-based applications executed on resource-constrained neural processing array-based architectures of embedded systems, this study proposes an accuracy-and-performance-aware NAS (APNAS). We first present a method for estimating the cycle count in an RNN so that a time-consuming hardware simulator does not need to be run during the network search. We then define a reward function in the RL to trade-off the accuracy and performance (i.e., cycle count). Our evaluation, demonstrated that APNAS can generate neural network models with only 0.55 GPU days on an Nvidia GTX 1080Ti GPU and obtain an average of 53% fewer cycles against a manually developed neural network model (ResNet) and a state-of-the-art NAS considering only validation accuracy (ENAS) for the CIFAR-10 and the CIFAR-100 datasets.

In addition, unlike NAS, which is unaware of the cycle count, APNAS successfully trades off the accuracy and cycle count by varying the weight for considering the cycle count in the RNN. In this paper, we also discuss the limitations of the current APNAS and present useful observations and findings to achieve both accuracy improvement and cycle count reduction in future work.

APPENDIX

SYMBOLS AND ABBREVIATIONS

The following tables describe the symbols and abbreviations used in this paper.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Müller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*. [Online]. Available: http://arxiv.org/abs/1604.07316

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[6] S. H. I. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 802–810.

[7] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique, "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2019, pp. 553–559.

[8] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Des. Test.*, vol. 37, no. 2, pp. 30–57, Apr. 2020.

[9] J. J. Zhang, K. Liu, F. Khalid, M. A. Hanif, S. Rehman, T. Theocharides, A. Artussi, M. Shafique, and S. Garg, "Building robust machine learning systems: Current progress, research challenges, and opportunities," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–4.

[10] M. Shafique, T. Theocharides, C.-S. Bouganis, M. A. Hanif, F. Khalid, R. Hafiz, and S. Rehman, "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, Mar. 2018, pp. 827–832.

[11] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: Review, opportunities and challenges," *Briefings Bioinf.*, vol. 19, no. 6, pp. 1236–1246, Nov. 2018.

[12] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature Med.*, vol. 25, no. 1, p. 24, 2019.

[13] B. S. Prabakaran, A. G. Jiménez, G. M. Martínez, and M. Shafique, "EMAP: A cloud-edge hybrid framework for EEG monitoring and cross-correlation based real-time anomaly prediction," 2020, *arXiv:2004.10491*. [Online]. Available: http://arxiv.org/abs/2004.10491

[14] F. Khalid, H. Ali, H. Tariq, M. A. Hanif, S. Rehman, R. Ahmed, and M. Shafique, "QuSecNets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks," in *Proc. IEEE 25th Int. Symp. On-Line Test. Robust Syst. Design*, Jul. 2019, pp. 182–187.

[15] A. Marchisio, M. A. Hanif, M. Martina, and M. Shafique, "PruNet: Class-blind pruning method for deep neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2018, pp. 1–8.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[17] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*. [Online]. Available: http://arxiv.org/abs/1611.01578

[18] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, *arXiv:1802.03268*. [Online]. Available: http://arxiv.org/abs/1802.03268

[19] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*. [Online]. Available: http://arxiv.org/abs/1806.09055

[20] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–34.

[21] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 10734–10742.

[22] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 2820–2828.

[23] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*. [Online]. Available: http://arxiv.org/abs/1812.00332

[24] A. A. Awan, H. Subramoni, and D. K. Panda, "An in-depth performance characterization of CPU-and GPU-based DNN training on modern architectures," in *Proc. Mach. Learn. HPC Environ.*, 2017, p. 8.

[25] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, Jun. 2017.

[26] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[27] M. A. Hanif, R. V. W. Putra, M. Tanvir, R. Hafiz, S. Rehman, and M. Shafique, "MPNA: A massively-parallel neural array accelerator with dataflow optimization for convolutional neural networks," *CoRR*, vol. abs/1810.12910, pp. 1–8, Oct. 2018.

[28] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2017, pp. 553–564.

[29] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. Annu. Int. Symp. Comput. Archit.*, 2017, pp. 27–40.

[30] G. von Zitzewitz, "Survey of neural networks in autonomous driving," Advanced Seminar Summer Semester, Technische Univ. Munchen, Munich, Germany, Tech. Rep., Jul. 2017.

[31] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation," *Artif. Intell. Rev.*, vol. 52, no. 2, pp. 1089–1106, Aug. 2019.

[32] Q. Abbas, M. E. A. Ibrahim, and M. A. Jaffar, "A comprehensive review of recent advances on deep vision systems," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 39–76, Jun. 2019.

[33] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, Jan. 2019.

[34] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2009, pp. 53–60.

[35] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, "Towards an embedded biologically-inspired machine vision processor," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2010, pp. 273–278.

[36] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 247–257, Jun. 2010.

[37] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2014, pp. 682–687.

[38] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "4.6 A 1.93 TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 1–3.

[39] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proc. 25th Ed. Great Lakes Symp. VLSI*, 2015, pp. 199–204.

[40] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[41] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.

[42] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Design*, Oct. 2013, pp. 13–19.

[43] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017, *arXiv:1711.00436*. [Online]. Available: http://arxiv.org/abs/1711.00436

[44] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," 2019, *arXiv:1902.07638*. [Online]. Available: http://arxiv.org/abs/1902.07638

[45] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1251–1258.

[46] A. Jain, K. Nandakumar, and A. Ross, "Score normalization in multimodal biometric systems," *Pattern Recognit.*, vol. 38, no. 12, pp. 2270–2285, Dec. 2005.

[47] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[48] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*. [Online]. Available: http://arxiv.org/abs/1605.07146

[49] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[50] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, Dept. Eng., Univ. Cambridge, Cambridge, U.K., 2016.

[51] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "MorphNet: Fast & simple resource-constrained structure learning of deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1586–1595.

[52] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 285–300.

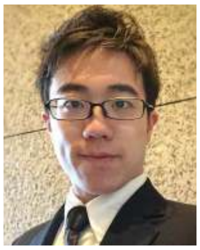
[53] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[55] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[56] D. M. Hawkins, "The problem of overfitting," *J. Chem. Inf. Comput. Sci.*, vol. 44, no. 1, pp. 1–12, Jan. 2004.

[57] T. Elsken, J. Hendrik Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," 2018, *arXiv:1804.09081*. [Online]. Available: http://arxiv.org/abs/1804.09081



PANITI ACHARARIT received the bachelor's and master's degrees in engineering from the Tokyo Institute of Technology, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree with the Department of Information and Communication Engineering, School of Engineering, under the supervision of Assoc. Prof. Yuko Hara-Azumi. He was an Internship Engineer with the Toshiba Corporate Research and Development Center, in 2017, and a Visiting Student at with the Vienna University of Technology (TU Wien), in 2018. His research interests include embedded systems hardware for machine learning, artificial intelligence (AI), and neuromorphic computing. He was a recipient of the Royal Thai government scholarship allocated upon Ministry of Science and Technology, Thailand for his capabilities.

MUHAMMAD ABDULLAH HANIF (Graduate Student Member, IEEE) received the B.Sc. degree in electronic engineering from GIKI, Pakistan, and the M.Sc. degree in electrical engineering with specialization in digital systems and signal processing (DSSP) from the School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan. He is currently a University Assistant with the Vienna University of Technology (TU Wien), Austria. In the past, he has also worked as a Research Associate with the Vision Processing (VISpro) lab, Information Technology University (ITU), Pakistan, and as a Lab Engineer with the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI), Pakistan. He was a recipient of President's Gold Medal for his outstanding academic performance during his M.S. degree.

RACHMAD VIDYA WICAKSANA PUTRA received the B.Sc. degree in electrical engineering and the M.Sc. degree (Hons.) in electronics engineering from the Bandung Institute of Technology (ITB), Indonesia, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.) at the Embedded Computing Systems Group, Institute of Computer Engineering, TU Wien, under the supervision of Prof. Dr. Muhammad Shafique. He is also a Research Assistant with the Embedded Computing Systems Group, Institute of Computer Engineering, TU Wien. He was also with ITB as a Teaching Assistant with the Electrical Engineering Department, School of Electrical Engineering and Informatics (SEEI), from 2012 to 2017, and as a Research Assistant with the University Center of Excellence, Microelectronics Center ITB, from 2014 to 2017. His research interests mainly include computer architecture, VLSI design, system-on-chip, brain-inspired and neuromorphic computing, emerging computing paradigms and technologies. He was a recipient of the Indonesian Endowment Fund for Education (IEFE/LPDP) Scholarship from Ministry of Finance, Republic of Indonesia.

MUHAMMAD SHAFIQUE (Senior Member, IEEE) received the Ph.D. degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2011. Afterwards, he established and led a highly recognized research group at KIT for several years as well as conducted impactful Research and Development activities in Pakistan. In October 2016, he joined the Institute of Computer Engineering at the Faculty of Informatics, Technische Universität Wien (TU Wien), Vienna, Austria, as a Full Professor of computer architecture and robust, energy-efficient technologies. Since September 2020, he is with the Division of Engineering, New York University Abu Dhabi (NYU AD), United Arab Emirates. He has given several Keynotes, Invited Talks, and Tutorials, as well as organized many special sessions at premier venues. He has served as the PC Chair, General Chair, Track Chair, and PC member for several prestigious IEEE/ACM conferences. He holds one U.S. patent has coauthored six books, more than 10 book chapters, and over 250 articles in premier journals and conferences. His research interests are in brain-inspired computing, AI and machine learning hardware and system-level design, energy-efficient systems, robust computing, hardware security, emerging technologies, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer analysis, modeling, design, and optimization of computing and memory systems. The researched technologies and tools are deployed in application use cases from Internet-of-Things (IoT), smart Cyber-Physical Systems (CPS), and ICT for Development (ICT4D) domains.

Dr. Shafique received the 2015 ACM/SIGDA Outstanding New Faculty Award, AI 2000 Chip Technology Most Influential Scholar Award, in 2020, six gold medals, and several best paper awards and nominations at prestigious conferences.

YUKO HARA-AZUMI (Member, IEEE) received the Ph.D. degree in information science from Nagoya University, in 2010. She was a JSPS Postdoctoral Research Fellow with Ritsumeikan University, from 2010 to 2012, during which she was also a Visiting Scholar with the University of California, Irvine, USA, and Karlsruhe Institute of Technology, Germany. In 2012, she joined the Nara Institute of Science and Technology as an Assistant Professor. Since 2014, she has been with the Department of Information and Communications Engineering, School of Engineering, Tokyo Institute of Technology, where she is currently an Associate Professor. Her research interests include system-level microprocessor design and design automation technology, especially on high-level and logic synthesis and hardware/software co-design, for the embedded/Internet-of-Things (IoT) systems. She currently serves as organizing and program committees of numerous premier conferences including DAC, ICCAD, DATE, ASP-DAC, CASES, and so on. She is a member of ACM.